**Non-distributable early draft**

**Version: 2020 02 28**

**{groundhogR}**

**1. Front-end overview**

The function *groundhog.library()* replaces the functions: *library()* and *install.packages()*

In contrast to library(), with groundhog.library()

* A date must be included within the groundhog.library() command
* The version of the package available on that day, is automatically installed if needed
* All dependencies for that package are installed as available that day.

GroundhogR allows having multiple (possibly all) versions of a package available, and loading whichever one is wanted. For an older project you work with an older version of a package to ensure reproducibility, while for a new project you work with a newer version.

For example, to work with the package 'rio' as it was available on 2017-09-15

instead of library('rio')🡪

groundhog.library('rio', '2017-09-15')

Even on the same script (.r file) you can switch which version you use, like this:

groundhog.library('rio', '2019-01-27')

***2 Recommended uses***

***USE 1: Writing new R Scripts.***

When working on new scripts, unless you have a reason not to, choose a date for the whole script and stick to it. Maybe the date when the project was started. It makes sense to set the date as a variable you can call though your script. You can give that variable any name, perhaps 'groundhog.day' makes sense.

So the top of your reproducible R scripts will look something like this:

#### Written in R Version 3.6.1 - ####

library('groundhogR')

groundhog.date="2019-04-21"

groundhog.library('rio', date= groundhog.day)

groundhog.library('pwr', date= groundhog.day)

<your code>

***USE 2. Revive existing scripts.***

The whole reason for {groundhogR} to exist is that packages change over time in ways that make old scripts not work. A likely use of {groundhogR} is thus to revive old scripts that no longer work.

To recover a non-working script replace its library() statements, with groundhog.library() ones, trying older dates until the script works again.

You can start setting the date to the date when the script you are reviving was last saved.

If that does not work, roll back the clock further (and possibly separately for each package in the script).

This process is facilitated by knowing when a package had a version change.

Use groundhogR::toc() to get the table of contents, like this:

groundhogR::toc('rio')

groundhogR::toc('pwr')

To get a single set with all relevant changes worth trying:

groundhogR cross.toc(c('rio','pwr','R'))

*2.4 Use 3. Document changes in packages*

To explain an error in a paper, or document the introduction of a new feature, etc., it may be useful to run within the same script the same code for two versions of the same package.

For example, one may run some commands with one version of a package and later with a different version (useful for debugging and explaining errors caused by package changes).

If an older version of a package is not compatible with a newer version of R, the needed version of R will be indicated. The user can install it and run the same script on it.

Note that R Studio can manage multiple versions of R and users select the desired one when starting R Studio.

<pending>: want to create the command changeR() which will switch the version of R being used

**3 Back end**

Two key design features that enable {groundhogR} to maintain multiple versions of the same package within the same R installation, and calling specific versions on the spot:

(1) Richer directory structure for the library of packages, and

(2) Dynamically generated search path of packages modified with each groundhog.library() call.

Specifically:

1. **Directory structure.**

With base R, there is a single library folder with package names (without indicating version) as subfolders. Those subfolders contain only the latest installed version of the corresponding package, and the files are deleted and replaced when a package gets updated.[[1]](#footnote-1)

{groundhogR}, instead, gives a different subfolder to each package version. When a new version is installed, it is stored in a new subfolder, keeping all existing ones unchanged.

So for example, in base R, the package rio 0.5.1 is stored here:

C:\Users\mike\Documents\R\win-library\3.6\Rio

While in {groundhogR} it is stored [[2]](#footnote-2)

C:\Users\mike\Documents\groundhogR\R-3.6.2\rio\_0.5.1

So while when with base R {rio} is updated, the same folder now has different contents

C:\Users\mike\Documents\R\win-library\3.6\Rio

With {groundhogR} a new folder is added, keeping both versions available.

C:\Users\mike\Documents\groundhogR\R-3.6.2\rio\_0.5.2

C:\Users\mike\Documents\groundhogR\R-3.6.2\rio\_0.5.1

1. **Dynamic search path**

The library() command in R uses the .libpaths() path to search for installed packages. That path is stable for all library() calls (but can be modified by users).

groundhog.library(), in contrast, uses changes the searchpath for every call, so as to find the right version of the package for the date indicated, and R Version being used.

For instance, groundhog.library('rio','2017-10-11') will find the version of rio for that, date, and all the dependencies for rio for that date, and add the path to each specific needed version to the search path, so that R uses the corresponding version

**Comparison to {checkpoint}**

The 'checkpoint' package is offered by Microsoft to enable reproducible R Code relying on Microsoft's MRAN archive (with daily copies of every file on CRAN, starting in late 2014)[[3]](#footnote-3)

*groundhogR* is thus a substitute for *checkpoint*.

This section discusses feature of *checkpoint* that were deemed undesirable and which are absent from groundhogR

1. *Lack of feedback on errors to debug*

When checkpoint fails to install a package it does not provide feedback on how to correct the problem. For example, the package igraph\_0.7 (with over 500 reverse dependencies), will not install in R version 3.5.0 or higher. To run a script that depends on it, one must use an older versions of R which will allow igraph 0.7 to be installed.

But, the only feedback checkpoint provides a user that attempts to install igraph 0.7 in a newer version of R is that the package failed to load (and therefore so would the 500+ packages that depend on it, or those that depend on those packages, etc.).

{groundhogR} tells users which R version is needed to run igrpah 0.7 (or any package that fails to install), and gives instructions for installing and running older R.

1. *Multiple identical installations*

Checkpoint starts an empty package library for every project, leading to new installations of every package needed for every project. If the same package is used for 20 scripts, it is installed 20 times by Checkpoint, and 20 copies are permanently stored.

groundhogR uses a single comprehensive library that has no duplicate package installs.

1. *Script flow*

For checkpoint to run, scripts must be kept in a project folder, and be already saved there.

So, e.g., if a user starts a new script without saving it in an existing project, checkpoint will not work.

When the date for checkpoint is changed, all scripts in that project will reinstall and all packages for that new date (even if the new date leads to the same package being downloaded and installed again). From that point onwards users can use install.packages() and they will use the previously chosen date, but this is not self-evident from the command itself, and if a line of code is skipped, or a file was not saved, a different package may be installed, in ways that are not traceable. Mistakes seem likely.

1. *No packages before 2014*

Checkpoint relies exclusively on MRAN, which started in late 2014.

No earlier packages can be installed to render old scripts reproducible.

{groundhogR} uses the CRAN archive of source files to access any CRAN package ever available.

1. *Difficult to debug non-working scripts.*

If an existing script does work due to backwards incompatibility of a required package, and the date when the packages used in that script is unknown (a likely scenario), one can try to revive the script by trying alternative install dates for each of the packages, but this is not possible with checkpoint, as all packages share the same date. In addition, changing the date to debug one package will lead to reinstalling all packages in any script in that project. Thus every date attempted coud lead to several minutes of installation time before anything can be attempted.

1. It is possible to have multiple libraries with base R, and many users do without awareness have more than one attached in .libpaths(). In practice, users tend to install all packages in the same library. More importantly, the name of subfolders in base R is just the name of the package, while in {groundhogR} it is the name and version combined. package\_version. [↑](#footnote-ref-1)
2. Note then, that there is a parent directory with the R version for which the package was installed. So there could also be a C:\Users\mike\Documents\groundhogR\**R-3.6.3**\rio\_0.5.1 and a C:\Users\mike\Documents\groundhogR\**R-3.6.1**\rio\_0.5.1

   But, when deciding whether to install a new copy of rio\_0.5.1, groundhogR checks if another identical copy already exists in an R version directory with the same minor (e.g., R-3.6.x) and if it does, it loads it instead of re-installin it. Unless, the force.course=TRUE option is checked, in which a full R version match is enforced, and if 3.6.1 has it, but using 3.6.2, it would be installed anyway. [↑](#footnote-ref-2)
3. I have learned that there are several individual days missing from the MRAN archive, it is not literally *every* day. [↑](#footnote-ref-3)