Package 'groundhog'

May 8, 2026

Title Version-Control for CRAN, GitHub, and GitLab Packages

Version 3.4.0

Description Make R scripts reproducible, by ensuring that every time a given script is run, the same version of the used packages are loaded (instead of whichever version the user running the script happens to have installed). This is achieved by using the command groundhog.library() instead of the base command library(), and including a date in the call. The date is used to call on the same version of the package every time (the most recent version available at that date). Load packages from CRAN, GitHub, or Gitlab.

URL https://groundhogr.com/, https://github.com/CredibilityLab/groundhog

BugReports https://github.com/CredibilityLab/groundhog/issues

Depends utils

Imports methods

Suggests git2r, remotes

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Uri Simonsohn [aut, cre] (ORCID: <https://orcid.org/0000-0002-8601-7211>), Hugo Gruson [ctb, aut] (ORCID: <https://orcid.org/0000-0002-4094-1476>)

Maintainer Uri Simonsohn <urisohn@gmail.com>

Repository CRAN

Date/Publication 2026-01-10 12:40:02 UTC

Contents

`cross.toc` *Show toc table with multiple packages*

Description

Show toc table with multiple packages

Usage

```
cross.toc(pkgs, date1 = "1970-1-1", date2 = Sys.Date())
```

Arguments

`pkgs`	character vector containing the package names.
`date1`, `date2`	date range to consider (in the format "%Y-%m-%d").

Value

A `data.frame` with 3 columns:

Version The package version number

Published The date at which the specific version was published

Package The package name

See Also

`toc()` for the same function for a single package.

Examples

```
## Not run:
cross.toc(c("magrittr", "R"))
cross.toc(c("magrittr", "rlang"), date1 = "2012-02-01", date2 = "2020-02-01")

## End(Not run)
```

`get.groundhog.folder` *Get current local path to groundhog folder*

Description

Get current local path to groundhog folder

Usage

```
get.groundhog.folder()
```

Value

the path to the groundhog folder, the meta-library where `groundhog.library()` downloads and stores packages that can be loaded

Note

you can change the location of this folder with the command `set.groundhog.folder("path")`.

See Also

`set.groundhog.folder()`

Examples

```
## Not run:
get.groundhog.folder()

## End(Not run)
```

`get.snowball` *Generates dataframe with all dependencies needed to install a package, in the order they will be loaded*

Description

Generates dataframe with all dependencies needed to install a package, in the order they will be loaded

Usage

```
get.snowball(pkg, date, include.suggests = FALSE, force.install = FALSE)
```

Arguments

`pkg`	character string, name of target package to load (and install if needed),
`date`	character string (yyyy-mm-dd), or date value, with the date which determines the version of the package, and all dependencies, to be loaded (and installed if needed).
`include.suggests`	logical, defaults to `FALSE`. When set to `TRUE`, includes dependencies classified in the DESCRIPTION file as `suggested`.
`force.install`	logical, defaults to `FALSE`. When set to `TRUE`, the column `installed` in the generated snowball is set FALSE for all packages, causing them to be installed even if already installed.

Value

a dataframe with all packages that need to be installed, their version , whether they are installed, where to obtain them if not locally available (CRAN vs MRAN), which date to use for MRAN, installation time from source (in seconds), and local path for storage

Examples

```
## Not run:
get.snowball("rio", "2020-07-12")

## End(Not run)
```

`groundhog.library` *Install & load CRAN, GitHub, and GitLab packages as current on given date*

Description

Load requested package(s) as current on a requested date. If the needed version of a package, or its dependencies, is not already installed, groundhog automatically installs it. `groundhog.library()` thus substitutes both `library()` and `install.packages()`. There is no change in setup or configuration parameters needed to start using groundhog; simply edit your script going between `library()` and `groundhog.library()` as needed. Groundhog often installs/uninstalls packages in the default personal library. These changes can be reversed in a few seconds, with `restore.library()`

Usage

```
groundhog.library(
  pkg,
  date,
  quiet.install = TRUE,
  include.suggests = FALSE,
  ignore.deps = c(),
  force.source = FALSE,
  force.install = FALSE,
  force.source.main = FALSE,
  force.install.main = FALSE,
  tolerate.R.version = "",
  cores = -1
)
```

Arguments

`pkg`	character string or vector with name of package(s) to load/install.
`date`	character string (yyyy-mm-dd), or date value, with the date which determines the version of the package, and all dependencies, to be loaded (and installed if needed). The most recent date accepted is 2 days prior to when the code is executed.
`quiet.install`	logical, defaults to `TRUE`. When set to `FALSE`, displays output generated by `install.packages()` when installing from source
`include.suggests`	logical, defaults to `FALSE`. When set to `TRUE`, loads dependencies classified in the DESCRIPTION file as `suggested`.
`ignore.deps`	an optional character vector containing dependencies which are already loaded in the R session, and create a conflict with a needed dependency for the package being loaded (mismatch of version), but which should be ignored and groundhog.library() should proceed tolerating the conflict.

`force.source` logical (defaults to `FALSE`). When set to `TRUE`, if the requested package, or its dependencies, needs to be installed, they will be installed from source (much slower than from binaries).

`force.install` logical (defaults to `FALSE`). When set to `TRUE`, will re-install the requested packages and their dependencies even if they are already installed.

`force.source.main` logical (defaults to `FALSE`). When set to `TRUE`, if the requested package needs to be installed it will be installed from source (but dependencies are installed from binaries if needed and available).

`force.install.main` logical (defaults to `FALSE`). When set to `TRUE`, will re-install the requested packages even if they are already installed (but dependencies will not be re-installed).

`tolerate.R.version` optional character string containing an R version which `groundhog.library()` will not throw an error for using, even if the date entered corresponds to a more recent major R release.

`cores` Integer. The maximum number of cores to use during parallel installation of source packages. The default, -1, uses the total number of cores available minus 1. Setting `core=1` leads to installing source packages, and also to downloading binaries, sequentially. When installation fails, you may want to try cores=1

Details

For more information about groundhog check out groundhogr.com

Examples

```
## Not run:
groundhog.library("magrittr", "2022-04-15")

pkgs <- c('pwr','metafor')
groundhog.library(pkgs, "2022-04-15")

# When running an existing  script that relied on `library()` to load packages,
# you can wrap the library calls in double-quotes, loading the packages with
# groundhog:

  groundhog.library(
       "
        library('pwr')
        library('metafor')
        library('tidyr')
        library('rio')
        library('this.path')
       "
       ,'2022-04-01')

#Allow using R 3.6.3 despite entering a date that corresponds to R >=4.0.0
  groundhog.library('rio', '2022-04-11', tolerate.R.version='3.6.3')
```

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

`meta.groundhog` *Load a specific version of groundhog, as available on a given date*

Description

Load a specific version of groundhog, as available on a given date

Usage

```
meta.groundhog(date)
```

Arguments

`date`	character string (yyyy-mm-dd), or date value, with the date which determines the version of groundhog to load

Examples

```
## Not run:
#Load groundhog as available on 2021-03-12 (v1.3.2)
meta.groundhog("2021-03-12")

## End(Not run)
```

`restore.library` *Restore default library of packages, undoing all changes made by groundhog*

Description

In a few seconds reverse changes made by groundhog to the default personal library (where R packages are usually installed into with `installed.packages()`). If you are just trying groundhog out for the first time, or you generally rely on base R's `library()` and want to use `groundhog.library()` sporadically then you may want to run `restore.library()` when you are done with your one-time use of groundhog. This will undo any and all changes made by `groundhog` to that library. In most circumstances, restoring a library takes a few seconds, even if the library has had 100s of package modifications.

Usage

```
restore.library(days = 0)
```

Arguments

days	an optional numeric argument used to choose among alternative restore points. When days is set, groundhog restores the personal library to the most recent restore point that is at least days days old. If dais is not set, groundhog restores to the most recent restore point overall. For example, if there are two restore points: one from today, and one from 7 days ago, running restore.library() would restore to the former, setting days=3 would restore to the latter, and setting days=8 would result in an error. days = -1 restores to the oldest restore point available.

Details

When groundhog installs a package, it installs it into groundhog's library (location of that library is obtainable with get.groundhog.folder()). Groundhog then immediately moves the installed package(s) (and their dependencies) to the default personal library (location of that library obtainable with: .libPaths()[1]). Altering the packages in the local folder is necessary for groundhog to work properly for two main reasons. First, R Studio often loads packages from that library before users run the code in a script, creating version conflicts that cannot be avoided when attempting to load other versions of those packages with groundhog. Second, R scripts often run processes in independent R sessions, for example when doing parallel processing. Those background processes will also look for packages in the default personal folder. Because the personal library can only hold one version of a given package, before moving new packages in, groundhog moves any existing other versions of those packages out, to another directory (a local archive). Those files are not deleted, just moved, making it easy and fast to recover. When the first change in the personal folder is made on a given calendar date, groundhog makes a list of all packages available in the personal folder before such change (saving a copy of the results from installed.packages(.libPaths()[1])), this saved file is referred to as a a 'restore point'. With restore.library() groundhog looks up a restore point, obtain the set of packages that used to be installed, and removes any packages *installed by groundhog* which are in the personal library but were not in that restore point; similarly, it moves back to the local library any packages *removed by groundhog* that were in the restore point but are not currently there. This process take a few seconds even for 100+ packages. Note that there is only one restore point per calendar date, so one effectively restores the personal library to how it was before *any* changes were made to it that day with groundhog. Restore points are saved permanently and can be restored at any point. The set of restore points available is stored in the hidden dataframe .available.restore.points. To choose among them use the days argument in restore.library. The default is to restore based on the most recent restore point, so if a user installs groundhog, tests it, and wants to undo all changes made by groundhog, the default behavior will achieve this goal. Note: restoring can take a few minutes if the groundhog folder is on a different drive from the default personal R library (e.g., two different hard drives) or if it is in Dropbox.

Examples

```
## Not run:
restore.library()
restore.library(7)
restore.library(-1)

## End(Not run)
```

`set.groundhog.folder` *Set groundhog folder location*

Description

Set groundhog folder location

Usage

```
set.groundhog.folder(path)
```

Arguments

`path`	Character. The path to the groundhog folder containing the library where packages are downloaded and installed. For best performance and reliability, it is recommended that the groundhog folder be in the same volume/drive as the folder used as the default R library, and that it not be in a Dropbox (or similar) folders. Groundhog will be slower and occasionally produce errors when the groundhog folder is in Dropbox or a different drive.

Value

(invisibly) `TRUE` upon success.

See Also

`get.groundhog.folder()`

Examples

```
## Not run:
set.groundhog.folder("c:/groundhog_folder")

## End(Not run)
```

show_startup *Show suppressed startup messages*

Description

Displays startup messages that were suppressed during the most recent `groundhog.library()` call. These messages are formatted with proper newlines and package version information.

Usage

```
show_startup()
```

Value

Invisibly returns the startup messages string, or NULL if no messages were suppressed.

Examples

```
## Not run:
groundhog.library("some_package", "2024-01-01")
show_startup()  # View any suppressed startup messages

## End(Not run)
```

toc *Show CRAN publication dates for all versions of a given package*

Description

Show CRAN publication dates for all versions of a given package

Usage

```
toc(pkg, dependencies = FALSE)
```

Arguments

pkg	(required) package name
dependencies	logical (defaults to FALSE). Should the output contain package dependencies (Imports, Depends and Suggests) for pkg.

Value

a data.frame where each row corresponds to one version of pkg, a date column contains the publication date, and when dependencies=TRUE, columns show package dependencies over time as well.

Examples

```
## Not run:
toc("R")
toc("magrittr")
toc("rio",dependencies = TRUE)

## End(Not run)
```

`try.renaming.method.again` *Attempt faster method of copying packages across libraries in the future*

Description

Groundhog often moves packages between the groundhog library and the default personal library. This is done by renaming the package folders ('renaming' the parent directory of a folder, effectively moves it). This process is nearly instantaneous even for 100+ packages. The renaming method, however, is sometimes unavailable for some configurations (e.g., when the groundhog and personal folders are on different drives/volumes, say external vs internal hard drives). When groundhog fails to move a package by renaming it, it will produce an error, and will also make a note to permanently switch to the slower method of moving packages by first coping them, and then deleting the original, which takes up to a few seconds per package, and is thus much slower than renaming. If you believe the error was circumstantial and want to give renaming files another chance, you may run `try.renaming.method.again()`. Future groundhog.library() calls will again attempt it. If it fails again you will just get a new error message and groundhog will again switch methods. It is safe to err on the side of trying again, so unless you know you are using multiple physical drives, you probably should try again, and investigate a possible alternative source of the problem. To debug, you may choose `cores=1` to force sequential installation, `force.install=TRUE` to reinstall possibly poorly installed dependencies, and as always, inspecting the console log as packages get installed.

Usage

```
try.renaming.method.again(quiet = FALSE)
```

Arguments

`quiet`	logical, defaults to `FALSE`. When set to `TRUE` it does not display confirmation message.

Index