Package 'miniCRAN'

September 15, 2020

Version 0.2.14

License GPL-2

Copyright Andrie de Vries, Microsoft Corporation

Title Create a Mini Version of CRAN Containing Only Selected Packages

Description Makes it possible to create an internally consistent repository consisting of selected packages from CRAN-like repositories. The user specifies a set of desired packages, and 'miniCRAN' recursively reads the dependency tree for these packages, then downloads only this subset. The user can then install packages from this repository directly, rather than from CRAN. This is useful in production settings, e.g. server behind a firewall, or remote locations with slow (or zero) Internet access.

URL https://github.com/andrie/miniCRAN

BugReports https://github.com/andrie/miniCRAN/issues

Imports graphics, httr, methods, stats, tools, utils, igraph, assertthat (>= 0.2.0)

Suggests devtools, knitr, rmarkdown, testthat (>= 2.1.0), covr, withr, mockery, testthis, roxygen2

LazyData true

LazyLoad true

VignetteBuilder knitr

RoxygenNote 7.1.1

Encoding UTF-8

Language en-GB

Config/testthat/edition 3

NeedsCompilation no

Author Andrie de Vries [aut, cre, cph], Alex Chubaty [ctb], Microsoft Corporation [cph]

Maintainer Andrie de Vries <apdevries@gmail.com>

Repository CRAN

Date/Publication 2020-09-15 19:10:03 UTC

R topics documented:

miniCRAN-package . . . 2
.listFiles . . . 4
addLocalPackage . . . 4
addOldPackage . . . 6
addPackage . . . 8
addPackageListingGithub . . . 10
basePkgs . . . 11
checkVersions . . . 12
cranJuly2014 . . . 14
getCranDescription . . . 14
makeDepGraph . . . 15
makeLibrary . . . 16
makeRepo . . . 17
pkgAvail . . . 19
pkgDep . . . 20
plot.pkgDepGraph . . . 22
repoBinPath . . . 23
repoPrefix . . . 24
twodigitRversion . . . 25
updatePackages . . . 26

Index **29**

`miniCRAN-package` *description*

Description

At the end of 2014, CRAN consisted of more than 6,000 packages. Many organisations need to maintain a private mirror of CRAN, but with only a subset of packages that are relevant to them.

Details

`miniCRAN` makes it possible to create an internally consistent repository consisting of selected packages from CRAN-like repositories. The user specifies a set of desired packages, and miniCRAN recursively reads the dependency tree for these packages, then downloads only this subset.

There are many reasons for not creating a complete mirror CRAN using `rsync`:

- You may wish to mirror only a subset of CRAN, for security, legal compliance or any other in-house reason
- You may wish to restrict internal package use to a subset of public packages, to minimize package duplication, or other reasons of coding standards
- You may wish to make packages available from public repositories other than CRAN, e.g. BioConductor, r-forge, OmegaHat, etc.
- You may wish to add custom in-house packages to your repository

The ambition of `miniCRAN` is to eventually satisfy all of these considerations.

Making a private repo

- `pkgAvail()`: Read from a local (or remote) CRAN-like repository and determine available packages.
- `pkgDep()`: Find (recursive) package dependencies.
- `makeRepo()` : Make a mini CRAN repository, by downloading packages (and their dependencies) and creating the appropriate file structure for a repository. This allows you to use functions like `utils::available.packages()` and `utils::install.packages()` on your local repository.

This subset will be internally consistent, i.e. the following functions will work as expected:

- `utils::available.packages()`
- `utils::install.packages()`

The main function is `makeRepo()` - this will download all the required packages, with their dependencies, into the appropriate repository file structure, and then create the repository index (PACKAGES) file.

Updating packages in a repo

- `oldPackages()`: Indicates packages which have a (suitable) later version on the repositories * `updatePackages()`: Offers to download and install such packages

Creating dependencies

To get a recursive list of dependencies as well as a plot, use `pkgDep()` followed by `makeDepGraph()`.

- `pkgDep()`: Find (recursive) package dependencies.
- `makeDepGraph()`: Create graph of selected package dependencies.
- `plot.pkgDepGraph()`: Create a visualization of the dependency graph

Package options

`minicran.mran` preferred MRAN URL. Defaults to `https://mran.microsoft.com` for R versions 3.2.2 and greater. Versions earlier than 3.2.2 use HTTP instead of HTTPS.

Author(s)

Maintainer: Andrie de Vries <apdevries@gmail.com> [copyright holder]

Other contributors:

- Alex Chubaty <alex.chubaty@gmail.com> [contributor]
- Microsoft Corporation [copyright holder]

See Also

Useful links:

- `https://github.com/andrie/miniCRAN`
- Report bugs at `https://github.com/andrie/miniCRAN/issues`

.listFiles *List pre-built packages in a directory based on file extension*

Description

List pre-built packages in a directory based on file extension

Usage

```
.listFiles(pkgs, path, type)
```

Arguments

pkgs	Character vector of package names
path	Character string specifying the directory containing packages to be added.
type	Character indicating the package type (e.g., "source", "win.binary", etc.).

Value

Installs the packages and returns the new package index.

Examples

```
## Not run:
 .listFiles('path/to/my/packages', type = "source")

## End(Not run)
```

addLocalPackage *Add local packages to a miniCRAN repository.*

Description

Examine the contents of a directory specified by pkgPath for pre-built packages matching the names specified by pkgs, and add these to the miniCRAN repository.

Usage

```
addLocalPackage(
  pkgs = NULL,
  pkgPath = NULL,
  path = NULL,
  type = "source",
  Rversion = R.version,
  writePACKAGES = TRUE,
```

```
  deps = FALSE,
  quiet = FALSE,
  build = FALSE
)
```

Arguments

pkgs	Character vector of packages to download
pkgPath	Character vector of directory location containing packages to be added. Note that pkgPath should be the parent directory of the package (i.e., the package directory path is constructed from file.path(pkgPath,pkgs)).
path	Destination download path. This path is the root folder of your new repository.
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to download.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version
writePACKAGES	If TRUE, calls write_PACKAGES() to update the repository PACKAGES file.
deps	Not used. See note.
quiet	If TRUE, suppress status messages (if any), and the progress bar during download.
build	Logical indicating whether packages should be build prior to adding.

Details

To build a package from source and then add it, use build = TRUE. Note that package development libraries and the devtools package must be installed on your system in order to build packages.

Value

Installs the packages and returns the new package index.

Note

Currently, adding local packages does not check nor download their dependencies.

Author(s)

Alex Chubaty

Examples

```
## Not run:
 addLocalPackage("myPackage", "path/to/my/prebuilt/package",
                 "path/to/my/miniCRAN/repo")

 addLocalPackage("myPackage", "path/to/my/package/sourcecode",
                 "path/to/my/miniCRAN/repo", build = TRUE)

## End(Not run)
```

`addOldPackage` *Add old package versions to a miniCRAN repository.*

Description

Will download and add older source package versions. Older binary versions are not normally available on CRAN and should be built from source on the platform for which they are required. As such, specifying `type!="source"` will likely fail as the download will not be successful.

Usage

```
addOldPackage(
  pkgs = NULL,
  path = NULL,
  vers = NULL,
  repos = getOption("repos"),
  type = "source",
  Rversion = R.version,
  writePACKAGES = TRUE,
  deps = FALSE,
  quiet = TRUE
)
```

Arguments

`pkgs`	Character vector of packages to download
`path`	Destination download path. This path is the root folder of your new repository.
`vers`	The package version to install.
`repos`	URL(s) of the 'contrib' sections of the repositories, e.g. `"http://cran.us.r-project.org"`. Passed to `available.packages()`
`type`	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to `download.packages()`.
`Rversion`	Version of R (only used if `type` is not `source`.) Defaults to R.version, but this can be specified as any of the following formats:

- a character string with the two digit R version, e.g. "3.1"
- a list with components `major` and `minor`
- the result of `getRversion()`
- the result of R.version

`writePACKAGES`	If TRUE, calls `write_PACKAGES()` to update the repository PACKAGES file.
`deps`	logical indicating whether the package dependencies should be added (default `TRUE`).
`quiet`	If TRUE, suppress status messages (if any), and the progress bar during download.

Value

Adds the packages, rebuilds the package index, and invisibly returns the number of packages written to the index files.

Note

Dependencies for old package versions cannot be determined automatically and must be specified by the user in `pkgs` and `vers`. Thus, `deps=FALSE` is the default for this function.

See Also

Other update repo functions: `addPackage()`, `checkVersions()`, `makeRepo()`, `updatePackages()`

Examples

```
### `checkVersions` and `add.packages.miniCRAN` require an existing miniCRAN repo

# Specify list of packages to download
revolution <- c(CRAN = getOption("minicran.mran"))
revolution
pkgs <- c("foreach")
pkgTypes <- c("source", "win.binary")

pdb <- cranJuly2014

## Not run:
  pdb <- pkgAvail(repos = revolution, type = "source")

## End(Not run)

pkgList <- pkgDep(pkgs, availPkgs = pdb, repos = revolution, type = "source", suggests = FALSE)
pkgList

## Not run:
  # Create temporary folder for miniCRAN
  dir.create(pth <- file.path(tempdir(), "miniCRAN"))

  # Make repo for source and win.binary
  makeRepo(pkgList, path = pth, repos = revolution, type = pkgTypes)
```

```
  # Add other versions of a package (and assume these were added previously)
  oldVers <- data.frame(package = c("foreach", "codetools", "iterators"),
                        version = c("1.4.0", "0.2-7", "1.0.5"),
                        stringsAsFactors = FALSE)
  pkgs <- oldVers$package
  addOldPackage(pkgs, path = pth, vers = oldVers$version, repos = revolution, type = "source")
  # NOTE: older binary versions would need to be build from source

  # List package versions in the miniCRAN repo (produces warning about duplicates)
  pkgVersionsSrc <- checkVersions(pkgs, path = pth, type = "source")
  pkgVersionsBin <- checkVersions(pkgs, path = pth, type = "win.binary")

  # After inspecting package versions, remove old versions
  basename(pkgVersionsSrc) # "foreach_1.4.0.tar.gz"  "foreach_1.4.2.tar.gz"
  basename(pkgVersionsBin) # "foreach_1.4.0.zip"     "foreach_1.4.2.zip"
  file.remove(c(pkgVersionsSrc[1], pkgVersionsBin[1]))

  # Rebuild package index after adding/removing files
  updateRepoIndex(pth, type = pkgTypes, Rversion = R.version)

  pkgAvail(pth, type = "source")

  # Add new packages (from CRAN) to the miniCRAN repo
  addPackage("Matrix", path = pth, repos = revolution, type = pkgTypes)

  # Delete temporary folder
  unlink(pth, recursive = TRUE)

## End(Not run)
```

---

addPackage *Add packages to a miniCRAN repository.*

Description

Add packages to a miniCRAN repository.

Usage

```
addPackage(
  pkgs = NULL,
  path = NULL,
  repos = getOption("repos"),
  type = "source",
  Rversion = R.version,
  writePACKAGES = TRUE,
  deps = TRUE,
  quiet = FALSE
)
```

Arguments

pkgs	Character vector of packages to download
path	Destination download path. This path is the root folder of your new repository.
repos	URL(s) of the 'contrib' sections of the repositories, e.g. "http://cran.us.r-project.org". Passed to available.packages()
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to download.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version
writePACKAGES	If TRUE, calls write_PACKAGES() to update the repository PACKAGES file.
deps	logical indicating whether the package dependencies should be added (default TRUE).
quiet	If TRUE, suppress status messages (if any), and the progress bar during download.

Value

Installs the packages, rebuilds the package index, and invisibly returns the number of packages written to the index files.

See Also

Other update repo functions: addOldPackage(), checkVersions(), makeRepo(), updatePackages()

Examples

```
### `checkVersions` and `add.packages.miniCRAN` require an existing miniCRAN repo

# Specify list of packages to download
revolution <- c(CRAN = getOption("minicran.mran"))
revolution
pkgs <- c("foreach")
pkgTypes <- c("source", "win.binary")

pdb <- cranJuly2014

## Not run:
  pdb <- pkgAvail(repos = revolution, type = "source")

## End(Not run)

pkgList <- pkgDep(pkgs, availPkgs = pdb, repos = revolution, type = "source", suggests = FALSE)
```

```
pkgList

## Not run:
  # Create temporary folder for miniCRAN
  dir.create(pth <- file.path(tempdir(), "miniCRAN"))

  # Make repo for source and win.binary
  makeRepo(pkgList, path = pth, repos = revolution, type = pkgTypes)

  # Add other versions of a package (and assume these were added previously)
  oldVers <- data.frame(package = c("foreach", "codetools", "iterators"),
                        version = c("1.4.0", "0.2-7", "1.0.5"),
                        stringsAsFactors = FALSE)
  pkgs <- oldVers$package
 addOldPackage(pkgs, path = pth, vers = oldVers$version, repos = revolution, type = "source")
  # NOTE: older binary versions would need to be build from source

  # List package versions in the miniCRAN repo (produces warning about duplicates)
  pkgVersionsSrc <- checkVersions(pkgs, path = pth, type = "source")
  pkgVersionsBin <- checkVersions(pkgs, path = pth, type = "win.binary")

  # After inspecting package versions, remove old versions
  basename(pkgVersionsSrc) # "foreach_1.4.0.tar.gz"  "foreach_1.4.2.tar.gz"
  basename(pkgVersionsBin) # "foreach_1.4.0.zip"     "foreach_1.4.2.zip"
  file.remove(c(pkgVersionsSrc[1], pkgVersionsBin[1]))

  # Rebuild package index after adding/removing files
  updateRepoIndex(pth, type = pkgTypes, Rversion = R.version)

  pkgAvail(pth, type = "source")

  # Add new packages (from CRAN) to the miniCRAN repo
  addPackage("Matrix", path = pth, repos = revolution, type = pkgTypes)

  # Delete temporary folder
  unlink(pth, recursive = TRUE)

## End(Not run)
```

---

addPackageListingGithub *Add DESCRIPTION information from package on github.*

Description

Downloads the DESCRIPTION file from a package on github, parses the fields and adds (or replaces) a row in the available package database.

Usage

```
addPackageListingGithub(
  pdb = pkgAvail(),
  repo,
  username = NULL,
  branch = "master"
)
```

Arguments

pdb	Package database, usually the result of pkgAvail() or available.packages()
repo	Character vector. Name of repository on github, e.g. "andrie/rrd"
username	Optional character vector. Name of repository on github, e.g. "andrie/rrd"
branch	name of branch, defaults to "master"

Examples

```
# Create package database
pdb <- cranJuly2014

## Not run:
  pdb <- pkgAvail(repos = c(CRAN = getOption("minicran.mran")))

  # Overwrite pdb with development version of miniCRAN at github
  newpdb <- addPackageListingGithub(pdb = pdb, "andrie/miniCRAN")
  newpdb["miniCRAN", ]

  # Add package from github that's not currently on CRAN
  newpdb <- addPackageListingGithub(pdb = pdb, repo = "RevolutionAnalytics/checkpoint")
  newpdb["checkpoint", ]

  set.seed(1)
  plot(makeDepGraph("checkpoint", availPkgs = newpdb, suggests = TRUE))

## End(Not run)
```

---

basePkgs *Returns names of base packages.*

Description

Retrieves names of installed packages by calling utils::installed.packages() and returning only those packages where Priority == "base".

Usage

```
basePkgs()
```

See Also

pkgDep()

Other dependency functions: makeDepGraph(), pkgDep(), plot.pkgDepGraph()

---

checkVersions *Check for previous versions of packages in a miniCRAN repository.*

Description

Checks for previous versions, and returns the file paths for packages with multiple versions. You can subsequently decide which version to keep.

Usage

```
checkVersions(pkgs = NULL, path = NULL, type = "source", Rversion = R.version)
```

Arguments

pkgs	Character vector of packages to be installed. If not provided, checks all files for multiple package versions.
path	The local path to the directory where the miniCRAN repo resides.
type	character, indicating the type of package to download and install. See install.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version

Value

Returns invisibly the file paths to packages with multiple versions for removal.

list with an element for each type, consisting of a character vector of download paths

See Also

Other update repo functions: addOldPackage(), addPackage(), makeRepo(), updatePackages()

Examples

```
### `checkVersions` and `add.packages.miniCRAN` require an existing miniCRAN repo

# Specify list of packages to download
revolution <- c(CRAN = getOption("minicran.mran"))
revolution
pkgs <- c("foreach")
pkgTypes <- c("source", "win.binary")

pdb <- cranJuly2014

## Not run:
  pdb <- pkgAvail(repos = revolution, type = "source")

## End(Not run)

pkgList <- pkgDep(pkgs, availPkgs = pdb, repos = revolution, type = "source", suggests = FALSE)
pkgList

## Not run:
  # Create temporary folder for miniCRAN
  dir.create(pth <- file.path(tempdir(), "miniCRAN"))

  # Make repo for source and win.binary
  makeRepo(pkgList, path = pth, repos = revolution, type = pkgTypes)

  # Add other versions of a package (and assume these were added previously)
  oldVers <- data.frame(package = c("foreach", "codetools", "iterators"),
                        version = c("1.4.0", "0.2-7", "1.0.5"),
                        stringsAsFactors = FALSE)
  pkgs <- oldVers$package
 addOldPackage(pkgs, path = pth, vers = oldVers$version, repos = revolution, type = "source")
  # NOTE: older binary versions would need to be build from source

  # List package versions in the miniCRAN repo (produces warning about duplicates)
  pkgVersionsSrc <- checkVersions(pkgs, path = pth, type = "source")
  pkgVersionsBin <- checkVersions(pkgs, path = pth, type = "win.binary")

  # After inspecting package versions, remove old versions
  basename(pkgVersionsSrc) # "foreach_1.4.0.tar.gz"  "foreach_1.4.2.tar.gz"
  basename(pkgVersionsBin) # "foreach_1.4.0.zip"     "foreach_1.4.2.zip"
  file.remove(c(pkgVersionsSrc[1], pkgVersionsBin[1]))

  # Rebuild package index after adding/removing files
  updateRepoIndex(pth, type = pkgTypes, Rversion = R.version)

  pkgAvail(pth, type = "source")

  # Add new packages (from CRAN) to the miniCRAN repo
  addPackage("Matrix", path = pth, repos = revolution, type = pkgTypes)

  # Delete temporary folder
```

```
  unlink(pth, recursive = TRUE)

## End(Not run)
```

`cranJuly2014` *Stored version of available.packages()*

Description

Copy of the result of `utils::available.packages()` on July 1, 2014.

Usage

```
cranJuly2014
```

Format

matrix

`getCranDescription` *Obtains DESCRIPTION metadata from CRAN for each package.*

Description

This is a wrapper around `tools::CRAN_package_db` and may be deprecated in future versions of the package.

Usage

```
getCranDescription(
  pkg,
  repos = getOption("repos"),
  type = "source",
  pkgs = pkgDep(pkg, repos = repos, type = type)
)
```

Arguments

`pkg`	Character vector of packages.
`repos`	URL(s) of the 'contrib' sections of the repositories, e.g. `"http://cran.us.r-project.org"`. Passed to `available.packages()`
`type`	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to `download.packages()`.
`pkgs`	Character vector of packages to download

Examples

```
## Not run:
getCranDescription(c("igraph", "ggplot2", "XML"),
  repos = c(CRAN = getOption("minicran.mran"))
)

## End(Not run)
```

`makeDepGraph` *Create dependency graph from available packages.*

Description

Each package is a node, and a dependency is an edge

Usage

```
makeDepGraph(
  pkg,
  availPkgs,
  repos = getOption("repos"),
  type = "source",
  suggests = TRUE,
  enhances = FALSE,
  includeBasePkgs = FALSE,
  ...
)
```

Arguments

`pkg`	Character vector of packages.
`availPkgs`	Data frame with an element called package. The package element is a vector of available packages. Defaults to reading this list from CRAN, using `available.packages()`
`repos`	URL(s) of the 'contrib' sections of the repositories, e.g. `"http://cran.us.r-project.org"`. Passed to `available.packages()`
`type`	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to `download.packages()`.
`suggests`	If TRUE, retrieves Suggests dependencies (non-recursively)
`enhances`	If TRUE, retrieves Enhances dependencies (non-recursively)
`includeBasePkgs`	If TRUE, include base R packages in results
`...`	Other arguments passed to `available.packages()`

See Also

pkgDep() to extract package dependencies

Other dependency functions: basePkgs(), pkgDep(), plot.pkgDepGraph()

Examples

```
availPkgs <- cranJuly2014

## Not run:
availPkgs <- pkgAvail(
  repos = c(CRAN = getOption("minicran.mran")),
  type = "source"
  )

## End(Not run)


# Create dependency graph using stored database of available packages
p <- makeDepGraph(
  c("ggplot2", "forecast"),
  availPkgs = availPkgs
)

if(require(igraph)) plot(p)



## Not run:
  # Create dependency graph using newly retrieved database from CRAN

  p <- makeDepGraph(
  c("ggplot2", "forecast"),
  repos = c(CRAN = getOption("minicran.mran")),
  type = "source"
)
if(require(igraph)) plot(p)

## End(Not run)
```

---

makeLibrary *Deprecated function to download packages to local folder.*

Description

Deprecated function to download packages to local folder.

Usage

```
makeLibrary(pkgs, path, type = "source")
```

Arguments

pkgs	Character vector of packages to download
path	Destination download path. This path is the root folder of your new repository.
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to `download.packages()`.

---

makeRepo *Downloads packages from CRAN to specified path and creates a local repository.*

Description

Given a list of packages, downloads these packages to a specified destination folder using the required CRAN folder structure, and finally creates the PACKAGES index file. Since the folder structure mimics the required structure and files of a CRAN repository, it supports functions like `utils::install.packages()`.

Usage

```
makeRepo(
  pkgs,
  path,
  repos = getOption("repos"),
  type = "source",
  Rversion = R.version,
  download = TRUE,
  writePACKAGES = TRUE,
  quiet = FALSE
)

updateRepoIndex(path, type = "source", Rversion = R.version)
```

Arguments

pkgs	Character vector of packages to download
path	Destination download path. This path is the root folder of your new repository.
repos	URL(s) of the 'contrib' sections of the repositories, e.g. `"http://cran.us.r-project.org"`. Passed to `available.packages()`
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to `download.packages()`.

Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version
download	If TRUE downloads packages.
writePACKAGES	If TRUE, calls write_PACKAGES() to update the repository PACKAGES file.
quiet	If TRUE, suppress status messages (if any), and the progress bar during download.

Value

character vector of downloaded package files

Repo folder structure

A repository has two main folders, one for source packages, and the other for binary packages. Inside the binary package folder, bin, you will find subfolders for Windows as well as the various OSX binaries.

```
+-Root
...+-src/contrib
......+-PACKAGES
..+-bin
.......+-windows/contrib/version
..........+-PACKAGES
.......+-macosx/contrib/version
..........+-PACKAGES
.......+-macosx/mavericks/contrib/version
..........+-PACKAGES
.......+-macosx/leopard/contrib/version
..........+-PACKAGES
```

Note

Internally makes use of utils::download.packages() and write_PACKAGES()

See Also

Other update repo functions: addOldPackage(), addPackage(), checkVersions(), updatePackages()

Examples

```
# Specify list of packages to download
revolution <- c(CRAN = getOption("minicran.mran"))
pkgs <- c("foreach")

pdb <- cranJuly2014

## Not run:
  pdb <- pkgAvail(
    repos = c(CRAN = getOption("minicran.mran")),
    type = "source"
  )

## End(Not run)

pkgList <- pkgDep(pkgs, availPkgs = pdb, repos = revolution,
                  type = "source", suggests = FALSE)
pkgList


## Not run:
# Create temporary folder for miniCRAN
dir.create(pth <- file.path(tempdir(), "miniCRAN"))

# Make repo for source and win.binary
makeRepo(pkgList, path = pth, repos = revolution, type = "source")

# List all files in miniCRAN
list.files(pth, recursive = TRUE)

# Check for available packages
pkgAvail(repos = pth, type = "source")

# Repeat process for windows binaries
makeRepo(pkgList, path = pth, repos = revolution, type = "win.binary")
pkgAvail(repos = pth, type = "win.binary")

# Delete temporary folder
unlink(pth, recursive = TRUE)

## End(Not run)
```

---

pkgAvail *Reads available packages from CRAN repository.*

Description

This is a thin wrapper around `utils::available.packages()`. If the argument `path` is supplied, then the function attempts to read from a local repository, otherwise attempts to read from a CRAN mirror at the `repos` url.

Usage

```
pkgAvail(
  repos = getOption("repos"),
  type = "source",
  Rversion = R.version,
  quiet = FALSE
)
```

Arguments

repos	URL(s) of the 'contrib' sections of the repositories, e.g. "http://cran.us.r-project.org". Passed to available.packages()
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to download.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version
quiet	If TRUE, suppresses warnings

See Also

pkgDep()

---

pkgDep *Retrieves package dependencies.*

Description

Performs recursive retrieve for Depends, Imports and LinkLibrary. Performs non-recursive retrieve for Suggests.

Usage

```
pkgDep(
  pkg,
  availPkgs,
  repos = getOption("repos"),
  type = "source",
  depends = TRUE,
  suggests = TRUE,
  enhances = FALSE,
```

```
  includeBasePkgs = FALSE,
  Rversion = R.version,
  quiet = FALSE,
  ...
)
```

Arguments

pkg	Character vector of packages.
availPkgs	Data frame with an element called package. The package element is a vector of available packages. Defaults to reading this list from CRAN, using available.packages()
repos	URL(s) of the 'contrib' sections of the repositories, e.g. "http://cran.us.r-project.org". Passed to available.packages()
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to download.packages().
depends	If TRUE, retrieves Depends, Imports and LinkingTo dependencies (non-recursively)
suggests	If TRUE, retrieves Suggests dependencies (non-recursively)
enhances	If TRUE, retrieves Enhances dependencies (non-recursively)
includeBasePkgs	If TRUE, include base R packages in results
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version
quiet	If TRUE, suppresses warnings
...	Other arguments passed to available.packages()

Value

character vector of package names

See Also

Other dependency functions: basePkgs(), makeDepGraph(), plot.pkgDepGraph()

Examples

```
## Not run:
pkgDep(pkg = c("ggplot2", "plyr", "reshape2"),
       repos = c(CRAN = getOption("minicran.mran"))
)
```

```
## End(Not run)

pdb <- cranJuly2014
## Not run:
pdb <- pkgAvail(repos = c(CRAN = getOption("minicran.mran")))

## End(Not run)

pkgDep(pkg = c("ggplot2", "plyr", "reshape2"), pdb)
```

`plot.pkgDepGraph` *Plots a package dependency graph.*

Description

Plots a package dependency graph.

Usage

```
## S3 method for class 'pkgDepGraph'
plot(
  x,
  pkgsToHighlight,
  main = paste(attr(x, "pkgs"), collapse = ", "),
  legendPosition = c(-1.2, -1),
  shape = "circle",
  vertex.size = 8,
  cex = 1,
  ...
)
```

Arguments

`x`	Object to plot
`pkgsToHighlight`	Optional character vector with names of package to highlight. If missing, defaults to packages used in original call to `makeDepGraph()`
`main`	Title of plot
`legendPosition`	Numeric vector of length 2, indicating (x, y) position of edge legend. Both values should be in the range [-1; 1]. If `NULL`, the edge legend is not displayed.
`shape`	Shape of edge. See `igraph::igraph.plotting()`. Could be "none", "circle", "square", ...
`vertex.size`	Size of vertex shape. `igraph::igraph.plotting()`
`cex`	Vertex label size.
`...`	Ignored

See Also

Other dependency functions: basePkgs(), makeDepGraph(), pkgDep()

Examples

```
tags <- "chron"

# Plot using defaults
pdb <- cranJuly2014

## Not run:
  pdb <- pkgAvail(
    repos = c(CRAN = getOption("minicran.mran")),
    type = "source"
  )

## End(Not run)

dg <- makeDepGraph(tags, availPkgs = pdb  , includeBasePkgs = FALSE,
                   suggests = TRUE, enhances = TRUE)

set.seed(42);
plot(dg)

# Move edge legend to top left
set.seed(42);
plot(dg, legendPosition = c(-1, 1))

# Change font size and shape size
set.seed(42);
plot(dg, legendPosition = c(-1, 1), vertex.size = 20,  cex = 0.5)


# Move vertex legend to top right
set.seed(42);
plot(dg, legendPosition = c(1, 1), vertex.size = 20,  cex = 0.5)
```

---

repoBinPath *Construct path to full binary location*

Description

Construct path to full binary location

Usage

```
repoBinPath(path, type, Rversion)
```

Arguments

path	Destination download path. This path is the root folder of your new repository.
type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to download.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version

---

repoPrefix *Get the path to the repo directory containing the package files.*

Description

Get the path to the repo directory containing the package files.

Usage

```
repoPrefix(type, Rversion)
```

Arguments

type	character, indicating the type of package to download and install. See install.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version

Value

The file path to the package files directory.

Repo folder structure

A repository has two main folders, one for source packages, and the other for binary packages. Inside the binary package folder, bin, you will find subfolders for Windows as well as the various OSX binaries.

```
+-Root
...+-src/contrib
```

```
......+-PACKAGES
..+-bin
.......+-windows/contrib/version
..........+-PACKAGES
.......+-macosx/contrib/version
..........+-PACKAGES
.......+-macosx/mavericks/contrib/version
..........+-PACKAGES
.......+-macosx/leopard/contrib/version
..........+-PACKAGES
```

Note

Not all versions of R are compatible with with all package types (e.g., `mac.binary.el-capitan` is only valid for R > 3.4.0).

`twodigitRversion` *Get a two-digit version of the R version*

Description

Get a two-digit version of the R version

Usage

```
twodigitRversion(Rversion = R.version)
```

Arguments

`Rversion`	Version of R (only used if `type` is not `source`.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components `major` and `minor` • the result of `getRversion()` • the result of R.version

Value

A character string representing the two-digit R version.

updatePackages *Check for available package updates in a miniCRAN repo.*

Description

`oldPackages()` indicates packages which have a (suitable) later version on the repositories whereas `updatePackages()` offers to download and install such packages.

Usage

```
oldPackages(
  path = NULL,
  repos = getOption("repos"),
  availPkgs = pkgAvail(repos = repos, type = type, Rversion = Rversion),
  method,
 availableLocal = pkgAvail(repos = path, type = type, Rversion = Rversion, quiet =
    quiet),
  type = "source",
  Rversion = R.version,
  quiet = FALSE
)

updatePackages(
  path = NULL,
  repos = getOption("repos"),
  method = NULL,
  ask = TRUE,
  availPkgs = pkgAvail(repos = repos, type = type, Rversion = Rversion),
  oldPkgs = NULL,
  type = "source",
  Rversion = R.version,
  quiet = FALSE
)
```

Arguments

`path`	Destination download path. This path is the root folder of your new repository.
`repos`	URL(s) of the 'contrib' sections of the repositories, e.g. `"http://cran.us.r-project.org"`. Passed to `available.packages()`
`availPkgs`	Data frame with an element called package. The package element is a vector of available packages. Defaults to reading this list from CRAN, using `available.packages()`
`method`	Download method, see `download.file()`.
`availableLocal`	all packages hosted in the miniCRAN repo, as returned by `pkgAvail()`. A subset can be specified; currently this must be in the same (character matrix) format as returned by `pkgAvail()`.

type	Possible values are (currently) "source", "mac.binary" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms. Passed to download.packages().
Rversion	Version of R (only used if type is not source.) Defaults to R.version, but this can be specified as any of the following formats: • a character string with the two digit R version, e.g. "3.1" • a list with components major and minor • the result of getRversion() • the result of R.version
quiet	If TRUE, suppress status messages (if any), and the progress bar during download.
ask	logical indicating whether to ask user before packages are actually downloaded and installed. Alternatively, the value "graphics" starts an interactive widget to allow the user to (de-)select from the list of packages which could be updated or added. The latter value only works on systems with a GUI version of select.list(), and is otherwise equivalent to ask = TRUE.
oldPkgs	if specified as non-NULL, updatePackages() only considers these packages for updating. This may be a character vector of package names or a matrix as returned by oldPackages().

Details

These functions are based on update.packages(). However, rather than looking for locally installed packages they look for the package source and binaries in the miniCRAN repository.

Value

oldPackages() returns a matrix with one row per package and columns for "Package", "LocalVer", "ReposVer" and "Repository". The matrix row names the package names.

updatePackages returns NULL invisibly.

See Also

updatePackages(), pkgAvail().

Other update repo functions: addOldPackage(), addPackage(), checkVersions(), makeRepo()

Examples

```
### `oldPackages` and `updatePackages` require an existing miniCRAN repo

# Specify list of packages to download
revolution <- c(CRAN = getOption("minicran.mran"))
pkgs <- c("foreach")

pdb <- cranJuly2014

## Not run:
```

```
  pdb <- pkgAvail(repos = revolution, type = "source")

## End(Not run)

pkgList <- pkgDep(pkgs, availPkgs = pdb, repos = revolution, type = "source", suggests = FALSE)
pkgList

## Not run:
  # Create temporary folder for miniCRAN
  dir.create(pth <- file.path(tempdir(), "miniCRAN"))

  # create the miniCRAN directory structure but only add bin files
  makeRepo(pkgList, path = pth, repos = revolution, type = "source", download = FALSE)
  makeRepo(pkgList, path = pth, repos = revolution, type = "win.binary", download = TRUE)

  # download old source package version and create repo index
  oldVers <- data.frame(package = c("foreach", "codetools", "iterators"),
                        version = c("1.4.0", "0.2-7", "1.0.5"),
                        stringsAsFactors = FALSE)
 addOldPackage(pkgList, path = pth, repos = revolution, vers = oldVers$version, type = "source")
  # NOTE: older binary versions would need to be build from source

  # Check if updated packages are available
  oldPackages(path = pth, repos = revolution, type = "source") # should need update
  oldPackages(path = pth, repos = revolution, type = "win.binary") # should be current

  # Update available packages
 updatePackages(path = pth, repos = revolution, type = "source", ask = FALSE) # should need update
  updatePackages(path = pth, repos = revolution, type = "win.binary") # should be current

  # Delete temporary folder
  unlink(pth, recursive = TRUE)

## End(Not run)
```

Index

∗ **Internal**
.listFiles, 4
repoBinPath, 23
repoPrefix, 24
twodigitRversion, 25
∗ **create repo functions**
pkgAvail, 19
∗ **datasets**
cranJuly2014, 14
∗ **dependency functions**
basePkgs, 11
makeDepGraph, 15
pkgDep, 20
plot.pkgDepGraph, 22
∗ **github functions**
addPackageListingGithub, 10
∗ **package**
miniCRAN-package, 2
∗ **update repo functions**
addOldPackage, 6
addPackage, 8
checkVersions, 12
makeRepo, 17
updatePackages, 26
.listFiles, 4
_PACKAGE *(*miniCRAN-package*)*, 2

addLocalPackage, 4
addOldPackage, 6, *9*, *12*, *18*, *27*
addPackage, *7*, 8, *12*, *18*, *27*
addPackageListingGithub, 10
available.packages(), *6*, *9*, *11*, *14*, *15*, *17*, *20*, *21*, *26*

basePkgs, 11, *16*, *21*, *23*

checkVersions, *7*, *9*, 12, *18*, *27*
cranJuly2014, 14

download.file(), *26*
download.packages(), *5*, *6*, *9*, *14*, *15*, *17*, *20*, *21*, *24*, *27*

getCranDescription, 14
getRversion(), *5*, *7*, *9*, *12*, *18*, *20*, *21*, *24*, *25*, *27*

igraph::igraph.plotting(), *22*
install.packages(), *12*, *24*

makeDepGraph, *12*, 15, *21*, *23*
makeDepGraph(), *3*, *22*
makeLibrary, 16
makeRepo, *7*, *9*, *12*, 17, *27*
makeRepo(), *3*
miniCRAN *(*miniCRAN-package*)*, 2
minicran *(*miniCRAN-package*)*, 2
miniCRAN-package, 2

oldPackages *(*updatePackages*)*, 26
oldPackages(), *3*

pkgAvail, 19
pkgAvail(), *3*, *11*, *26*, *27*
pkgDep, *12*, *16*, 20, *23*
pkgDep(), *3*, *12*, *16*, *20*
plot.pkgDepGraph, *12*, *16*, *21*, 22
plot.pkgDepGraph(), *3*

R.version, *5–7*, *9*, *12*, *18*, *20*, *21*, *24*, *25*, *27*
repoBinPath, 23
repoPrefix, 24

select.list(), *27*

twodigitRversion, 25

update.packages(), *27*
updatePackages, *7*, *9*, *12*, *18*, 26
updatePackages(), *3*, *27*
updateRepoIndex *(*makeRepo*)*, 17

utils::available.packages(), *3*, *14*, *19*
utils::download.packages(), *18*
utils::install.packages(), *3*, *17*
utils::installed.packages(), *11*

write_PACKAGES(), *5*, *7*, *9*, *18*