Package: packrat (via r-universe)

June 24, 2026

Type Package

Title A Dependency Management System for Projects and their R Package Dependencies

Version 0.9.3.9000

Description Manage the R packages your project depends on in an isolated, portable, and reproducible way.

License GPL-2

URL https://github.com/rstudio/packrat

BugReports https://github.com/rstudio/packrat/issues

Depends R (>= 3.0.0)

Imports tools, utils

Suggests devtools, httr, knitr, mockery, rmarkdown, testthat (>= 3.0.0), webfakes, withr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Repository https://kwb-r.r-universe.dev

Date/Publication 2026-06-08 19:03:54 UTC

RemoteUrl https://github.com/rstudio/packrat

RemoteRef HEAD

RemoteSha 3f07af4bd443116977c28e11a47f590c17da89ca

Contents

`packrat-package` *Packrat: Reproducible dependency management*

Description

Packrat is a tool for managing the R packages your project depends on in an isolated, portable, and reproducible way.

Details

Use packrat to make your R projects more:

- **Isolated**: Installing a new or updated package for one project won't break your other projects, and vice versa. That's because packrat gives each project its own private package library.
- **Portable**: Easily transport your projects from one computer to another, even across different platforms. Packrat makes it easy to install the packages your project depends on.
- **Reproducible**: Packrat records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

Use `init` to create a new packrat project, `snapshot` to record changes to your project's library, and `restore` to recreate your library the way it was the last time you (or anyone!) took a snapshot.

Using these simple functions and sharing packrat's files lets you collaborate in a shared, consistent environment with others as your project grows and changes, and provides an easy way to share your results when you're done.

Anatomy of a packrat project

A packrat project contains a few extra files and directories. The `init` function creates these files for you, if they don't already exist.

`packrat/lib/` Private package library for this project.

`packrat/src/` Source packages of all the dependencies that packrat has been made aware of.

`packrat/packrat.lock` Lists the precise package versions that were used to satisfy dependencies, including dependencies of dependencies. (This file should never be edited by hand!)

`.Rprofile` Directs R to use the private package library (when it is started from the project directory).

Using packrat with version control

Packrat is designed to work hand in hand with Git, Subversion, or any other version control system. Be sure to check in the `.Rprofile`, `packrat.lock` files, and everything under `packrat/src/`. You can tell your VCS to ignore `packrat/lib/` (or feel free to check it in if you don't mind taking up some extra space in your repository).

Author(s)

Posit Software, PBC

See Also

Useful links:

- https://github.com/rstudio/packrat
- Report bugs at https://github.com/rstudio/packrat/issues

Examples

```
## Not run:
# Create a new packrat project from an existing directory of \R code
init()

# Install a package and take a snapshot of the new state of the library
install.packages("TTR")
snapshot()

# Accidentally remove a package and restore to add it back
remove.packages("TTR")
restore()

## End(Not run)
```

bundle *Bundle a Packrat Project*

Description

Bundle a packrat project, for easy sharing.

Usage

```
bundle(
  project = NULL,
  file = NULL,
  include.src = TRUE,
  include.lib = FALSE,
  include.bundles = TRUE,
  include.vcs.history = FALSE,
  overwrite = FALSE,
  omit.cran.src = FALSE,
  ...
)
```

Arguments

project	The project directory. Defaults to the currently activate project. By default, the current project active under packratMode is checked.
file	The path to write the bundle. By default, we write the bundle to packrat/bundles/<project>-<date>.t with <date> as returned by Sys.date().
include.src	Include the packrat sources?
include.lib	Include the packrat private library?
include.bundles	Include other packrat bundle tarballs (as in packrat/bundles/)?
include.vcs.history	Include version control history (ie, .git/ or .svn/ folders)?
overwrite	Boolean; overwrite the file at file if it already exists?
omit.cran.src	Boolean; when TRUE, packages whose sources can be retrieved from CRAN are excluded from the bundle.
...	Optional arguments passed to tar.

Details

The project is bundled as a gzipped tarball (.tar.gz), which can be unbundled either with packrat::unbundle (which restores the project as well), R's own utils::untar, or through most system tar implementations.

The tar binary is selected using the same heuristic as restore.

Value

The path (invisibly) to the bundled project.

`clean` *Remove Packages from the Library*

Description

Remove packages from the given library.

Usage

```
clean(
  packages = NULL,
  project = NULL,
  lib.loc = libDir(project),
  dry.run = FALSE,
  force = FALSE
)
```

Arguments

`packages`	A set of package names to remove from the project. When `NULL`, `unused_packages` is used to find packages unused in the project.
`project`	The project directory. Defaults to current working directory.
`lib.loc`	The library to clean. Defaults to the private package library associated with the project directory.
`dry.run`	Perform a dry run, returning records on which packages would have been moved by the current clean action.
`force`	Force package removal, even if they are still in use within the project?

Examples

```
## Not run:

# Get unused package records
unused_packages()

# Clean all unused packages
clean()

# Clean specific packages
clean("foo")


## End(Not run)
```

disable *Disable the use of Packrat in a Project*

Description

Disable packrat within a project, reverting to the use of standard user package libraries.

Usage

```
disable(project = NULL, restart = TRUE)
```

Arguments

project	The directory in which packrat will be disabled (defaults to the current working directory)
restart	If TRUE, restart the R session after disabling packrat.

Note

Disabling packrat for a project removes the packrat initialization code from the .Rprofile file, resulting in the use of standard user package libraries. Note that the packrat directory is not deleted, but remains unused.

To re-enable the use of packrat for a project you can call the init function.

The restart parameter will only result in a restart of R when the R environment packrat is running within makes available a restart function via getOption("restart").

init *Initialize Packrat on a new or existing* R *project*

Description

Given a project directory, makes a new packrat project in the directory.

Usage

```
init(
  project = ".",
  options = NULL,
  enter = TRUE,
  restart = enter,
  infer.dependencies = TRUE
)
```

Arguments

project	The directory that contains the R project.
options	An R list of options, as specified in packrat-options.
enter	Boolean, enter packrat mode for this project after finishing a init?
restart	If TRUE, restart the R session after init.
infer.dependencies	If TRUE, infer package dependencies by examining the R code.

Details

init works as follows:

1. Application dependencies are computed by examining the R code throughout the project for library and require calls. You can opt out of this behavior by setting infer.dependencies to FALSE.
2. A snapshot is taken of the version of each package currently used by the project as described in snapshot, and each package's sources are downloaded.
3. A private library is created in the directory.
4. The snapshot is applied to the directory as described in restore.

When init is finished, all the packages on which the project depends are installed in a new, private library located inside the project directory.

You must restart your R session in the given project directory after running init in order for the changes to take effect!

When R is started in the directory, it will use the new, private library. Calls to require and library will load packages from the private library (except for 'base' or 'recommended' R packages, which are found in the system library), and functions such as install.packages will modify that private library. You can sync this private library with packrat using snapshot and restore.

Note

The restart parameter will only result in a restart of R when the R environment packrat is running within makes available a restart function via getOption("restart").

See Also

packrat for a description of the files created by init.

Examples

```
## Not run:

## initialize a project using a local repository of packages
packrat::init(options = list(local.repos = "~/projects/R"))


## End(Not run)
```

`install` *Install a local development package.*

Description

Uses `R CMD INSTALL` to install the package. Will also try to install dependencies of the package from CRAN, if they're not already installed.

Usage

```
install(
  pkg = ".",
  reload = TRUE,
  quick = FALSE,
  local = TRUE,
  args = getOption("devtools.install.args"),
  quiet = FALSE,
  dependencies = NA,
  build_vignettes = !quick,
  keep_source = getOption("keep.source.pkgs")
)
```

Arguments

`pkg`	package description, can be path or package name.
`reload`	if `TRUE` (the default), will automatically reload the package after installing.
`quick`	if `TRUE` skips docs, multiple-architectures, demos, and vignettes, to make installation as fast as possible.
`local`	if `FALSE` builds the package first: this ensures that the installation is completely clean, and prevents any binary artefacts (like '`.o`', `.so`) from appearing in your local package directory, but is considerably slower, because every compile has to start from scratch.
`args`	An optional character vector of additional command line arguments to be passed to `R CMD install`. This defaults to the value of the option `"devtools.install.args"`.
`quiet`	if `TRUE` suppresses output from this function.
`dependencies`	`logical` indicating to also install uninstalled packages which this `pkg` depends on/links to/suggests. See argument `dependencies` of `install.packages`.
`build_vignettes`	if `TRUE`, will build vignettes. Normally it is `build` that's responsible for creating vignettes; this argument makes sure vignettes are built even if a build never happens (i.e. because `local = TRUE`.
`keep_source`	If `TRUE` will keep the srcrefs from an installed package. This is useful for debugging (especially inside of RStudio). It defaults to the option `"keep.source.pkgs"`.

Details

By default, installation takes place using the current package directory. If you have compiled code, this means that artefacts of compilation will be created in the `src/` directory. If you want to avoid this, you can use `local = FALSE` to first build a package bundle and then install it from a temporary directory. This is slower, but keeps the source directory pristine.

If the package is loaded, it will be reloaded after installation.

`install_local` *Install a Package from a Local Repository*

Description

This function can be used to install a package from a local 'repository'; i.e., a directory containing package tarballs and sources.

Usage

```
install_local(pkgs, ..., lib = .libPaths()[1], repos = get_opts("local.repos"))
```

Arguments

`pkgs`	A character vector of package names.
`...`	Optional arguments passed to `install`.
`lib`	The library in which the package should be installed.
`repos`	The local repositories to search for the package names specified.

`lockfile-metadata` *Get / Set packrat lockfile metadata*

Description

Get and set metadata in the current packrat-managed project lockfile `packrat.lock`

Usage

```
set_lockfile_metadata(repos = NULL, r_version = NULL, project = NULL)

get_lockfile_metadata(metadata = NULL, simplify = TRUE, project = NULL)
```

Arguments

repos	A named character vector of the form c(<repoName> = "<pathToRepo>").
r_version	A length-one character vector with suitable numeric version string. See package_version.
project	The project directory. When in packrat mode, defaults to the current project; otherwise, defaults to the current working directory.
metadata	The lockfile field name(s) to draw from.
simplify	Boolean; if TRUE the returned metadata will be un-listed.

Details

Project's packrat.lock contains some metadata before packages dependencies informations. The project's lockfile is created and updated programmatically by snapshot. However it could be necessary sometimes to modify manually some of those values. For example, it could be useful to set another repository CRAN url when deploying to a offline environnement.

available metadata

- r_version: R version the project depends on
- repos: Name of repos and their url recorded packages can be retrieve from. Only url is recommended to change if need. Name of repos is used in package records and must be identical

Examples

```
## Not run:
# changes repos url
repos <- old_repos <- get_lockfile_metadata("repos")
repos
repos["CRAN"] <- "https://cran.r-project.org/"
set_lockfile_metadata(repos = repos)
get_lockfile_metadata("repos")
# setting back old state
# set_lockfile_metadata(repos = old_repos)

# changes R version
rver <- old_rver <- get_lockfile_metadata("r_version")
rver
rver <- "3.4.1"
set_lockfile_metadata(r_version = rver)
get_lockfile_metadata("r_version")
# Setting back old state
# set_lockfile_metadata(r_version = old_rver)

## End(Not run)
```

packify *Automatically Enter Packrat Mode on Startup*

Description

Install/augment the `.Rprofile` in a project, so that all R sessions started in this directory enter `packrat` mode, and use the local project library.

Usage

```
packify(project = NULL, quiet = FALSE)
```

Arguments

`project`	The directory in which to install the `.Rprofile` file.
`quiet`	Be chatty?

Details

It is not normally necessary to call `packify` directly; these files are normally installed by `init`. `packify` can be used to restore the files if they are missing (for instance, if they were not added to source control, or were accidentally removed).

You'll need to restart R in the specified directory after running `packify` in order to start using the private package library.

packrat-external *Managing External Libraries*

Description

These functions provide a mechanism for (temporarily) using packages outside of the packrat private library. The packages are searched within the 'default' libraries; that is, the libraries that would be available upon launching a new R session.

Usage

```
with_extlib(packages = NULL, expr, envir = parent.frame())

extlib(packages)

user_lib()

packrat_lib()
```

Arguments

packages	An optional set of package names (as a character vector) to load for the duration of evaluation of expr. Whether packages is provided or NULL (the default), expr is evaluated in an environment where the external library path is in place, not the local (packrat) library path.
expr	An R expression.
envir	An environment in which the expression is evaluated.

Examples

```
## Not run:
with_extlib("lattice", xyplot(1 ~ 1))
with_extlib(expr = packageVersion("lattice"))
# since devtools requires roxygen2 >= 5.0.0 for this step, this
# should fail unless roxygen2 is available in the packrat lib.loc
with_extlib("devtools", load_all("path/to/project"))
# this method will work given roxygen2 is installed in the
# non-packrat lib.loc with devtools
with_extlib(expr = devtools::load_all("path/to/project"))

## End(Not run)
```

---

packrat-mode *Packrat Mode*

Description

Use these functions to switch packrat mode on and off. When within packrat mode, the R session will use the private library generated for the current project.

Usage

```
packrat_mode(
  on = NULL,
  project = NULL,
  auto.snapshot = get_opts("auto.snapshot"),
  clean.search.path = FALSE
)

on(
  project = NULL,
  auto.snapshot = get_opts("auto.snapshot"),
  clean.search.path = TRUE,
  print.banner = TRUE
)

off(project = NULL, print.banner = TRUE)
```

Arguments

on	Turn packrat mode on (TRUE) or off (FALSE). If omitted, packrat mode will be toggled.
project	The directory in which packrat mode is launched – this is where local libraries will be used and updated.
auto.snapshot	Perform automatic, asynchronous snapshots?
clean.search.path	Detach and unload any packages loaded from non-system libraries before entering packrat mode?
print.banner	Print the packrat banner when entering / exiting packrat mode? The packrat banner informs you of the new packrat mode state, as well as the library path in use.

---

packrat-options *Get/set packrat project options*

Description

Get and set options for the current packrat-managed project.

Usage

```
get_opts(options = NULL, simplify = TRUE, project = NULL)

set_opts(..., project = NULL, persist = TRUE)

opts
```

Arguments

options	A character vector of valid option names.
simplify	Boolean; unlist the returned options? Useful for when retrieving a single option.
project	The project directory. When in packrat mode, defaults to the current project; otherwise, defaults to the current working directory.
...	Entries of the form key = value, used for setting packrat project options.
persist	Boolean; persist these options for future sessions?

Valid Options

- auto.snapshot: Perform automatic, asynchronous snapshots when running interactively? (logical; defaults to FALSE)
- use.cache: Install packages into a global cache, which is then shared across projects? The directory to use is read through Sys.getenv("R_PACKRAT_CACHE_DIR"). Windows support is currently experimental. (logical; defaults to FALSE)

- print.banner.on.startup: Print the banner on startup? Can be one of TRUE (always print), FALSE (never print), and 'auto' (do the right thing) (defaults to "auto")
- vcs.ignore.lib: If TRUE, version control configuration is modified to ignore packrat private libraries. (logical; defaults to TRUE)
- vcs.ignore.src: If TRUE, version control configuration is modified to ignore packrat private sources. (logical; defaults to FALSE)
- external.packages: Packages which should be loaded from the user library. This can be useful for very large packages which you don't want duplicated across multiple projects, e.g. BioConductor annotation packages, or for package development scenarios wherein you want to use e.g. devtools and roxygen2 for package development, but do not want your package to depend on these packages. (character; defaults to Sys.getenv("R_PACKRAT_EXTERNAL_PACKAGES"))
- local.repos: Ad-hoc local 'repositories'; i.e., directories containing package sources within sub-directories. (character; empty by default)
- load.external.packages.on.startup: Load any packages specified within external.packages on startup? (logical; defaults to TRUE)
- ignored.packages: Prevent packrat from tracking certain packages. Dependencies of these packages will also not be tracked (unless these packages are encountered as dependencies in a separate context from the ignored package). (character; empty by default)
- ignored.directories: Prevent packrat from looking for dependencies inside certain directories of your workspace. For example, if you have set your "local.repos" to be inside your local workspace so that you can track custom packages as git submodules. Each item should be the relative path to a directory in the workspace, e.g. "data", "lib/gitsubmodule". Note that packrat already ignores any "invisible" files and directories, such as those whose names start with a "." character. (character; empty by default)
- quiet.package.installation: Emit output during package installation? (logical; defaults to TRUE)
- snapshot.recommended.packages: Should 'recommended' packages discovered in the system library be snapshotted? See the Priority field of available.packages() for more information – 'recommended' packages are those normally bundled with CRAN releases of R on OS X and Windows, but new releases are also available on the CRAN server. (logical; defaults to FALSE)
- snapshot.fields: What fields of a package's DESCRIPTION file should be used when discovering dependencies? (character, defaults to c("Imports", "Depends", "LinkingTo"))
- symlink.system.packages: Symlink base R packages into a private packrat/lib-R directory? This is done to further encapsulate the project from user packages that have been installed into the R system library. (boolean, defaults to TRUE)

Examples

```
## Not run:
## use 'devtools' and 'knitr' from the user library
packrat::set_opts(external.packages = c("devtools", "knitr"))

## set local repository
packrat::set_opts(local.repos = c("~/projects/R"))
```

```
## get the set of 'external packages'
packrat::opts$external.packages()

## set the external packages
packrat::opts$external.packages(c("devtools", "knitr"))

## End(Not run)
```

---

packrat-resources *Paths to Packrat Resources*

Description

These functions provide a mechanism for retrieving the paths to Packrat resource directories. Each of these directories can be overridden by setting either an environment variable, or an R option.

Usage

```
project_dir(project = NULL)

src_dir(project = NULL)

lib_dir(project = NULL)

bundles_dir(project = NULL)
```

Arguments

project	The project directory.

Project Directory

`project_dir()` is special – the `R_PACKRAT_PROJECT_DIR` environment variable is set and unset by `on` and `off`, respectively, and generally should not be overridden by the user.

Directory Resolution

The following table shows the order in which resource directories are discovered (from left to right). The first non-empty result is used.

API	Environment Variable	R Option	Default Value
`project_dir()`	`R_PACKRAT_PROJECT_DIR`	`packrat.project.dir`	`getwd()`
`src_dir()`	`R_PACKRAT_SRC_DIR`	`packrat.src.dir`	`"packrat/src"`
`lib_dir()`	`R_PACKRAT_LIB_DIR`	`packrat.lib.dir`	`"packrat/lib"`
`bundles_dir()`	`R_PACKRAT_BUNDLES_DIR`	`packrat.bundles.dir`	`"packrat/bundles"`
(none)	`R_PACKRAT_LIB_R_DIR`	`packrat.lib-r.dir`	`"packrat/lib-R"`
(none)	`R_PACKRAT_LIB_EXT_DIR`	`packrat.lib-ext.dir`	`"packrat/lib-ext"`

repos_create *Create a Local, CRAN-like Repository*

Description

Generate a local CRAN-like repository which can be used to store and distribute R packages.

Usage

```
repos_create(path, name = basename(path), add = TRUE)
```

Arguments

path	Path to a local CRAN-like repository.
name	The name to assign to the repository. Defaults to the directory name in which the reopsitory is created.
add	Add this new repository to the current set of repositories?

repos_upload *Upload a Package to a Local CRAN-like Repository*

Description

Upload a Package to a Local CRAN-like Repository

Usage

```
repos_upload(package, to, ...)
```

Arguments

package	Path to a package tarball. The tarball should be created by R CMD build; alternatively, it can be the path to a folder containing the source code for a package (which will then be built with R CMD build) and then uploaded to the local repository.
to	The name of the CRAN-like repository. It (currently) must be a local (on-disk) CRAN repository.
...	Optional arguments passed to R CMD build.

repository-management *Add a Repository*

Description

Add a repository to the set of currently available repositories. This is effectively an easier-to-use wrapper over interacting with the "repos" option, which is otherwise set with options(repos = ...).

Usage

```
repos_add(..., overwrite = FALSE)

repos_add_local(..., overwrite = FALSE)

repos_set(...)

repos_set_local(...)

repos_remove(names)

repos_list()
```

Arguments

...	Named arguments of the form <repoName> = <pathToRepo>.
overwrite	Boolean; overwrite if a repository with the given name already exists?
names	The names of repositories (as exist in e.g. names(getOption("repos"))).

Details

repos_add_local is used for adding file-based repositories; that is, CRAN repositories that live locally on disk and not on the internet / local network.

restore *Apply the most recent snapshot to the library*

Description

Applies the most recent snapshot to the project's private library.

Usage

```
restore(
  project = NULL,
  overwrite.dirty = FALSE,
  prompt = interactive(),
  dry.run = FALSE,
  restart = !dry.run
)
```

Arguments

project	The project directory. When in packrat mode, if this is NULL, then the directory associated with the current packrat project is used. Otherwise, the project directory specified is used.
overwrite.dirty	A dirty package is one that has been changed since the last snapshot or restore. Packrat will leave these alone by default. If you want to guarantee that restore will put you in the exact state represented by the snapshot being applied, use overwrite.dirty = TRUE.
prompt	TRUE to prompt before performing potentially destructive changes (package removals or downgrades); FALSE to perform these operations without confirmation.
dry.run	If TRUE, compute the changes to your packrat state that would be made if a restore was performed, without actually executing them.
restart	If TRUE, restart the R session after restoring.

Details

restore works by adding, removing, and changing packages so that the set of installed packages and their versions matches the snapshot exactly.

There are three common use cases for restore:

- **Hydrate**: Use restore after copying a project to a new machine to populate the library on that machine.
- **Sync**: Use restore to apply library changes made by a collaborator to your own library. (In general, you want to run restore whenever you pick up a change to packrat.lock)
- **Rollback**: Use restore to undo accidental changes made to the library since the last snapshot.

restore cannot make changes to packages that are currently loaded. If changes are necessary to currently loaded packages, you will need to restart R to apply the changes (restore will let you know when this is necessary). It is recommended that you do this as soon as possible, because any library changes made between running restore and restarting R will be lost.

Note

restore can be destructive; it will remove packages that were not in the snapshot, and it will replace newer packages with older versions if that's what the snapshot indicates. restore will warn you

before attempting to remove or downgrade a package (if `prompt` is `TRUE`), but will always perform upgrades and new installations without prompting.

`restore` works only on the private package library created by packrat; if you have other libraries on your path, they will be unaffected.

The `restart` parameter will only result in a restart of R when the R environment packrat is running within makes available a restart function via `getOption("restart")`.

To install packages hosted in private repositories on GitHub, GitLab, and Bitbucket, you must either set the option `packrat.authenticated.downloads.use.renv` to `TRUE` and ensure that `curl` is available on your system, or ensure that the `httr` package is available in your R library.

In addition, you must make credentials for your provider available in the appropriate environment variable(s): `GITHUB_PAT`, `GITLAB_PAT`, and/or `BITBUCKET_USERNAME` and `BITBUCKET_PASSWORD`. These environment variables are hidden from package installation subprocesses.

Packrat does not support installation from enterprise instances of GitHub, GitLab, or Bitbucket.

Packrat selects a `tar` binary with the following heuristic: If a `TAR` environment variable exists, Packrat will use that. Otherwise, it will either look for a `tar` binary on the `PATH` on Unix, or look for the system `tar` on Windows. If no binary is found in those locations, it will use R's internal `tar` implementation, which may cause errors with long filenames.

See Also

`snapshot`, the command that creates the snapshots applied with `restore`.

`status` to view the differences between the most recent snapshot and the library.

---

search_path *Get Packages on the Search Path*

Description

Retrieve the packages on the search path, as well as the associated library location.

Usage

```
search_path()
```

snapshot *Capture and store the packages and versions in use*

Description

Finds the packages in use in the project, and stores a list of those packages, their sources, and their current versions in packrat.

Usage

```
snapshot(
  project = NULL,
  available = NULL,
  lib.loc = libDir(project),
  ignore.stale = FALSE,
  dry.run = FALSE,
  prompt = interactive(),
  snapshot.sources = TRUE,
  infer.dependencies = TRUE
)
```

Arguments

project	The project directory. Defaults to current working directory.
available	A database of available packages.
lib.loc	The library to snapshot. Defaults to the private library associated with the given directory.
ignore.stale	Stale packages are packages that are different from the last snapshot, but were installed by packrat. Typically, packages become stale when a new snapshot is available, but you haven't applied it yet with restore. By default, packrat will prevent you from taking a snapshot when you have stale packages to prevent you from losing changes from the unapplied snapshot. If your intent is to overwrite the last snapshot without applying it, use ignore.stale = TRUE to skip this check.
dry.run	Computes the changes to your packrat state that would be made if a snapshot were performed, and prints them to the console.
prompt	TRUE to prompt before performing snapshotting package changes that might be unintended; FALSE to perform these operations without confirmation. Potentially unintended changes include snapshotting packages at an older version than the last snapshot, or missing despite being present in the last snapshot.
snapshot.sources	Boolean; should package sources be downloaded during snapshot?
infer.dependencies	If TRUE, infer package dependencies by examining R code used within the project. This included the R code contained within .R files, as well as other multi-mode documents (e.g. .Rmd).

Note

snapshot modifies the project's packrat.lock file, and the sources stored in the project's packrat/src directory. If you are working with a version control system, your collaborators can sync the changes to these files and then use restore to apply your snapshot.

See Also

restore to apply a snapshot. status to view the differences between the most recent snapshot and the library.

Examples

```
## Not run:
# Take a snapshot of the current project
snapshot()

# See what changes would be included in a snapshot
snapshot(dry.run = TRUE)


## End(Not run)
```

---

status *Show differences between the last snapshot and the library*

Description

Shows the differences between the project's packrat dependencies, its private package library, and its R scripts.

Usage

```
status(project = NULL, lib.loc = libDir(project), quiet = FALSE)
```

Arguments

project	The directory that contains the R project.
lib.loc	The library to examine. Defaults to the private library associated with the project directory.
quiet	Print detailed information about the packrat status to the console?

Details

These differences are created when you use the normal R package management commands like install.packages, update.packages, and remove.packages. To bring these differences into packrat, you can use snapshot.

Differences can also arise if one of your collaborators adds or removes packages from the packrat dependencies. In this case, you simply need to tell packrat to update your private package library using restore.

Value

Either NULL if a packrat project has not yet been initialized, or a (invisibly) a data.frame with components:

package	The package name,
packrat.version	The package version used in the last snapshot,
packrat.source	The location from which the package was obtained,
library.version	The package version available in the local library,
currently.used	Whether the package is used in any of the R code in the current project.

---

unbundle *Unbundle a Packrat Project*

Description

Unbundle a previously bundled project.

Usage

```
unbundle(bundle, where, ..., restore = TRUE)
```

Arguments

bundle	Path to the bundled file.
where	The directory where we will unbundle the project.
...	Optional arguments passed to tar.
restore	Boolean; should we restore the library after unbundle-ing the project?

---

unused_packages *Find Unused Packages in a Project*

Description

Unused packages are those still contained within your project library, but are unused in your project.

Usage

```
unused_packages(project = NULL, lib.loc = libDir(project))
```

Arguments

project	The project directory.
lib.loc	The library to check.

Index