Package 'Require'

September 4, 2026

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 2026-09-03

Version 2.1.1

Depends R (>= 4.1)

Imports callr, data.table (>= 1.10.4), methods, pak, processx, sys, tools, utils

Suggests cli, covr, curl, diffobj, fpCompare, gitcreds, httr, parallel, pkgcache, pkgload, rematch2, rmarkdown, knitr, rlang, roxygen2, rprojroot, testthat (>= 3.0.0), tibble, waldo, withr

Encoding UTF-8

Language en-CA

License GPL-3

VignetteBuilder knitr, rmarkdown

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

ByteCompile yes

Config/roxygen2/version 8.1.0

Collate 'CRAN.R' 'Require-helpers.R' 'Require-package.R' 'messages.R' 'Require2.R' 'RequireOptions.R' 'envs.R' 'extract.R' 'helpers.R' 'pak.R' 'pkgDep.R' 'pkgDep3.R' 'pkgSnapshot.R' 'setLibPaths.R' 'setup.R' 'zzz.R'

Config/testthat/edition 3

NeedsCompilation no

Author Eliot J B McIntire [aut, cre] (ORCID: <https://orcid.org/0000-0002-6914-8316>), Alex M Chubaty [ctb] (ORCID: <https://orcid.org/0000-0001-7146-8135>), His Majesty the King 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 2026-09-04 05:10:59 UTC

Contents

`Require-package` *Require: 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.

This is an "all in one" function that will run `install.packages` for CRAN and GitHub 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(INSTALL_opts = "--no-multiarch"),
  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),
  type = getOption("pkgType"),
  typeExplicit = !missing(type),
  upgrade = FALSE,
  returnDetails = FALSE,
  ...
)

Install(
  packages,
  packageVersionFile,
  libPaths,
  install_githubArgs = list(),
  install.packagesArgs = list(INSTALL_opts = "--no-multiarch"),
  standAlone = getOption("Require.standAlone", FALSE),
  install = TRUE,
  repos = getOption("repos"),
  purge = getOption("Require.purge", FALSE),
  verbose = getOption("Require.verbose", FALSE),
  type = getOption("pkgType"),
  typeExplicit = !missing(type),
  upgrade = FALSE,
  ...
)
```

Arguments

packages	Either a character vector of packages to install via `install.packages`, then load (i.e., with `library`), or, for convenience, a vector or list (using `c` or `list`) of unquoted package names to install and/or load (as in `require`, but vectorized). Passing vectors of names may not work in all cases, so user should confirm before relying on this behaviour in operational 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. As a convenience for copy-pasting long lists, any character element that

contains newlines is split into one package per line, with whitespace trimmed and blank or #-prefixed lines dropped. For example, `Require("\n dplyr\n # ggplot2\n PredictiveEcology/LandR@development\n")` is equivalent to `Require(c("dplyr", "PredictiveEcology/LandR@development"))`. Alternatively, an unquoted `{...}` block is accepted, with each line treated as one package spec, e.g. `Require({ dplyr; lme4; PredictiveEcology/LandR@development })`. Note that R's parser strips comments inside `{...}` before this function runs, so to omit a package delete the line (or comment it out – both have the same effect). Version constraints such as `pkg (>= 1.0)` do not parse inside `{...}` and must use the quoted string form.

`packageVersionFile`
: Character string of a file name or logical. If `TRUE`, then this function will load the default file, `getOption("Require.packageVersionFile")`. If this argument is provided, then this will override any packages passed to `packages`. By default, `Require` will attempt to resolve dependency violations (i.e., if this `packageVersionFile` specifies a version of a package that violates the dependency specification of another package). If a user wishes to attempt to install the `packageVersionFile` without assessing the dependencies, set `dependencies = FALSE`.

`libPaths`
: The library path (or libraries) where all packages should be installed, and looked for to load (i.e., call `library`). This can be used to create isolated, stand alone package installations, if used with `standAlone = TRUE`. Currently, the path supplied here will be prepended to `.libPaths()` (temporarily during this call) to `Require` if `standAlone = FALSE` or will set (temporarily) `.libPaths()` to `c(libPaths, tail(libPaths(), 1)` to keep base packages.

`install_githubArgs`
: Deprecated. Values passed here are merged with `install.packagesArgs`, with the `install.packagesArgs` taking precedence if conflicting.

`install.packagesArgs`
: List of optional named arguments, passed to `install.packages`. Default is only `--no-multi-arch`, meaning that only the current architecture will be built and installed (e.g., 64 bit, not 32 bit, in many cases).

`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.

`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 e.g., `install.packages`. If `TRUE`, the default, then this function will try to install any missing packages or dependencies.

`require`
: Logical or character string. If `TRUE`, the default, then the function will attempt to call `require` on all requested `packages`, possibly after they are installed. If a character string, then it will only call `require` on those specific packages (i.e., it will install the ones listed in `packages`, but load the packages listed in `require`)

repos — The remote repository (e.g., a CRAN mirror), passed to either `install.packages`, `install_github` or `installVersions`. **When** `options(Require.usePak = TRUE)`**:** `repos` is added to pak's repository list via `options(repos)`. However, pak always includes CRAN and Bioconductor as built-in defaults regardless of this setting – `repos` can only *add* sources, it cannot prevent pak from also searching CRAN. This differs from the default (`usePak = FALSE`) behaviour where `repos` strictly controls which repositories are used. Use `pak::cache_clean()` to clear pak's download cache if needed.

purge — Logical. Should all caches be purged? Default is `getOption("Require.purge", FALSE)`. 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 `R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE`, or if unset, defaults to 3600 (one hour – see `utils::available.packages`).

Internally, there are calls to `available.packages`.

verbose — Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if `1` or TRUE, more outputs; `2` even more. NOTE: in `Require` function, when `verbose >= 2`, also returns details as if `returnDetails = TRUE` (for backwards compatibility).

type — See `utils::install.packages`

typeExplicit — Logical. Whether `type` was supplied by the caller rather than defaulted from `getOption("pkgType")`. Defaults to `!missing(type)` and should not normally be set by hand. It exists because `getOption("pkgType")` is `"source"` on Linux, so the value of `type` alone cannot say whether source-only builds were actually asked for; only an explicit request pins pak to source.

upgrade — When `FALSE`, the default, will only upgrade a package when the version on in the local library is not adequate for the version requirements of the `packages`. Note: for convenience, `update` can be used for this argument.

returnDetails — Logical. If `TRUE` the return object will have an attribute: `attr(.., "Require")` which has lots of information about the processes of the installs.

... — 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.

Details

Require declares hard `Imports` on `callr` and `processx` to ensure they are installed alongside Require. They are not called directly from Require's R code; they are runtime dependencies of `pak`'s background `r_session` subprocess. pak's CRAN binary ships embedded copies of these helpers but does not declare them as standalone `Imports`, and on some Windows configurations pak's subprocess wedges unless standalone `processx` / `callr` are also visible in `.libPaths()` (symptom: every per-ref `pak::pak()` call fails with an empty reason string and no pak progress output). Declaring them in Require's Imports guarantees they're present wherever Require is installed.

`Install` is the same as `Require(..., require = FALSE)`, for convenience.

Value

`Require` is intended to replace `base::require`, thus it returns a logical, named vector indicating whether the named packages have been loaded. Because `Require` also has the ability to install packages, a return value of `FALSE` does not mean that it did not install correctly; rather, it means it did not attach with `require`, which could be because it did not install correctly, or also because e.g., `require = FALSE`.

`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.cachePkgDir")` 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.cachePkgDir" = 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)

Authors:

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

Other contributors:

- Alex M Chubaty <achubaty@for-cast.ca> (ORCID) [contributor]
- His Majesty the King 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

```
opts <- Require:::.setupExample()

library(Require)
getCRANrepos(ind = 1)
Require("utils") # analogous to require(stats), but it checks for
#   pkg dependencies, and installs them, if missing

# unquoted version
Require(c(tools, utils))

# Install in a new local library (libPaths)
tempPkgFolder <- file.path(tempdir(), "Require/Packages")
# use standAlone, means it will put it in libPaths, even if it already exists
#   in another local library (e.g., personal library)
Install("crayon", libPaths = tempPkgFolder, standAlone = TRUE)

# Mutual dependencies, only installs once -- e.g., cli
tempPkgFolder <- file.path(tempdir(), "Require/Packages")
Install(c("cli", "R6"), libPaths = tempPkgFolder, standAlone = TRUE)

# Mutual dependencies, only installs once -- e.g., rlang
tempPkgFolder <- file.path(tempdir(), "Require/Packages")
Install(c("rlang", "ellipsis"), libPaths = tempPkgFolder, standAlone = TRUE)

################################################################################
# Isolated projects -- Use a project folder and pass to libPaths or set .libPaths() #
################################################################################
# GitHub packages
if (requireNamespace("gitcreds", quietly = TRUE)) {
  # if (is(try(gitcreds::gitcreds_get(), silent = TRUE), "gitcreds")) {
  ProjectPackageFolder <- file.path(tempdir(), "Require/ProjectA")
  if (requireNamespace("curl")) {
    Require("PredictiveEcology/fpCompare@development",
      libPaths = ProjectPackageFolder,
```

```
  )
}

# No install because it is there already
Install("PredictiveEcology/fpCompare@development",
  libPaths = ProjectPackageFolder,
) # the latest version on GitHub

################################################################################
# Mixing and matching GitHub, CRAN, with and without version numbering
################################################################################
pkgs <- c(
  "remotes (<=2.4.1)", # old version
  "digest (>= 0.6.28)", # recent version
 "PredictiveEcology/fpCompare@a0260b8476b06628bba0ae73af3430cce9620ca0" # exact version
)
Require::Require(pkgs, libPaths = ProjectPackageFolder)
# }
}
Require:::.cleanup(opts)
```

`.downloadFileMasterMainAuth` *GITHUB_PAT-aware and* `main-master`*-aware download from GitHub*

Description

Equivalent to `utils::download.file`, but taking the `GITHUB_PAT` environment variable and using it to access the Github url.

Usage

```
.downloadFileMasterMainAuth(
  url,
  destfile,
  need = "HEAD",
  verbose = getOption("Require.verbose"),
  verboseLevel = 2
)
```

Arguments

`url`	a `character` string (or longer vector for the `"libcurl"` method) naming the URL of a resource to be downloaded.
`destfile`	a character string (or vector, see the `url` argument) with the file path where the downloaded file is to be saved. Tilde-expansion is performed.

need	If specified, user can suggest which master or main or HEAD to try first. If unspecified, HEAD is used.
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).
verboseLevel	A numeric indicating what verbose threshold (level) above which this message will show.

Value

This is called for its side effect, namely, the same as utils::download.file, but using a GITHUB_PAT, it if is in the environment, and trying both master and main if the actual url specifies either master or main and it does not exist.

.installed.pkgs *Partial alternative (faster) to* installed.packages

---

Description

This reads the DESCRIPTION files only, so can only access fields that are available in the DESCRIPTION file. This is different than installed.packages which has many other fields, like "Built", "NeedsCompilation" etc. If those fields are needed, then this function will return an empty column in the returned character matrix.

Usage

```
.installed.pkgs(
  lib.loc = .libPaths(),
  which = c("Depends", "Imports", "LinkingTo"),
  other = NULL,
  purge = getOption("Require.purge", FALSE),
  packages = NULL,
  collapse = FALSE
)
```

Arguments

lib.loc	character vector describing the location of R library trees to search through, or NULL for all known trees (see .libPaths).
which	a character vector listing the types of dependencies, a subset of c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances"). Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances".

other	Can supply other fields; the only benefit here is that a user can specify "github" (lower case) and it will automatically add c("GithubRepo", "GithubUsername", "GithubRef", "GithubSHA1", "GithubSubFolder") fields
purge	Logical. Should all caches be purged? Default is getOption("Require.purge", FALSE). 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 R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE, or if unset, defaults to 3600 (one hour – see utils::available.packages). Internally, there are calls to available.packages.
packages	Character vector. If NULL (default), then all installed packages are searched for. If a character vector is supplied, then it will only return information about those packages (and is thus faster to execute).
collapse	Logical. If TRUE then the dependency fields will be collapsed; if FALSE (default) then the which fields will be kept separate.

---

availablePackagesOverride *Create a custom "available.packages" object*

Description

This is the mechanism by which install.packages determines which packages should be installed from where. With this override, we can indicate arbitrary repos, Package, File for each individual package.

Usage

```
availablePackagesOverride(
  toInstall,
  repos,
  purge,
  type = getOption("pkgType"),
  verbose = getOption("Require.verbose")
)
```

Arguments

toInstall	A pkgDT object
repos	The remote repository (e.g., a CRAN mirror), passed to either install.packages, install_github or installVersions. **When** options(Require.usePak = TRUE)**:** repos is added to pak's repository list via options(repos). However, pak always includes CRAN and Bioconductor as built-in defaults regardless of this setting – repos can only *add* sources, it cannot prevent pak from also searching CRAN. This differs from the default (usePak = FALSE) behaviour where

repos strictly controls which repositories are used. Use pak::cache_clean() to clear pak's download cache if needed.

purge — Logical. Should all caches be purged? Default is getOption("Require.purge", FALSE). 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 R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE, or if unset, defaults to 3600 (one hour – see utils::available.packages).

Internally, there are calls to available.packages.

type — See utils::install.packages

verbose — Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

---

availableVersionOK *Needs* VersionOnRepos, versionSpec *and* inequality *columns*

Description

Needs VersionOnRepos, versionSpec and inequality columns

Usage

```
availableVersionOK(pkgDT)
```

Arguments

pkgDT — A pkgDT object

---

cacheClearPackages *Clear cached package tarballs*

Description

Removes downloaded package archives from the cache that cachePkgDir() returns. With usePak = TRUE (the default) this delegates to pak::cache_clean() (no packages arg) or pak::cache_delete(package = ...) (selective). With usePak = FALSE it walks Require's legacy bookkeeping dir and unlinks tarballs there.

Usage

```
cacheClearPackages(
  packages,
  ask = interactive(),
  Rversion = versionMajorMinor(),
  clearCranCache = FALSE,
  verbose = getOption("Require.verbose")
)

clearRequirePackageCache(
  packages,
  ask = interactive(),
  Rversion = versionMajorMinor(),
  clearCranCache = FALSE,
  verbose = getOption("Require.verbose")
)
```

Arguments

packages	Either missing or a character vector of package names (currently cannot specify version number) to remove from the cache.
ask	Logical. If TRUE, asks the user to confirm before deleting.
Rversion	An R version (major.minor, e.g., "4.2"). Defaults to the current R version. **Ignored under** usePak = TRUE – see "Migration note".
clearCranCache	Logical. If TRUE, also clears the local crancache cache, which is only relevant if options(Require.useCranCache = TRUE).
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

Details

Require's own bookkeeping (SHA DB, mirrors.csv, available.packages snapshots) is preserved by this function; use cachePurge() to clear those as well.

clearRequirePackageCache() is a deprecated alias.

Value

Called for their side effect of deleting cached package files; return NULL invisibly.

Migration note

The Rversion parameter is honoured only on the legacy path – pak's cache is not partitioned by R version the way Require's cache was, so pak::cache_clean()/pak::cache_delete() operate on the whole cache regardless of Rversion. The function emits a verbose note when Rversion != versionMajorMinor() under usePak = TRUE.

cacheDefaultDir *The default cache directory for Require Cache*

Description

A wrapper around `tools::R_user_dir("Require", which = "cache")` that creates the directory, if it does not exist.

Usage

```
cacheDefaultDir()
```

Value

The default cache directory

cacheDir *Path to (package) cache directory*

Description

`cacheDir()` returns Require's own scratch directory (SHA database, available.packages snapshots, mirrors.csv, pkgDep cache); `cachePkgDir()` returns the package binary tarball cache.

Usage

```
cacheDir(create, verbose = getOption("Require.verbose"))

cachePkgDir(create)
```

Arguments

`create`	A logical indicating whether the path should be created if it does not exist. Default is `FALSE`.
`verbose`	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in `Require` function, when `verbose >= 2`, also returns details as if `returnDetails = TRUE` (for backwards compatibility).

Value

A path string. When `create = TRUE`, the directory is created (with a `README` placed in `cacheDir()`'s root if absent); otherwise the function just returns what the path would be.

What goes where

Function	What it holds	Default location
`cacheDir()`	Require-internal bookkeeping (SHA DB, mirrors.csv, pkgDep cache)	`tools::R_user_dir("Require",`
`cachePkgDir()`	Package binary tarballs	pak's `cache_summary()$cachepa`
`cachePkgDir()`	Package binary tarballs	`<cacheDir>/packages/<Rver>` (l

Both defaults flow from `tools::R_user_dir()`, so setting `R_USER_CACHE_DIR=/some/path` in `.Renviron` redirects **both** caches to sibling subdirectories of `/some/path/R/` – pak's cache lands in `pkgcache/pkg/`, Require's in `Require/`. That's the one-knob way to set up a shared cache across machines or R versions.

How `cachePkgDir()` changes with `usePak`

`getOption("Require.usePak", TRUE)` **(default)** Thin wrapper over `pak::cache_summary()$cachepath`. The directory is owned by pak/pkgcache; location is controlled by `R_USER_CACHE_DIR` (read at pak's subprocess spawn time). Default: `tools::R_user_dir("pkgcache", "cache")/pkg`.

`usePak = FALSE` **(legacy)** Returns `<cacheDir>/packages/<Rver>`, controlled by `R_REQUIRE_CACHE`.

Require-internal bookkeeping files always live next to the legacy path (`<cacheDir>/packages/<Rver>`) regardless of `usePak` – pak doesn't know about them and would treat them as stray files.

Deprecations

The following Require-specific knobs and helpers were folded into the pair above. Each is still functional for one release cycle and emits a deprecation warning when used.

Deprecated	Use instead
`cacheGetOptionCachePkgDir()`	`cachePkgDir()`
`rpackageFolder()` (internal)	(inlined into `checkLibPaths()`)
`purgeCache()`	`cachePurge()`
`clearRequirePackageCache()`	`cacheClearPackages()`
`options("Require.cachePkgDir")`	`R_USER_CACHE_DIR` env var
`Sys.getenv("R_REQUIRE_PKG_CACHE")`	`R_USER_CACHE_DIR` env var

`cacheGetOptionCachePkgDir` *Get the option for* `Require.cachePkgDir` *(deprecated)*

Description

Deprecated. Use `cachePkgDir()` instead – it is now the single getter for the package-tarball cache and wraps `pak::cache_summary()$cachepath` under `usePak = TRUE` (the default).

This function is kept for one release cycle as a functional shim. It still resolves a user-supplied path from `options("Require.cachePkgDir")` or the `R_REQUIRE_PKG_CACHE` environment variable when set, but those two knobs are themselves deprecated and ignored under `usePak = TRUE`. To redirect pak's package cache to a shared location, set `R_USER_CACHE_DIR` in `.Renviron` (pak's standard env var). See `cacheDir()` for the full migration table.

Resolution order (legacy path):

1. If `R_REQUIRE_PKG_CACHE` is set, return it.
2. Else if `options("Require.cachePkgDir")` is character, return it.
3. Else if the option is `TRUE`, return `cachePkgDir(FALSE)`.
4. Else if the option is `FALSE`, return `NULL`.
5. Otherwise, return `cachePkgDir(FALSE)`.

Usage

```
cacheGetOptionCachePkgDir()
```

Value

The package cache directory, or `NULL` when caching is disabled. Deprecated in favour of `cachePkgDir()`.

`cachePurge` *Purge everything in the Require cache*

Description

Clears Require's own bookkeeping caches: the `available.packages()` snapshot, the GitHub SHA database, the `DESCRIPTION` cache, and the pkgDep memoisation database. These live under `cacheDir()` and are distinct from the package binary tarball cache.

Usage

```
cachePurge(packages = FALSE, repos = getOption("repos"))

purgeCache(packages = FALSE, repos = getOption("repos"))
```

Arguments

packages	Either a character vector of packages to install via `install.packages`, then load (i.e., with `library`), or, for convenience, a vector or list (using `c` or `list`) of unquoted package names to install and/or load (as in `require`, but vectorized). Passing vectors of names may not work in all cases, so user should confirm before relying on this behaviour in operational 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. As a convenience for copy-pasting long lists, any character element that contains newlines is split into one package per line, with whitespace trimmed and blank or #-prefixed lines dropped. For example, `Require("\n dplyr\n # ggplot2\n PredictiveEcology/LandR@development\n")` is equivalent to `Require(c("dplyr", "PredictiveEcology/LandR@development"))`. Alternatively, an unquoted `{...}` block is accepted, with each line treated as one package spec, e.g. `Require({ dplyr; lme4; PredictiveEcology/LandR@development })`. Note that R's parser strips comments inside `{...}` before this function runs, so to omit a package delete the line (or comment it out – both have the same effect). Version constraints such as `pkg (>= 1.0)` do not parse inside `{...}` and must use the quoted string form.
repos	The remote repository (e.g., a CRAN mirror), passed to either `install.packages`, `install_github` or `installVersions`. **When** `options(Require.usePak = TRUE)`**:** `repos` is added to pak's repository list via `options(repos)`. However, pak always includes CRAN and Bioconductor as built-in defaults regardless of this setting – `repos` can only *add* sources, it cannot prevent pak from also searching CRAN. This differs from the default (`usePak = FALSE`) behaviour where `repos` strictly controls which repositories are used. Use `pak::cache_clean()` to clear pak's download cache if needed.

Details

With `packages = TRUE`, the package binary cache is also cleared by calling `cacheClearPackages()` – which under `usePak = TRUE` (the default) delegates to `pak::cache_clean()`, and under the legacy path walks Require's bookkeeping dir directly.

`purgeCache()` is a deprecated alias.

Value

Run for its side effect, namely, all cached objects are removed.

---

checkLibPaths *Creates the directories, and adds version number*

Description

Creates the directories, and adds version number

Usage

```
checkLibPaths(libPaths, ifMissing, exact = FALSE, ...)
```

Arguments

libPaths	The library path (or libraries) where all packages should be installed, and looked for to load (i.e., call library). This can be used to create isolated, stand alone package installations, if used with standAlone = TRUE. Currently, the path supplied here will be prepended to .libPaths() (temporarily during this call) to Require if standAlone = FALSE or will set (temporarily) .libPaths() to c(libPaths, tail(libPaths(), 1) to keep base packages.
ifMissing	An alternative path if libPaths argument is missing.
exact	Logical. If FALSE, the default, then checkLibPaths will append the R version number on the libPaths supplied. If TRUE, checkLibPaths will return exactly the libPaths supplied.
...	Not used, but allows other functions to pass through arguments.

---

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
```

---

compareVersion2 *Compare package versions*

Description

Alternative to `utils::compareVersion` that is vectorized on `version`, `versionSpec` and/or `inequality`. This will also return an NA element in the returned vector if one of the arguments has NA for that element.

Usage

```
compareVersion2(version, versionSpec, inequality)
```

Arguments

version	One or more package versions. Can be character or numeric_version.
versionSpec	One or more versions to compare to. Can be character or numeric_version.
inequality	The inequality to use, i.e., >=.

Value

a logical vector of the length of the longest of the 3 arguments.

See Also

Other version specifications: extractPkgName(), parseGitHub(), trimRedundancies(), trimVersionNumber()

---

dealWithMissingLibPaths

Only checks for deprecated libPath argument (singular)

Description

Only checks for deprecated libPath argument (singular)

Usage

```
dealWithMissingLibPaths(
  libPaths,
  standAlone = getOption("Require.standAlone", FALSE),
  ...
)
```

Arguments

libPaths	The library path (or libraries) where all packages should be installed, and looked for to load (i.e., call library). This can be used to create isolated, stand alone package installations, if used with standAlone = TRUE. Currently, the path supplied here will be prepended to .libPaths() (temporarily during this call) to Require if standAlone = FALSE or will set (temporarily) .libPaths() to c(libPaths, tail(libPaths(), 1) to keep base packages.
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.
...	Checks for the incorrect argument libPath (no s)

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")

dlGitHubDESCRIPTION(
  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 getOption("Require.purge", FALSE). 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 R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE, or if unset, defaults to 3600 (one hour – see utils::available.packages). Internally, there are calls to available.packages.
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 -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

Details

dlGitHubDESCRIPTION retrieves the DESCRIPTION file from GitHub.com

Value

DESCRIPTIONFileVersionV() and DESCRIPTIONFileOtherV() return a character vector with one element per file (the Version field, or the field named in other); dlGitHubDESCRIPTION() returns the data.table of parsed GitHub refs with the local path of each downloaded DESCRIPTION file in a DESCFile column.

---

detachAll *Detach and unload all packages*

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 doSort 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 -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

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 not loaded, so was not unloaded.

dlArchiveVersionsAvailable

Available and archived versions

Description

These are wrappers around available.packages and also get the archived versions available on CRAN.

Usage

```
dlArchiveVersionsAvailable(
  package,
  repos = getOption("repos"),
  verbose = getOption("Require.verbose")
)

available.packagesCached(
  repos,
  purge,
  verbose = getOption("Require.verbose"),
  returnDataTable = TRUE,
  type
)
```

Arguments

package	A single package name (without version or github specifications)
repos	The remote repository (e.g., a CRAN mirror), passed to either `install.packages`, `install_github` or `installVersions`. **When** `options(Require.usePak = TRUE)`**:** `repos` is added to pak's repository list via `options(repos)`. However, pak always includes CRAN and Bioconductor as built-in defaults regardless of this setting – `repos` can only *add* sources, it cannot prevent pak from also searching CRAN. This differs from the default (`usePak = FALSE`) behaviour where `repos` strictly controls which repositories are used. Use `pak::cache_clean()` to clear pak's download cache if needed.
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in `Require` function, when `verbose >= 2`, also returns details as if `returnDetails = TRUE` (for backwards compatibility).
purge	Logical. Should all caches be purged? Default is `getOption("Require.purge", FALSE)`. 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 R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE, or if unset, defaults to 3600 (one hour – see utils::available.packages).

Internally, there are calls to available.packages.

returnDataTable

Logical. If TRUE, the default, then the return is a data.table. Otherwise, it is a matrix, as per available.packages

type See utils::install.packages

Details

dlArchiveVersionsAvailable 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

dlArchiveVersionsAvailable(): a list with one data.table per package, the versions found in the CRAN archive (repo, PackageUrl and file metadata), empty when none. available.packagesCached(): the available-packages table for repos, as a data.table (default) or a matrix as utils::available.packages() returns it.

---

doLibPaths *Deals with missing libPaths arg, and takes first*

Description

Deals with missing libPaths arg, and takes first

Usage

```
doLibPaths(libPaths, standAlone = FALSE)
```

Arguments

libPaths	The library path (or libraries) where all packages should be installed, and looked for to load (i.e., call library). This can be used to create isolated, stand alone package installations, if used with standAlone = TRUE. Currently, the path supplied here will be prepended to .libPaths() (temporarily during this call) to Require if standAlone = FALSE or will set (temporarily) .libPaths() to c(libPaths, tail(libPaths(), 1) to keep base packages.
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.

envPkgCreate *1st level –> create the .pkgEnv object in Require*

Description

1st level –> create the .pkgEnv object in Require

Usage

```
envPkgCreate(parentEnv = asNamespace("Require"))
```

Arguments

parentEnv	The parent environment in which to make the new environment. Defaults to asNamespace("Require")

envPkgDepDepsCreate *3rd level for deps ##*

Description

3rd level for deps ##

Usage

```
envPkgDepDepsCreate()
```

envPkgDepDESCFileCreate *3rd level for DESCRIPTIONFile*

Description

3rd level for DESCRIPTIONFile

Usage

```
envPkgDepDESCFileCreate()
```

`extractPkgName` *Extract info from package character strings*

Description

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

Usage

```
extractPkgName(pkgs, filenames)

extractVersionNumber(pkgs, filenames)

extractInequality(pkgs)

extractPkgGitHub(pkgs)
```

Arguments

`pkgs`	A character string vector of packages with or without GitHub path or versions
`filenames`	Can be supplied instead of `pkgs` if it is a filename e.g., a .tar.gz or .zip that was downloaded from CRAN.

Value

Just the package names without extraneous info.

See Also

`trimVersionNumber()`

Other version specifications: `compareVersion2()`, `parseGitHub()`, `trimRedundancies()`, `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")
```

getDeps	*The* packages *argument may have up to 4 pieces of information for GitHub packages: name, repository, branch, version. For CRAN-alikes, it will only be 2 pieces: name, version. There can also be an inequality or equality, if there is a version.*

Description

The packages argument may have up to 4 pieces of information for GitHub packages: name, repository, branch, version. For CRAN-alikes, it will only be 2 pieces: name, version. There can also be an inequality or equality, if there is a version.

Usage

```
getDeps(
  pkgDT,
  which,
  recursive,
  type = type,
  repos,
  libPaths,
  verbose,
  parentChain = ""
)
```

Arguments

pkgDT	A pkgDT object e.g., from toPkgDT
which	a character vector listing the types of dependencies, a subset of c("Depends", "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.
type	See utils::install.packages
repos	is used for ap.
libPaths	A path to search for installed packages. Defaults to .libPaths()
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).
parentChain	A character string representing the chain of parent packages that required this package, e.g., "digest -> reproducible". Used to provide context in "not on CRAN" messages. Default "".

Details

If version is not supplied, it will take the local, installed version, if it exists. Otherwise, it is assumed that the HEAD is desired. The function will find it in the ap or on `github.com`. For github packages, this is obviously a slow step, which can be accelerated if user supplies a sha or a version e.g., getDeps("PredictiveEcology/LandR@development (==1.0.2)")

Value

A (named) vector of SaveNames, which is a concatenation of the 2 or 4 elements above, plus the `which` and the `recursive`.

`internetExists` *Internet Exists query*

Description

Simple test for internet availability.

Usage

```
internetExists(verbose = getOption("Require.verbose"), force = FALSE)
```

Arguments

`verbose`	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in `Require` function, when `verbose >= 2`, also returns details as if `returnDetails = TRUE` (for backwards compatibility).
`force`	If `TRUE`, probe even when `options("Require.checkInternet")` is `FALSE`. Used at points where we're about to do real work (e.g., install dispatch) and a 2-second probe is cheap relative to the cost of failing late. Defaults to `FALSE` to preserve the no-probe default behaviour for all other call sites.

Value

Logical. `TRUE` if internet is available, `FALSE` if not.

invertList *Invert a 2-level list*

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
```

joinToAvailablePackages *Join a data.table with a* `Package` *column to* `available.packages`

Description

Will join `available.packages()` with `pkgDT`, if `pkgDT` does not already have a column named `Depends`, which would be an indicator that this had already happened.

Usage

```
joinToAvailablePackages(pkgDT, repos, type, which, verbose)
```

Arguments

pkgDT	A `pkgDT` object e.g., from `toPkgDT`
repos	is used for ap.
type	See `utils::install.packages`

which	a character vector listing the types of dependencies, a subset of c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances"). Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances".
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

Value

The returned data.table will have most of the columns from available.packages appended to the pkgDT, including Depends, Imports, Suggests. It will change the column name that is normally returned from available.packages as Version to VersionOnRepos.

---

linkOrCopy *Create link to file, falling back to making a copy if linking fails.*

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)

fileRenameOrMove(from, to)
```

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

masterMainToHead *This converts master or main to HEAD for a git repo*

Description

This will also convert a git repo with nothing after the @ to @HEAD

Usage

```
masterMainToHead(gitRepo)
```

Arguments

gitRepo	A git repository of the form account/repo with optional @branch or @sha or @tag

Value

The git repository with @HEAD if it had @master, @main or no @.

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

Description

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

This will only show a message if the value of verbose is greater than the verboseLevel. This is mostly useful for developers of code who want to give users of their code easy access to how verbose their code will be. A developer of a function will place this messageVerbose internally, setting the verboseLevel according to how advanced they may want the message to be. 1 is a reasonable default for standard use, 0 would be for "a very important message for all users", 2 or above would be increasing levels of details for e.g., advanced use. If a user sets to -1 with this numeric approach, they can avoid all messaging.

Usage

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

messageVerbose(..., verbose = getOption("Require.verbose"), verboseLevel = 1)

messageVerboseCounter(
  pre = "",
  post = "",
  verbose = getOption("Require.verbose"),
  verboseLevel = 1,
```

```
  counter = 1,
  total = 1,
  minCounter = 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 -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).
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.
pre	A single text string to paste before the counter
post	A single text string to paste after the counter
counter	An integer indicating which iteration is being done
total	An integer indicating the total number to be done.
minCounter	An integer indicating the minimum (i.e,. starting value)

Value

Used for side effects, namely messaging that can be turned on or off with different numeric values of verboseLevel. A user sets the verboseLevel for a particular message.

---

modifyList2 modifyList *for multiple lists*

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. Note: default keep.null = TRUE, which is different than modifyList

Usage

```
modifyList2(..., keep.null = FALSE)

modifyList3(..., keep.null = TRUE)
```

Arguments

...	One or more named lists.
keep.null	If TRUE, NULL elements in val become NULL elements in x. Otherwise, the corresponding element, if present, is deleted from x.

Details

More or less a convenience around Reduce(modifyList, list(...)), with some checks, and the addition of keep.null = TRUE by default.

Value

A list: the first argument with each subsequent argument's elements merged in, later ones overriding earlier ones of the same name.

Note

modifyList3 retains the original behaviour of modifyList2 (prior to Oct 2022); however, it cannot retain NULL values in lists.

Examples

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

---

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()

## 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

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

pakEnv *2nd level*

Description

2nd level

Usage

```
pakEnv()
```

parseGitHub *Parse a github package specification*

Description

This converts a specification like `PredictiveEcology/Require@development` into separate columns, "Account", "Repo", "Branch", "GitSubFolder" (if there is one)

Usage

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

Arguments

`pkgDT`	A pkgDT data.table.
`verbose`	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in `Require` function, when `verbose >= 2`, also returns details as if `returnDetails = TRUE` (for backwards compatibility).

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.

See Also

Other version specifications: `compareVersion2()`, `extractPkgName()`, `trimRedundancies()`, `trimVersionNumber()`

pkgDepEnv *2nd level*

Description

2nd level

Usage

```
pkgDepEnv()
```

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 -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

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

```
opts <- Require:::.setupExample()

pkgDepIfDepRemoved("reproducible", "data.table")

Require:::.cleanup(opts)
```

---

pkgDepTopoSort *Reverse package depends*

Description

This is a wrapper around `tools::dependsOnPkgs`, but with the added option of `topoSort`, which will sort them such that the packages at the top will have the least number of dependencies that are in `packages`. 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.

`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 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.

Usage

```
pkgDepTopoSort(
  packages,
  deps,
  reverse = FALSE,
  topoSort = TRUE,
  libPaths,
  useAllInSearch = FALSE,
  returnFull = TRUE,
  recursive = TRUE,
  purge = getOption("Require.purge", FALSE),
  which = c("Depends", "Imports", "LinkingTo"),
  type = getOption("pkgType"),
  verbose = getOption("Require.verbose"),
  ...
)
```

```
pkgDep2(...)

pkgDep(
  packages,
  libPaths,
  which = c("Depends", "Imports", "LinkingTo"),
  recursive = TRUE,
  depends,
  imports,
  suggests,
  linkingTo,
  repos = getOption("repos"),
  keepVersionNumber = TRUE,
  includeBase = FALSE,
  includeSelf = TRUE,
  sort = TRUE,
  simplify = TRUE,
  purge = getOption("Require.purge", FALSE),
  verbose = getOption("Require.verbose"),
  type = getOption("pkgType"),
  Additional_repositories = FALSE,
  ...
)
```

Arguments

packages	Either a character vector of packages to install via `install.packages`, then load (i.e., with `library`), or, for convenience, a vector or list (using `c` or `list`) of unquoted package names to install and/or load (as in `require`, but vectorized). Passing vectors of names may not work in all cases, so user should confirm before relying on this behaviour in operational 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. As a convenience for copy-pasting long lists, any character element that contains newlines is split into one package per line, with whitespace trimmed and blank or #-prefixed lines dropped. For example, `Require("\n dplyr\n # ggplot2\n PredictiveEcology/LandR@development\n")` is equivalent to `Require(c("dplyr", "PredictiveEcology/LandR@development"))`. Alternatively, an unquoted `{...}` block is accepted, with each line treated as one package spec, e.g. `Require({ dplyr; lme4; PredictiveEcology/LandR@development })`. Note that R's parser strips comments inside `{...}` before this function runs, so to omit a package delete the line (or comment it out – both have the same effect). Version constraints such as `pkg (>= 1.0)` do not parse inside `{...}` and must use the quoted string form.
deps	An optional named list of (reverse) dependencies. If not supplied, then `tools::dependsOnPkgs(..., recursive = TRUE)` will be used

reverse	Logical. If TRUE, then this will use tools::pkgDependsOn to determine which packages depend on the packages
topoSort	Logical. If TRUE, the default, then the returned list of packages will be in order with the least number of dependencies listed in packages at the top of the list.
libPaths	A path to search for installed packages. Defaults to .libPaths()
useAllInSearch	Logical. If TRUE, then all non-core R packages in search() will be appended to packages to allow those to also be identified
returnFull	Logical. Primarily useful when reverse = TRUE. If TRUE, then then all installed packages will be searched. If FALSE, the default, only packages that are currently in the search() path and passed in packages will be included in the possible reverse dependencies.
recursive	Logical. Should dependencies of dependencies be searched, recursively. NOTE: Dependencies of suggests will not be recursive. Default TRUE.
purge	Logical. Should all caches be purged? Default is getOption("Require.purge", FALSE). 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 R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE, or if unset, defaults to 3600 (one hour – see utils::available.packages). Internally, there are calls to available.packages.
which	a character vector listing the types of dependencies, a subset of c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances"). Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances".
type	See utils::install.packages
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).
...	Currently only dependencies as an alternative to which. If specified, then which will be ignored.
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. **When** options(Require.usePak = TRUE)**:** repos is added to pak's repository list via options(repos). However, pak always includes CRAN and Bioconductor as built-in defaults regardless of this setting – repos can only *add* sources, it cannot prevent pak from also searching CRAN. This differs from the default (usePak = FALSE) behaviour where repos strictly controls which repositories are used. Use pak::cache_clean() to clear pak's download cache if needed.

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 `tail(.libPaths(), 1)`

includeSelf
: Logical. If TRUE, the default, then the dependencies will include the package itself in the returned list elements, otherwise, only the "dependencies"

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.

simplify
: Logical or numeric. If TRUE (or > 0), the default, the return object is "just" a character vector of package names (with version requirements). If FALSE (or 0), then a data.table will be returned with 4 columns, Package, packageFullName, parentPackage (the package name for which the given line entry is a dependency; will be "user" if it was user supplied) and deps, which is a list of data.tables of all dependencies. If a negative number, then it will return a similar data.table as with FALSE, however, duplications in the recursive package dependencies are left intact.

Additional_repositories
: Logical. If TRUE, then pkgDep will return a list of data.table objects (instead of character vectors) with a column packageFullName and possibly a second column Additional_repositories, which may have been specified in a DESCRIPTION file. NOTE: THIS ALTERS THE OUTPUT CLASS

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

```
opts <- Require:::.setupExample()

pkgDepTopoSort(c("Require", "data.table"), reverse = TRUE)

Require:::.cleanup(opts)


opts <- Require:::.setupExample()

pkgDep2("reproducible")
```

```
# much bigger one
pkgDep2("tidyverse")

Require:::.cleanup(opts)


opts <- Require:::.setupExample()

pkgDep("tidyverse", recursive = TRUE)

# GitHub, local, and CRAN packages
pkgDep(c("PredictiveEcology/reproducible", "Require", "plyr"))

Require:::.cleanup(opts)
```

---

pkgSnapshot *Take a snapshot of all the packages and version numbers*

Description

This can be used later by `Require` to install or re-install the correct versions. See examples.

Usage

```
pkgSnapshot(
  packageVersionFile = getOption("Require.packageVersionFile"),
  libPaths = .libPaths(),
  standAlone = FALSE,
  purge = getOption("Require.purge", FALSE),
  exact = TRUE,
  includeBase = FALSE,
  verbose = getOption("Require.verbose")
)

pkgSnapshot2(
  packageVersionFile = getOption("Require.packageVersionFile"),
  libPaths,
  standAlone = FALSE,
  purge = getOption("Require.purge", FALSE),
  exact = TRUE,
  includeBase = FALSE,
  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 .libPaths()[1].
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.
purge	Logical. Should all caches be purged? Default is getOption("Require.purge", FALSE). 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 R_AVAILABLE_PACKAGES_CACHE_CONTROL_MAX_AGE, or if unset, defaults to 3600 (one hour – see utils::available.packages). Internally, there are calls to available.packages.
exact	Logical. If TRUE, the default, then for GitHub packages, it will install the exact SHA, rather than the head of the account/repo@branch. 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 >=, allowing a subsequent user to install with a minimum version number, as opposed to an exact version number.
includeBase	Logical. Should R base packages be included, specifically, those in tail(.libPaths(), 1)
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

Details

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

pkgSnapshot2 returns a vector of package names and versions, with no file output. See examples.

Value

Will both write a file, and (invisibly) return a vector of packages with the version numbers. This vector can be used directly in `Require`, though it should likely be used with `require = FALSE` to prevent attaching all the packages.

Installing from a snapshot file

Pass the snapshot file to `Require()` via `packageVersionFile = "snapshot.txt"`. By default this routes through a multi-stage installer (gated by `options(Require.snapshotInstaller = "install.packages")`) that:

1. **Skips already-installed-at-target-version refs.**
2. **Cache pre-filter** via `pkgcache::pkg_cache_list()`. Source tarballs feed `pak::pkg_install(local::...)`; binaries are reserved for step (4). Rotten cache rows (missing fullpath, gzip-corrupt, DESCRIPTION mismatch) are auto-evicted via `pkgcache::pkg_cache_delete_files()` so future runs don't keep tripping on them.
3. **Parallel libcurl-multi download** for refs not in cache, chunked at 50 URLs per call (macOS file-descriptor limit). Walks priority URLs: row's `Repository` -> PPM -> CRAN -> CRAN/Archive. Up to 4 retries with exponential backoff (`Require.snapshotDownloadAttempts`).
4. **Hybrid binary-first install** via `install.packages(type = "binary")` for any ref that has a cache binary matching this R session (`R.version$platform`, `<major>.<minor>`). Skips compilation, reduces pak's parallel-build workload. Disable via `options(Require.snapshotInstallerHybrid = FALSE)`.
5. `pak::pkg_install(local::...)` **for the rest** with `dependencies = NA`, `upgrade = FALSE`. Refs already-installed-at-target-version are excluded upfront so pak doesn't reinstall-to-self.
6. `install.packages(repos = file://...)` **fallback** if pak refuses (its solver is strict; `install.packages` is best-effort and tolerates per-package compile failures).
7. **Bump-and-retry**: for any ref still missing, walks newer-than-pin versions from CRAN/PPM/Archive ascending and tries each until one installs (capped at 20 candidates). Disable via `options(Require.snapshotInstalle = FALSE)` for strict reproducibility (no drift, fail loudly).
8. **Diagnostic report** classifying each gap with a concrete `fix:` line.

Built binaries from this run are added back to pkgcache via `cacheBuiltBinaries()` (registered on `on.exit`), so a subsequent run hits step (4) instead of recompiling.

Common snapshot pitfalls

Version-coherence (the snapshot's own pins disagree) A ref's `DESCRIPTION` declares `Imports: X (>= V)` but the snapshot pins `X` at a version that doesn't satisfy it. pak's strict solver refuses to install. The installer runs a coherence pre-check before handing off to pak and prints any unsatisfied constraint with a fix suggestion (e.g., `servr 0.30 requires xfun (>= 0.42); snapshot pins xfun = 0.40 -> bump`

`R` **pseudo-package** `pkgSnapshot()` writes a row recording the running R version (e.g., `R,4.4,...`). The installer skips this row alongside base packages.

System library mismatches (compile failures) Source builds need the host's system libs to match what the package expects. The installer's `classifyCompileFailure()` recognises missing-header errors (jpeglib.h, gdal.h, geos_c.h, glpk.h, ft2build.h, sodium.h, ...) and prints the corresponding `brew install ...` (or apt) suggestion. R 4.5's removal of `Calloc/Free`, GDAL

>= 3.10's `const OGRSpatialReference*` ABI change, and Rcpp's `class_::constructor<>` template-arity limit are each pattern-matched and reported with a "bump <pkg>" suggestion.

Mac toolchain – `~/.R/Makevars` R's default compile flags only search `/opt/R/arm64/include`. To pick up Homebrew headers (libjpeg, glpk, freetype, etc.), add to `~/.R/Makevars`:

```
CPPFLAGS += -I/opt/homebrew/include
LDFLAGS  += -L/opt/homebrew/lib
```

Stale `pkgcache` **index** `pkgcache` shares state across R versions and architectures. Cache rows tagged with platform/rversion that don't match this session are filtered out (e.g., R-4.5 binaries when running R-4.4); validation also catches index rows whose file content disagrees with the index (a known historical bug-class). Both kinds get auto-evicted, so `pak::cache_clean()` is rarely needed.

Pak's `local::` **is source-only** Confirmed empirically that pak's `pkg_install("local::<file>")` rejects binary tarballs (`.tgz` / `.zip` content) with "Platform mismatch" – even when the binary is for the current platform. The hybrid stage installs binaries via `install.packages(type = "binary")` BEFORE pak runs, so pak only sees source refs.

Pak strict-aborts on first build failure `install.packages` continues past per-package compile failures; `pak::pkg_install` does not. The fallback to `install.packages` and the bump-and-retry stage together provide a best-effort completion guarantee even when individual refs are environment-fragile.

Snapshot-installer options

`Require.snapshotInstaller` `"install.packages"` to use the pipeline above; `"pak"` for the legacy direct-pak path.

`Require.snapshotInstallerUsePPM` TRUE (default) to prepend a PPM binary repo. PPM serves Mac binaries by content-negotiating the `R/<version>` User-Agent.

`Require.snapshotInstallerHybrid` TRUE (default) – pre-install cache binaries via `install.packages(type = "binary")` before pak.

`Require.snapshotInstallerBumpOnFail` TRUE (default) – walk newer versions for refs that fail at the pin. FALSE for strict reproducibility.

`Require.snapshotInstallerKnownFails` character vector of pkg names to skip in bump-retry (e.g. environment-dependent refs whose newer versions also won't help).

`Require.snapshotInstallerPakSilent` FALSE (default) – pak's resolver output reaches the user.

`Require.snapshotDownloadAttempts` Retry count for libcurl-multi downloads. Default 4.

`Require.snapshotDownloadChunk` URLs per `download.file()` call. Default 50 (stays under macOS's ~256 file-descriptor ulimit).

Examples

```
opts <- Require:::.setupExample()

# install one archived version so that below does something interesting
libForThisEx <- tempdir2("Example")
```

```
Require("crayon (==1.5.1)", libPaths = libForThisEx, require = FALSE)
# Normal use -- using the libForThisEx for example;
#    normally libPaths would be omitted to get all
#    packages in user or project library
tf <- tempfile()

# writes to getOption("Require.packageVersionFile")
# within project; also returns a vector
# of packages with version
pkgs <- pkgSnapshot(
  packageVersionFile = tf,
  libPaths = libForThisEx, standAlone = TRUE # only this library
)

# Now move this file to another computer e.g. by committing in git,
#   emailing, googledrive
#   on next computer/project
Require(packageVersionFile = tf, libPaths = libForThisEx)

# Using pkgSnapshot2 to get the vector of packages and versions
pkgs <- pkgSnapshot2(
  libPaths = libForThisEx, standAlone = TRUE
)
Install(pkgs) # will install packages from previous line

Require:::.cleanup(opts)
unlink(getOption("Require.packageVersionFile"))
```

`RequireOptions` `Require` *options*

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:

`install` Default: `TRUE`. This is the default argument to `Require`, but does not affect `Install`. If this is `FALSE`, then no installations will be attempted, and missing packages will result in an error.

`cachePkgDir` Deprecated, and ignored under `Require.usePak = TRUE` (the default): redirect pak's package cache with the `R_USER_CACHE_DIR` environment variable instead. `R_REQUIRE_PKG_CACHE` is deprecated for the same reason. Both are still honoured when `Require.usePak = FALSE`. Default: `cacheGetOptionCachePkgDir()`, which must be either a path or a logical. To turn off package caching, set this to `FALSE`. This can be set using an environment variable e.g., `Sys.setenv(R_REQUIRE_PKG_CACHE = "somePath")`, or `Sys.setenv(R_REQUIRE_PKG_CACHE = "TRUE")`; if that is not set, then an either a path or logical option (`options(Require.cachePkgDir = "somePath")` or `options(Require.cachePkgDir = TRUE)`). If `TRUE`, the default folder location `cachePkgDir()` 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.

`otherPkgs` Default: A character vector of packages that are generally more successful if installed from Source on Unix-alikes. Since there are repositories that offer binary packages builds for Linux (e.g., RStudio Package Manager), the vector of package names indicated here will default to a standard CRAN repository, forcing a source install. See also `spatialPkgs` option, which does the same for spatial packages.

`noRemotes` Default: `FALSE`. If `TRUE`, GitHub-style specs (`account/repo@branch`) passed to `Require`/`Install` are rewritten to their bare package name (the version constraint, if any, is preserved). The packages are then resolved from `repos` (e.g., a CRAN-like or r-universe repository serving prebuilt binaries) instead of being cloned and built from GitHub source. This avoids the need for git authentication and a source-build toolchain (e.g., Rtools on Windows), which is useful for workshops and other binary-only setups. Because version constraints are kept, `repos` must provide a version that satisfies them, otherwise resolution will fail (rather than silently falling back to GitHub). Ensure the relevant repository (e.g., `predictiveecology.r-universe.dev`) is in `getOption("repos")`.

`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).

`cloneFrom` Default: `NULL`. A path to an existing package library. When set, any package that is already installed there at the version being requested is copied or hard-linked into the destination library instead of being downloaded and installed, which is much faster when populating a new project library on a machine that already has the packages. Only packages that need no compilation and were built under a compatible R version are cloned; the rest go through the normal install machinery, as do `pak`, `callr`, `processx` and `cli`, whose native helper executables do not survive a file-by-file copy. Has no effect under `Require.usePak = TRUE` (the

default): cloning lives in the legacy (non-pak) install path, so this option is consulted only when `Require.usePak = FALSE`.

`downloadTimeout` Default: `300L` (seconds). Used as the floor for `options("timeout")` during GitHub source-archive downloads in the legacy (non-pak) install path. R's stock 60-second timeout is too short for slow connections fetching multi-MB zips. Has no effect under `Require.usePak = TRUE`, which delegates downloads to pak's own libcurl client.

`spatialPkgs` Default: A character vector of packages that are generally more successful if installed from Source on Unix-alikes. Since there are repositories that offer binary packages builds for Linux (e.g., Posit Package Manager), the vector of package names indicated here will default to a standard CRAN repository, forcing a source install. See also `otherPkgs` option, which does the same for non-spatial packages.

`useCranCache` Default: `FALSE`. A user can optionally use the locally cached packages that are available due to a user's use of the `crancache` package.

`checkInternet` Default: `TRUE`. Whether `Require` may run a short internet probe before work that needs the network. Results are cached for `Require.internetExistsTimeout` seconds (30). Setting this `FALSE` skips the probe at most call sites, but not those that pass `force = TRUE`, where failing late costs more than a 2-second check.

`forcePakReinstall` Default: `FALSE`. Set `TRUE` to force one clean `install.packages("pak")` into the project library. This is the remedy for a pak whose bundled callr/processx helper executables no longer run – the state a file-by-file copy of a library leaves pak in. `find.package()` cannot detect that (every file is present, they just do not run), so the reinstall has to be asked for explicitly.

`installPackagesSys` Default: `2L`. Legacy (non-pak) path only. `0` installs in-process with `install.packages()`; a non-zero value runs builds and installs through the `sys` package in a subprocess; `2` additionally performs the CRAN downloads itself rather than leaving them to `install.packages()`. Requires the `sys` package.

`offlineMode` Default: `FALSE`. When `TRUE`, `Require` attempts no network access and installs only from local caches. It is also set automatically – together with `Require.offlineModeSetAutomatically` – when an install fails and an internet probe finds no connection.

`packageVersionFile` Default: `"packageVersions.txt"`. The default file path used by `pkgSnapshot()` when writing a snapshot.

`snapshotInstaller` Default: `"pak"`. Which installer to use for a snapshot install. `"install.packages"` selects the pre-pak chain, which is retained but no longer developed. See `?pkgSnapshot` for the rest of the snapshot-installer options.

`snapshotInstallerUsePPM` Default: `TRUE`. Prepend a Posit Package Manager binary repository when installing a snapshot through the `install.packages` chain. PPM serves macOS binaries by content-negotiating the `R/<version>` User-Agent.

`standAlone` Default: `TRUE`. The default `standAlone` argument for `Require`, `Install` and `setLibPaths()`. `TRUE` means the given library path is used on its own, without the user and site libraries appended; `FALSE` appends them. It controls whether those libraries are added, not how many library paths may be given – passing several is always allowed.

`updateRprofile` Default: `FALSE`. Whether `setLibPaths()` also writes the library path into the project `.Rprofile`, so it persists across sessions.

`usePak` Default: `TRUE`. Use pak to resolve and install packages. `FALSE` selects the pre-pak path. Several options apply only to that path and do nothing under the default: `cachePkgDir`, `cloneFrom`, `downloadTimeout` and `installPackagesSys`.

verbose Default: 1. See ?Require.

Value

A named list of the package options and their default values.

`rmBase` *Recursive function to remove* `.basePkgs`

Description

Recursive function to remove `.basePkgs`

Usage

```
rmBase(includeBase = formals(pkgDep)[["includeBase"]], deps)
```

Arguments

`includeBase`	Logical. If `FALSE`, the default, then base packages will be removed.
`deps`	Either a list of dependencies, a data.table of dependencies with a column `Package` or a vector of dependencies.

`rversions` *R versions*

Description

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

Usage

```
rversions
```

Details

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

```
# install.packages("rversions")
v = rversions::r_versions()
keep = which(as.Date(v$date, format = "
             as.Date("2018-01-01", format = "
dput(v[keep, c("version", "date")])
```

setdiffNamed *Like* setdiff*, but takes into account names*

Description

This will identify the elements in l1 that are not in l2. If missingFill is provided, then elements that are in l2, but not in l1 will be returned, assigning missingFill to their values. This might be NULL or "", i.e., some sort of empty value. This function will work on named lists, named vectors and likely on other named classes.

Usage

```
setdiffNamed(l1, l2, missingFill)
```

Arguments

l1	A named list or named vector
l2	A named list or named vector (must be same class as l1)
missingFill	A value, such as NULL or "" or "missing" that will be given to the elements returned, that are in l2, but not in l1

Details

There are 3 types of differences that might occur with named elements: 1. a new named element, 2. an removed named element, and 3. a modified named element. This function captures all of these. In the case of unnamed elements, e.g., setdiff, the first two are not seen as differences, if the values are not different.

Value

A vector or list of the elements in l1 that are not in l2, and optionally the elements of l2 that are not in l1, with values set to missingFill

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 .libPath()
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.
updateRprofile	Logical or Character string. If TRUE, then this function will put several lines of code in the current directory's .Rprofile file setting up the package libraries for this and future sessions. If a character string, then this should be the path to an .Rprofile file. To reset back to normal, run setLibPaths() without a libPath. Default: getOption("Require.updateRprofile", FALSE), 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 -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

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

```
opts <- Require:::.setupExample()
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()

# will have 2 or more paths
otherLib <- file.path(td, "newProjectLib")
setLibPaths(otherLib, standAlone = FALSE)
# Can restart R, and changes will stay

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

Require:::.cleanup(opts)
unlink(otherLib, recursive = TRUE)
```

---

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. It will put this `binaryLinux` in the first position. If the `getOption("repos")` is `NULL`, it will put `backupCRAN` in second position.

Usage

```
setLinuxBinaryRepo(
  binaryLinux = urlForArchivedPkgs,
  backupCRAN = srcPackageURLOnCRAN
)
```

Arguments

`binaryLinux`	A CRAN repository serving binary Linux packages.
`backupCRAN`	If there is no CRAN repository set

Value

Called for its side effect on `options("repos")`; returns that `options()` result invisibly, or `NULL` when nothing needed changing.

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

Description

`setup` and `setupOff` are currently deprecated. These may be re-created in a future version. In its place, a user can simply put `.libPaths(libs, include.site = FALSE)` in their `.Rprofile` file, where `libs` is the directory where the packages should be installed and should be a folder with the R version number, e.g., derived by using `checkLibPaths(libs)`.

Usage

```
setup(
  newLibPaths,
  RPackageFolders,
  RPackageCache = cacheGetOptionCachePkgDir(),
  standAlone = getOption("Require.standAlone", TRUE),
  verbose = getOption("Require.verbose")
)

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

Arguments

`newLibPaths`	Same as `RPackageFolders`. This is for more consistent naming with `Require(..., libPaths = ...)`.
`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`.
`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 -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in `Require` function, when `verbose >= 2`, also returns details as if `returnDetails = TRUE` (for backwards compatibility).

`removePackages` Deprecated. Please remove packages manually from `.libPaths()`

Value

Nothing (`invisible()`); called for its side effects on `.libPaths()`, the package cache and options. Deprecated.

`sourcePkgs` *A list of R packages that should likely be installed from Source, not Binary*

Description

The list of R packages that `Require` installs from source on Linux, even if the `getOptions("repos")` is a binary repository. This list can be updated by the user by modifying the options `Require.spatialPkgs` or `Require.otherPkgs`. Default "force source only packages" are visible with `RequireOptions()`.

Usage

```
sourcePkgs(additional = NULL, spatialPkgs = NULL, otherPkgs = NULL)
```

Arguments

`additional`	Any other packages to be added to the other 2 argument vectors
`spatialPkgs`	A character vector of package names that focus on spatial analyses.
`otherPkgs`	A character vector of package names that often require system specific compilation.

Value

A sorted concatenation of the 3 input parameters.

`splitKeepOrderAndDTIntegrity` `split` *for a data.table that keeps integrity of a column of lists of data.table objects*

Description

`data.table::split` does 2 bad things:

1. reorders if using `f`
2. destroys the integrity of a column that is a list of data.tables, when using by So, to keep order, need by, but to keep integrity, need `f`. This function

Usage

```
splitKeepOrderAndDTIntegrity(pkgDT, splitOn)
```

Arguments

pkgDT	A pkgDT object e.g., from toPkgDT
splitOn	Character vector passed to data.table::split(..., f = splitOn)

Value

A list of data.table objects of length(unique(splitOn)).

---

sysInstallAndDownload *download.files or install.packages in a separate process*

Description

This uses sys package so that messaging can be controlled. This also provides the option to parallelize by spawning multiple background process to allow parallel e.g., downloads. Noting that if libcurl is installed (and detected using capabilities("libcurl")), then no explicit parallelism will be allowed, instead method = "libcurl" will be passed enabling parallel downloads.

Usage

```
sysInstallAndDownload(
  args,
  splitOn = "pkgs",
  doLine = "outfiles <- do.call(download.packages, args)",
  returnOutfile = FALSE,
  doLineVectorized = TRUE,
  tmpdir,
  libPaths,
  verbose
)
```

Arguments

args	A list with all arguments for a do.call to either download.file, install.packagesor a custom other func
splitOn	A character vector of the names in args to parallelize over. Defaults to pkgs. All other named elements in args will be assumed to be length 1 and used for every parallel process.
doLine	A character string with the "outfiles <- do.call(..., args)" line.
returnOutfile	A logical. If TRUE, then the names of the outfiles will be returned.

doLineVectorized
: A logical. If TRUE, and parallism is being used, this indicates that the doLine is a function that allows for multiple elements in args[[splitOn[[1]]]]. If FALSE, the function will make multiple sequential calls within each parallel process to the doLine call.

tmpdir
: A single path where all downloads will be put

libPaths
: The library path (or libraries) where all packages should be installed, and looked for to load (i.e., call library). This can be used to create isolated, stand alone package installations, if used with standAlone = TRUE. Currently, the path supplied here will be prepended to .libPaths() (temporarily during this call) to Require if standAlone = FALSE or will set (temporarily) .libPaths() to c(libPaths, tail(libPaths(), 1) to keep base packages.

verbose
: Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

Value

Mostly for side effects, namely installed packages or downloaded packages or files. However, in the case of returnOutfile = TRUE, then a list of filenames will be returned with any outputs from the doLine.

---

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()),
  create = TRUE
)
```

Arguments

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

tempdir
: Optional character string where the temporary dir should be placed. Defaults to .RequireTempPath()

create
: Logical. Should the directory be created. Default TRUE

Value

The normalized path to the temporary (sub-)directory, created when `create = TRUE`.

See Also

`tempfile2()`

`tempfile2` *Make a temporary subfile in a temporary (sub-)directory*

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 `file.path("smth", "smth2")` for nested temporary sub directories.
`tempdir`	Optional character string where the temporary dir should be placed. Defaults to `.RequireTempPath()`
`...`	passed to `tempfile`, e.g., `fileext`

Value

The normalized path to a temporary file inside `tempdir2(sub)`; the file is not created.

See Also

`tempdir2()`

`trimRedundancies` *Collapse redundant package specifications*

Description

Given package specifications that may name the same package more than once – e.g. `"pkg"`, `"pkg (>= 1.0)"` and `"user/pkg@branch"` – reduce them to one entry per package, keeping the most specific requirement. This is the reduction `Require()` applies to a dependency set before installing, exposed because packages that assemble their own `reqdPkgs`-style lists need the same rule to agree with what `Require()` will actually do.

Usage

```
trimRedundancies(
  pkgInstall,
  repos = NULL,
  purge = NULL,
  libPaths = NULL,
  verbose = getOption("Require.verbose"),
  type = getOption("pkgType")
)
```

Arguments

`pkgInstall`	Package specifications: a character vector, or a `data.table` as produced internally by `Require`.
`repos, purge, libPaths`	Unused; retained so existing positional calls keep working.
`verbose`	Numeric or logical, controlling messaging verbosity.
`type`	Package type, as in `utils::install.packages()`.

Value

A `data.table` with one row per package, with redundant entries removed. Callers wanting the specifications back as text take the `packageFullName` column.

See Also

Other version specifications: `compareVersion2()`, `extractPkgName()`, `parseGitHub()`, `trimVersionNumber()`

Examples

```
trimRedundancies(c("data.table", "data.table (>= 1.14.0)"))
```

trimVersionNumber *Trim version number off a compound package name*

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

Value

A character vector the length of pkgs, each element with its version specification (and any pak source prefix) removed.

See Also

extractPkgName()

Other version specifications: compareVersion2(), extractPkgName(), parseGitHub(), trimRedundancies()

Examples

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

updatePackages *Update installed packages with latest available versions*

Description

Similar to update.packages, but works for archived, non-archived, and Github packages.

Usage

```
updatePackages(
  libPaths = .libPaths()[1],
  purge = FALSE,
  verbose = getOption("Require.verbose")
)
```

Arguments

libPaths	The library to update; defaults to .libPaths()[1]
purge	Logical. Should the assessment of installed.packages purge the cached version. Default is FALSE
verbose	Numeric or logical indicating how verbose should the function be. If -1 or -2, then as little verbosity as possible. If 0 or FALSE, then minimal outputs; if 1 or TRUE, more outputs; 2 even more. NOTE: in Require function, when verbose >= 2, also returns details as if returnDetails = TRUE (for backwards compatibility).

Value

Run for its side effect, namely, updating installed packages to their latest possible state, whether they are on CRAN currently, archived, or on GitHub.

Index