Workflows for Reproducible Research with R & Git

Dependency Management

Johannes Breuer, Bernd Weiss, & Arnim Bleier
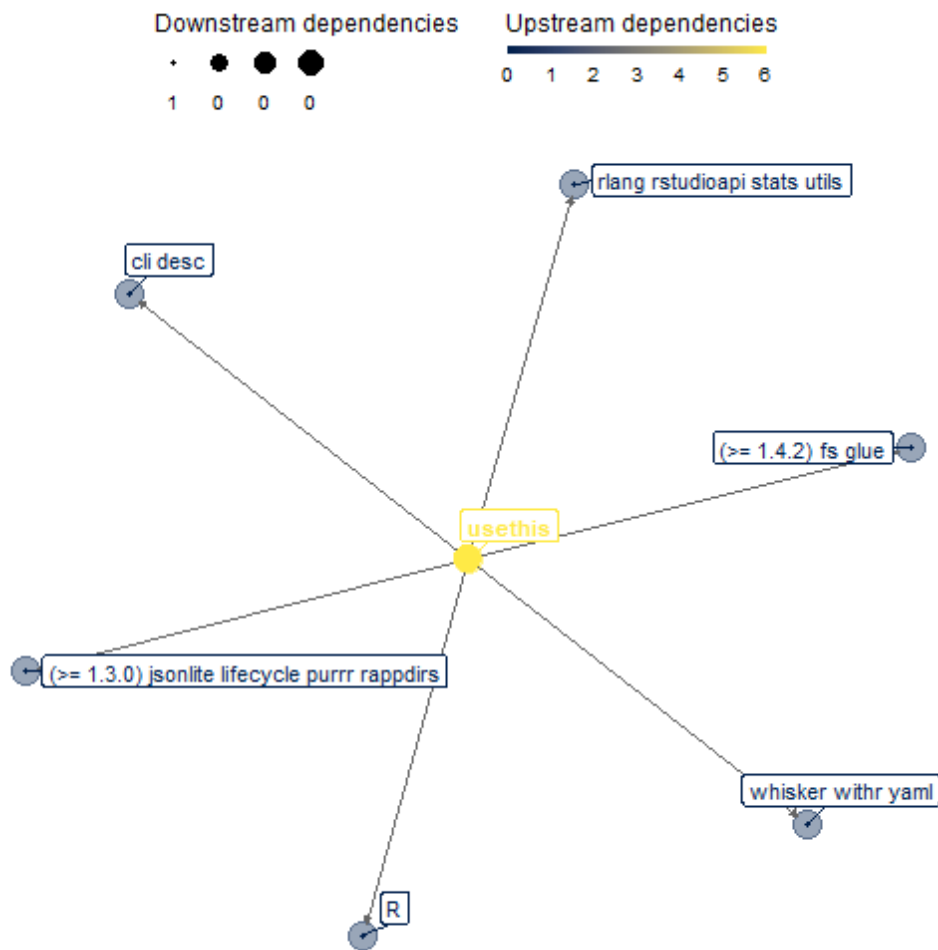
2023-11-17

Dependencies in R

Most R packages depend on other R packages.

All R packages depend on R.

Both R and R packages have versions.

Different versions of R packages may depend on different versions of R and different versions of other packages.

Dependencies in R

Dependencies in $\mathbb{R}$

usethis: Automate Package and Project Setup

Automate package and project setup tasks that are otherwise performed manually. This includes setting up unit testing, test coverage, continuous integration, Git, 'GitHub', licenses, 'Rcpp', 'RStudio' projects, and more.

Version:	2.2.2
Depends:	R (≥ 3.6)
Imports:	cli (≥ 3.0.1), clipr (≥ 0.3.0), crayon, curl (≥ 2.7), desc (≥ 1.4.2), fs (≥ 1.3.0), gert (≥ 1.4.1), gh (≥ 1.2.1), glue (≥ 1.3.0), jsonlite, lifecycle (≥ 1.0.0), purrr, rappdirs, rlang (≥ 1.1.0), rprojroot (≥ 1.2), rstudioapi, stats, utils, whisker, withr (≥ 2.3.0), yaml
Suggests:	covr, knitr, magick, pkgload, rmarkdown, roxygen2 (≥ 7.1.2), spelling (≥ 1.2), styler (≥ 1.2.0), testthat (≥ 3.1.8)
Published:	2023-07-06
Author:	Hadley Wickham 🅿 [aut], Jennifer Bryan 🅿 [aut, cre], Malcolm Barrett 🅿 [aut], Andy Teucher 🅿 [aut], Posit Software, PBC [cph, fnd]
Maintainer:	Jennifer Bryan <jenny at posit.co>
BugReports:	https://github.com/r-lib/usethis/issues
License:	MIT + file LICENSE
URL:	https://usethis.r-lib.org, https://github.com/r-lib/usethis
NeedsCompilation:	no
Language:	en-US
Materials:	README NEWS
In views:	ReproducibleResearch
CRAN checks:	usethis results

Documentation:

Reference manual: usethis.pdf

Source: https://cran.r-project.org/web/packages/usethis/index.html

"It's ~~turtles~~ software all the way down"

🐢

... or is it???

Digging into your tool stack

Your full `R` setup consists of:

1. Specific versions of `R` packages[1]
2. A specific version of `R`[2]
3. A specific version of your operating system
4. Specific hardware

[1] You can have different libraries with different (versions of) `R` packages.

[2] You can also have different versions of `R` installed on your machine.

Further dependencies
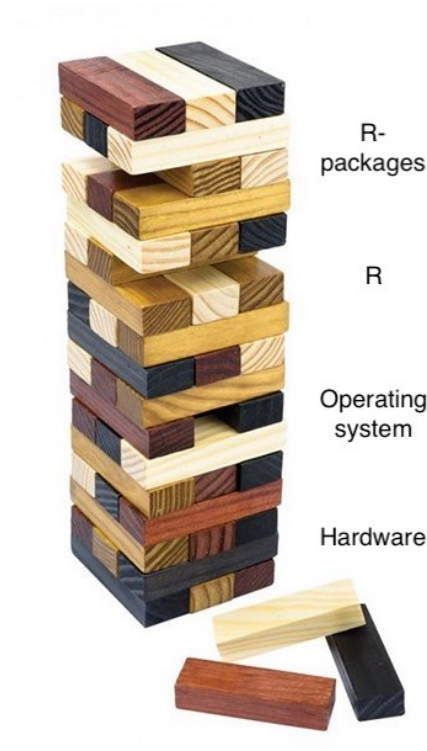
Note: We could also add system libraries between R and the OS (which are especially relevant in the Linux/Unix world).

The danger of dependencies
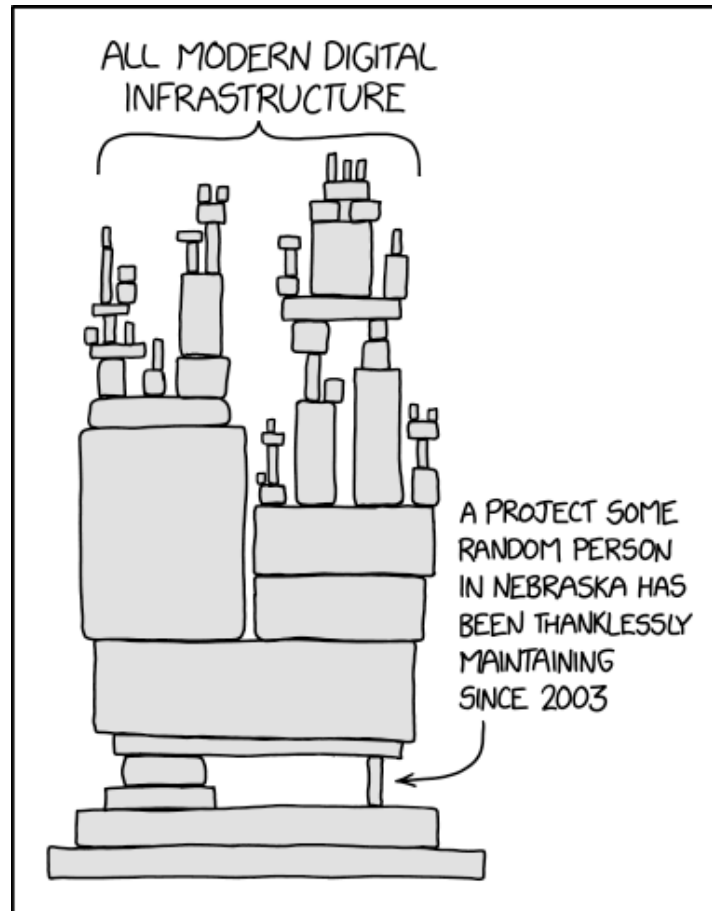
What to "ship" 🚢 ?

- `R` code (+ underlying data)

  - this should include information about the packages used

- information about the version of `R` and the used packages

- your whole computational environment (focus of the next session)

- overall goal: preventing what is known as "code rot" & "works-on-my-machine errors" (WOMME)

Dependency management solutions

As with almost everything in the `R` ecosystem, there are multiple solutions for dependency management:

- a manual approach
- ~~`checkpoint`~~[1]
- **groundhog**
- `renv`[2]
- `rang`[3]

[1] Not an option anymore as it relied on the *CRAN Time Machine snapshots* from the *Microsoft R Application Network* (MRAN) which was retired in July 2022.

[2] `renv` is the successor of `packrat` (which is not maintained anymore).

[3] Developed by members of the team Transparent Social Analytics at GESIS.

Manual approach to dependency management

There is an easy-to-use manual solution for providing information about the packages and R version used in your project:

```
sessionInfo()
```

```
## R version 4.3.2 Patched (2023-10-31 r85451 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=German_Germany.utf8  LC_CTYPE=German_Germany.utf8    LC_MONETARY=German_Germany.utf8 LC_NUM
## [5] LC_TIME=German_Germany.utf8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] groundhog_3.1.2 depgraph_0.1.0  emo_0.0.0.9000  lubridate_1.9.3 forcats_1.0.0   stringr_1.5.0   dplyr
## [10] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.4   tidyverse_2.0.0 knitr_1.45
##
## loaded via a namespace (and not attached):
##  [1] remotes 2.4.2.1            rlang 1.1.2           magrittr 2.0.3           compiler 4.3.2           r
```

groundhog

`groundhog` is a lightweight package that allows you to increase the reproducibility of your `R` scripts. It does so by installing and loading "packages & their dependencies as available on chosen date on CRAN".

Using `groundhog`

All you need to do to use `groundhog` is specifying the packages you want to use in your script and a date.

```r
install.packages("groundhog")
library(groundhog)

pkgs <- c("tidyverse", "janitor", "sjPlot")

groundhog.library(pkgs, date = "2023-11-71")
```

How `groundhog` works

From the package website: "groundhog relies on a database that contains virtually all package versions ever uploaded to CRAN, the date when they were published, and all dependencies."

`groundhog` also used to rely on MRAN. Since that has been retired, however, it now uses its own package repository: GRAN: Groundhog R Archive Neighbor.

How `groundhog` works

`groundhog` uses its own package library to install the packages you specified. From the documentation of the `restore.library()` function: "When groundhog installs a package, it installs it into groundhog's library."

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

```
## [1] "e:\\home/R_groundhog/groundhog_library/"
```

"Groundhog then immediately moves the installed package(s) (and their dependencies) to the default personal library."

```
.libPaths()[1]
```

```
## [1] "D:/Programme/R/library"
```

Note: You can find a lot more technical information on the `groundhog` website and within the help files for the `groundhog` functions.

Reversing changes made by `groundhog`

You can reverse the changes made by `groundhog` to your default personal `R` package library:

```
restore.library()
```

Pros and cons of `groundhog`

Pros:

- can be easily used to make existing `R` scripts more reproducible
- does not require a "project-based workflow"[1]
- does not require any specific knowledge on the reproducer's side

Cons:

- works with package snapshots from specific dates, not with specific package versions
    - in reality, our installed packages are very often not up-to-date
- installs specific package versions, but not specific versions of `R`

[1] While you as someone who highly values reproducibility, of course, do use a project-based workflow, the people who want to reproduce your analysis might not 😉

Choosing a date

The recommendation by the `groundhog` package authors for choosing a date is that "a good default is the first day of the month when starting your project". Once you have added `groundhog.library()` to your script, re-run it to make sure it produces the expected results.

You can also update all of the packages you use and then specify the current date as the date for `groundhog.library()`.

groundhog and R versions

The `groundhog` database also includes information on base R releases. As `groundhog` does not install specific versions of R, you should still specify the version of R you used in your script.

```
R.version.string
```

```
## [1] "R version 4.3.2 Patched (2023-10-31 r85451 ucrt)"
```

You can find the release dates for all versions of R via the CRAN archive page.

Excursus: Updating R

As you probably know, you can download the most recent version of R from the CRAN website. You can download older versions of R via the CRAN archive packages for *Windows* and *Mac OS X*.[1]

[1] How you install and update R on Linux depends on your distribution.

Excursus: Updating R

On Windows, you can also use the `installr` package to update R.[1]

```
install.packages("installr")
library(installr)

updateR()
```

[1] The `installr` package also offers some interesting other functionalities, such as installing `Git` via the `install.git()` function.

Excursus: Updating packages

The easiest way of updating packages is simply using
`install.packages()`. To update all packages or a specific set of
packages, you can use the `update.packages()` function.

```r
# update all installed packages
update.packages()

# update specific packages
update.packages(oldPkgs = c("tidyverse", "janitor", "sjPlot"))
```

Excursus: Updating packages

With the following code, you can detach and update all of the packages you have currently loaded in your R session (excluding core R packages):

```r
loaded_pkgs <- search()
loaded_pkgs <- loaded_pkgs[grep("^package:", loaded_pkgs)]

exclude_pkgs <- c("package:base", "package:stats", "package:graphics", "package:g
                  "package:utils", "package:datasets", "package:methods", "packag
loaded_pkgs <- loaded_pkgs[!loaded_pkgs %in% exclude_pkgs]

for (pkg in loaded_pkgs) {
  detach(pkg, character.only = TRUE, unload = TRUE)
}

update_pkgs <- gsub("^package:", "", loaded_pkgs)

install.packages(update_pkgs)
```

Excursus: Updating packages

If you want to install a specific version of a package (not the most recent one), the easiest option is to use a function from the `remotes` package.

```r
library(remotes)

install_version("tidyverse", version = "1.3.0")
```

Exercise time 🏋️ 💪 🏃 🚴

Solutions

More comprehensive dependency management options in `R`

`renv` and its advantages compared to `groundhog`

Let's talk about another option: `renv`

- `renv` is integrated with *RStudio*

- `renv` is developed by professional software engineers (rather than academic researchers)

- `renv` works with more repositories than `groundhog`

For more information on `groundhog` vs. `renv`, see:
https://groundhogr.com/renv/

What is `renv`?

The package `renv` helps to create reproducible environments for *R projects*

What is an R project?

> "R experts keep all the files associated with a project together — input data, R scripts, analytical results, figures. This is such a wise and common practice that RStudio has built-in support for this via projects."

Source: https://r4ds.had.co.nz/workflow-projects.html

Why?

Use `renv` to make your R projects more isolated, portable and reproducible.

- Isolated: Installing a new or updated package for one project won't break your other projects, and vice versa. That's because `renv` gives each project its own private library.

- Portable: Easily transport your projects from one computer to another, even across different platforms. `renv` makes it easy to install the packages your project depends on.

- Reproducible: `renv` records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

Source: https://rstudio.github.io/renv/

Let's talk about libraries
(and packages)

- When talking about a "reproducible R environment", this mostly (excluding R itself) refers to all the R packages that are being used

- Most of us have *one* "system library" for R packages, where all installed packages can be found (for example: NA)

- "The directories in R where the packages are stored are called the libraries" [1]

[1] https://hbctraining.github.io/Intro-to-R-flipped/lessons/04_introR_packages.html

Is one R system library a good idea?

However, having *one* central system library for R packages may be a bad idea, especially when things break -- and things will break! ("dependency hell")
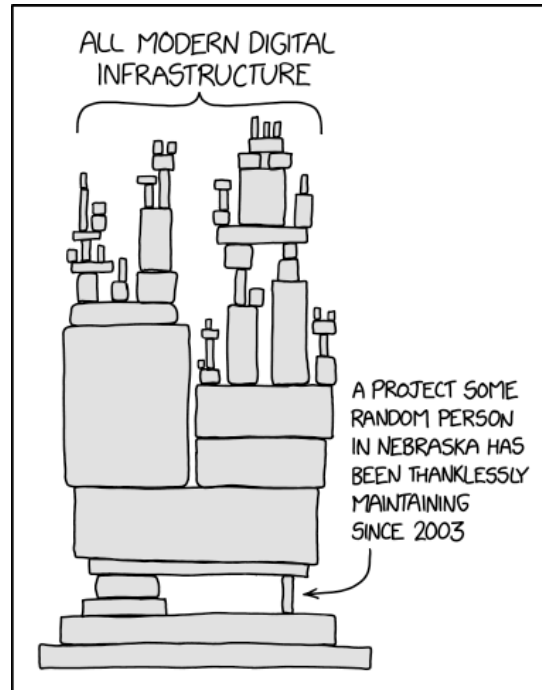
renv offers "project libraries"

- Instead of using *one* central system library for all installed `R` packages, `renv` sets up a "project library", i.e., each `R` project has its own library

- Project libraries means that each project has its own independent collection of R packages, i.e., multiple projects can have the same R package but, e.g., different versions of it

Where all the R packages grow... repositories

- The usual ways to install an R package is to download the package from an repository

- Most famous is CRAN, the Comprehensive R Archive Network (other sources include Bioconductor or GitHub)

- I am currently using:

```r
getOption("repos")
```

```
##                              CRAN
## "https://cran.rstudio.com/"
## attr(,"RStudio")
## [1] TRUE
```

renv and repositories

- `renv` creates an `R` project-related library and installs packages into that specific library

- Hence, it needs to know where (a) package(s) can be found; not all packages can be found on CRAN

How does renv work?

- It is assumed that your work is based on the model of an "`R project`" (in *RStudio*)

- Similar to Git, you first have to initialize your `R` project, letting `renv` know that it should take care of all things that are related to a reproducible `R` environment

- At some moment in time, when everything is working fine, and you want to preserve this state of your `R` project, you can take an `R`-related "snapshot" of your project

- This snapshot can be shared with others or your future self, guaranteeing that you can always restore this moment in time when everything in your `R` project was working well (meaning: in terms of a "reproducible `R` environment")

Workflow in `renv`
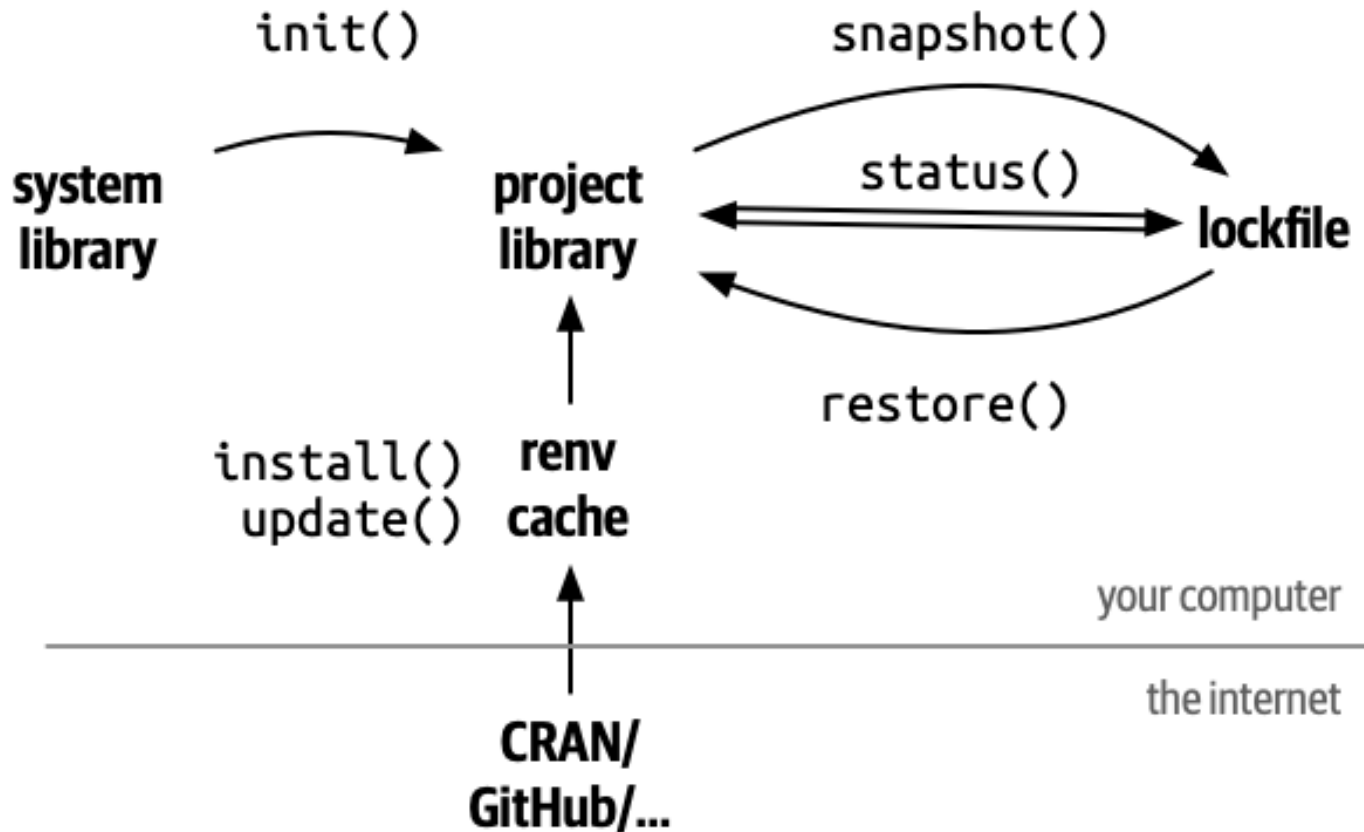
`renv::init()`

- `renv::init()` is run once and adds a few new directories and files to your `R` project

- `renv/library`: This directory is the project library that contains all packages currently used by your project

- `renv.lock`: Records metadata about every package ensuring that it can be re-installed on a new machine

- `.Rprofile`: Runs automatically every time you start `R` (in that project), renv uses this file to configure your `R` session to use the project library

Source: https://rstudio.github.io/renv/articles/renv.html#getting-started

renv in RStudio
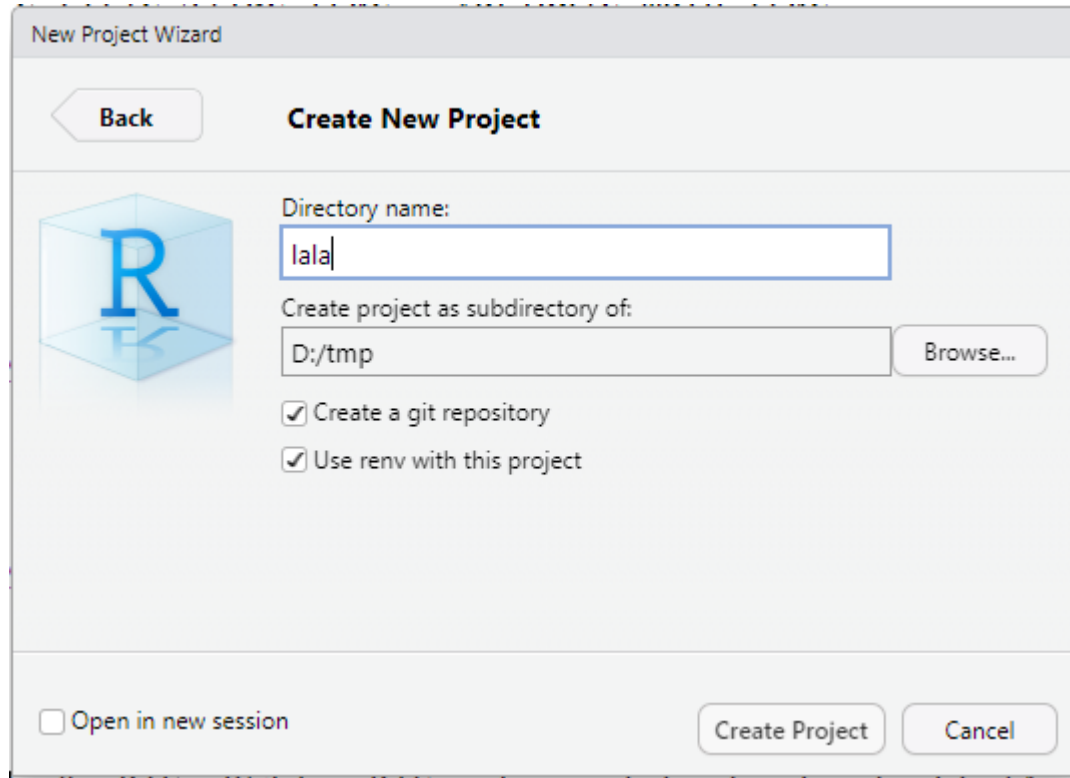
`renv::snapshot()` and `renv::restore()`

- Running `renv::snapshot()` updates `renv.lock` and documents the current collection (and their specific versions) of `R` packages that are being used in your `R` project

- To share an `R` project, you need to provide access to `renv.lock`, `.Rprofile`, `renv/settings.json` and `renv/activate.R` -- or, from a Git perspective, that's all you need to commit into your `Git` repository

- Next, using `renv::restore()` a collaborator can exactly reproduce (download and install specific `R` packages) your `R` environment

Installing additional R packages

Use `install.packages()` or `renv::install()`

Updating R packages in your R project

Use `renv::update()`...

More options for dependency management in $\mathbb{R}$

`rang`

"The goal of rang (Reconstructing Ancient Number-crunching Gears) is to obtain the dependency graph of R packages at a specific time point.

Although this package can also be used to ensure the **current R computational environment** can be reconstructed by future researchers, this package gears towards **reconstructing historical R computational environments** which have not been completely declared. For the former purpose, **packages such as renv, groundhog, miniCRAN, and Require should be used**. One can think of rang as an archaeological tool" (emphasis added).

Source: https://github.com/gesistsa/rang

`rang` (cont'd.)

For more information, see:

- https://chainsawriot.github.io/gesis2023_rang/#/title-slide

- https://github.com/gesistsa/rang