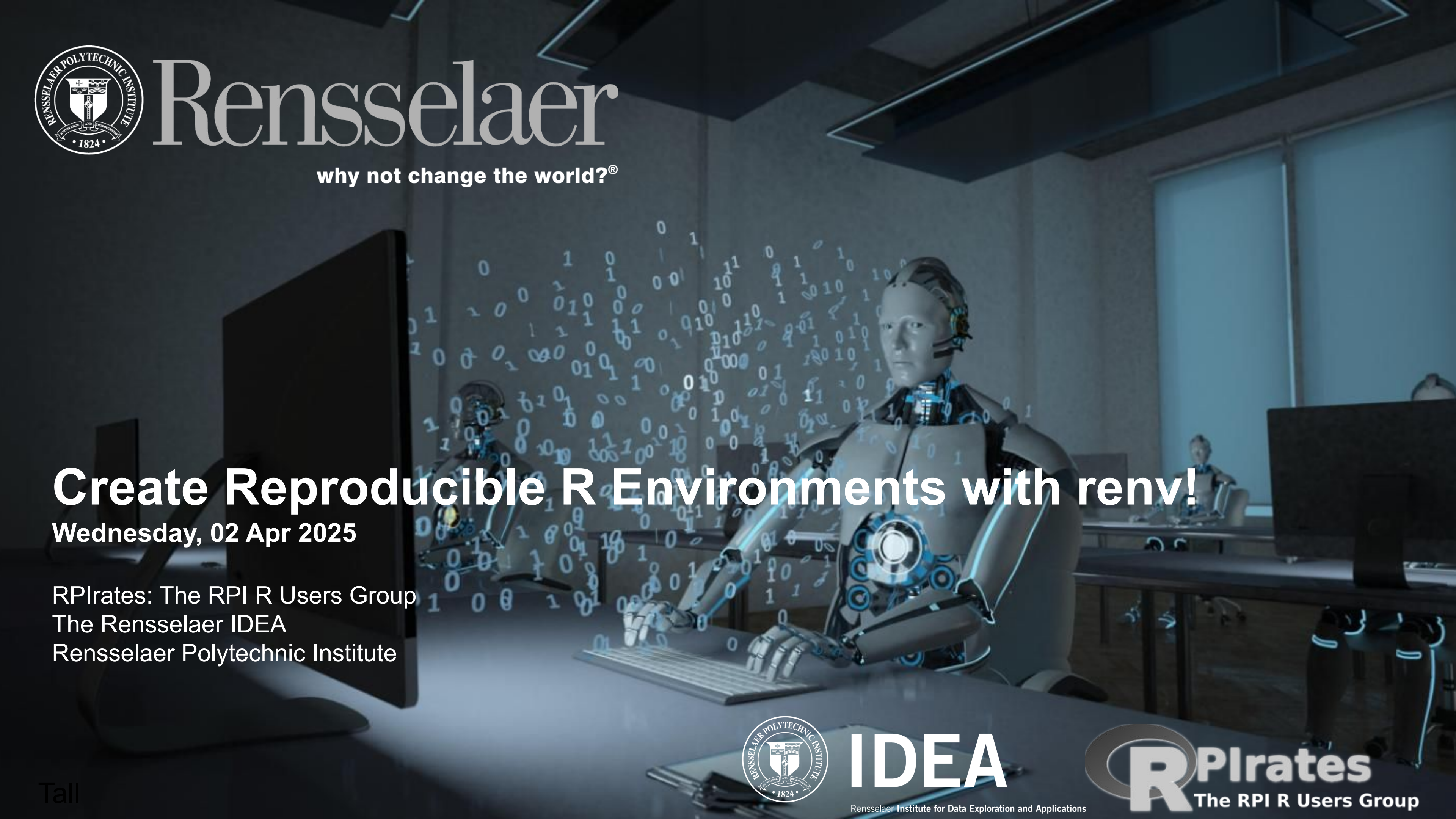


Rensselaer

why not change the world?®

Create Reproducible R Environments with renv!

Wednesday, 02 Apr 2025

RPIrates: The RPI R Users Group
The Rensselaer IDEA
Rensselaer Polytechnic Institute

IDEA
Rensselaer Institute for Data Exploration and Applications

RPIrates
The RPI R Users Group

Tall

renv
www.rstudio.com

These slides were inspired by "Creating and sharing reproducible environments with renv"

UCSB Carpentry. (2022). Reproducible Publications with Rstudio.

The Best of Intentions…

Assume you've followed all the recommended practices to create **reproducible projects**:

- You've chosen RStudio/R: *free* and *open-source*;
- You've used *relative paths*, and produced *clean and clear code*;
- Your project directory and data files…
  - …follow naming conventions
  - …are well-organized
  - …are beautifully documented

Is that all? *Almost, but not quite…*

Reality bites!

Imagine investing significant time and effort into completing a sophisticated data analysis, only to face the **realities of change**:

> *…R gets upgraded!*
>
> *…new package versions are released!*
>
> *…everything suddenly comes crashing down!*

How can you ensure that the packages you rely on for your project remain on the same version consistently, *even in the future*?

Welcome to Dependency Hell

2021-08-18 | Oliver Klee | @oliklee

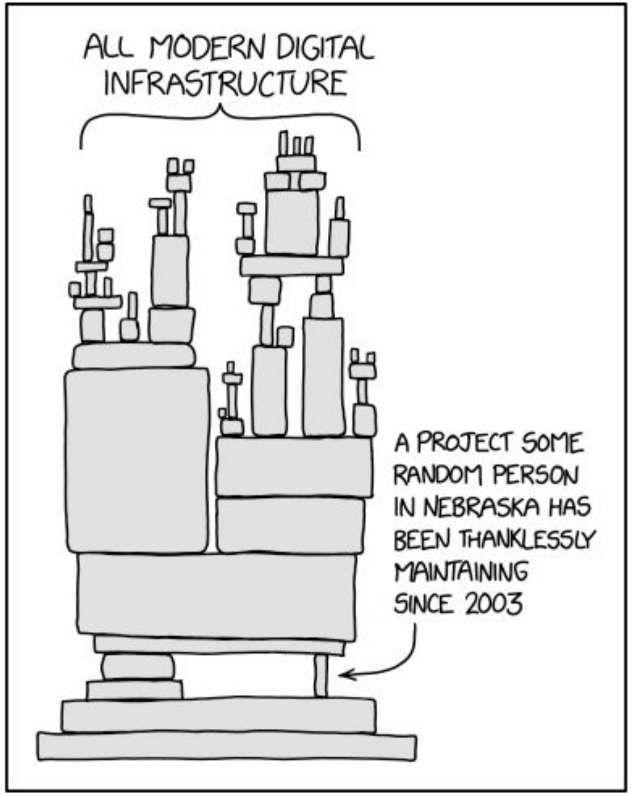
Photo by JJ Ying on Unsplash
https://unsplash.com/photos/YTtDxJPflqk

Avoiding Dependency Hell

One common reproducibility issue in research relates to *software dependencies*

"Dependency hell"

- A situation in software development where the *complex web of dependencies* between different software components becomes difficult to manage or resolve
- Occurs when multiple software libraries or packages have *conflicting or incompatible requirements* regarding the *versions of other libraries or packages* they rely on


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Avoiding Dependency Hell

Let's say your project uses two R packages: "package X" and "package Y":

```
"package X" version 1.0.0 depends on "package Z" version 2.0.0.
```

```
"package Y" version 1.5.0 depends on "package Z" version 1.0.0.
```

- You start your project by installing the latest versions of both packages
  - Everything works fine, and you proceed with your analysis using both packages
- A *few months later*, you must reproduce your analysis and reinstall the packages
- Unknown to you, "package Z" has been updated since your last installation
  - The new version is incompatible with the older version that "package Y" depends on.
- When you try to run your analysis, you encounter errors or unexpected behavior because "package Y" is no longer compatible with the updated version of "package Z"

This **mismatch in dependencies** can result in **reproducibility issues**, making it challenging to replicate your previous results

Avoiding Dependency Hell

Before sharing a project, ask yourself:

- What are the *versions of the packages* used for my project?
- Will my project be *usable on other systems* and/or *in the future?*
- How do we deal with new projects requiring different versions of a package?

Enhancing reproducibility with renv

It's important to maintain a **record of package versions** used in your analysis and **create a reproducible environment**; that's where the renv package comes in!

- renv enables you to maintain the **specific versions** of packages your project depends on
  - Ensures *stability* and *compatibility* throughout the project lifecycle by "freezing" package versions
  - Protects your project from unexpected issues or incompatibilities that may arise from updates
- renv allows you to create **isolated project-specific environments**
  - Captures specific versions of packages
  - Ensures that the same versions are used every time you reproduce your work

This makes it easy for an unwitting downstream user (maybe you!) to run an R project in the same environment in which it was originally developed.

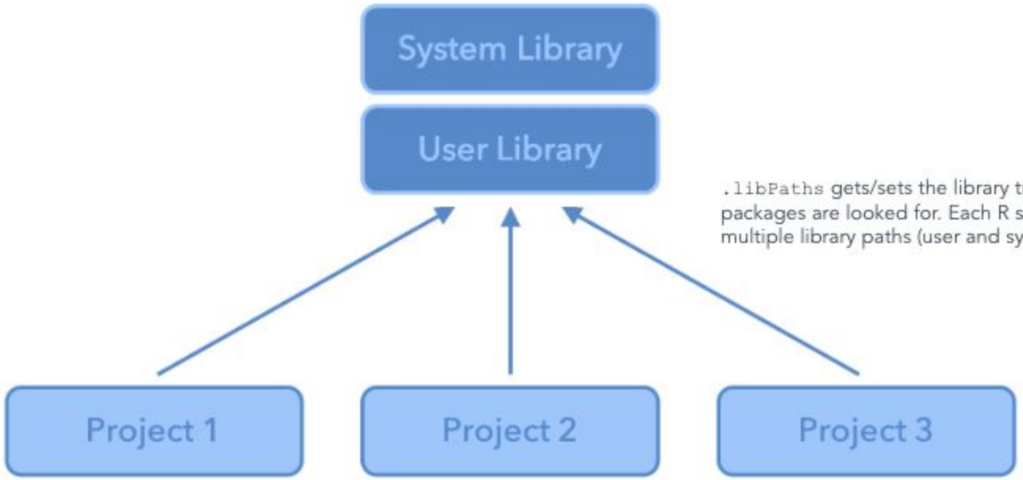
Enhancing reproducibility with renv

In sum, there are three main advantages of using renv:

- **Isolation:** Each project gets its own library of R packages.
  - In this way, you can upgrade and change package versions in one project without worrying about your other projects.
- **Portability:** You can more easily share and collaborate on projects while ensuring all are sharing the same common base
  - by sharing a "lockfile" (**renv.lock**) which captures the state of your R packages.
- **Reproducibility:** You can restore your R library exactly as specified in the renv.lock file.

How does renv work? Packages, libraries & projects

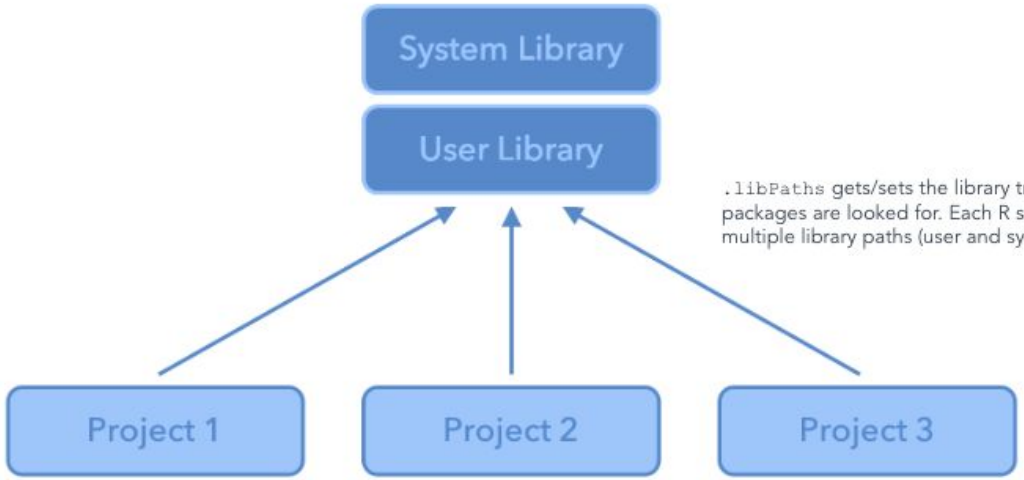
- A **package** is a collection of functions, data, and compiled code
- **Libraries** are the locations where packages live
- You may have multiple **projects** with different dependencies…
- …but are calling packages from the same library
- By default, we have two libraries, *System* and *User*



`.libPaths` gets/sets the library trees within which packages are looked for. Each R session can use multiple library paths (user and system)

How does renv work? Life without renv

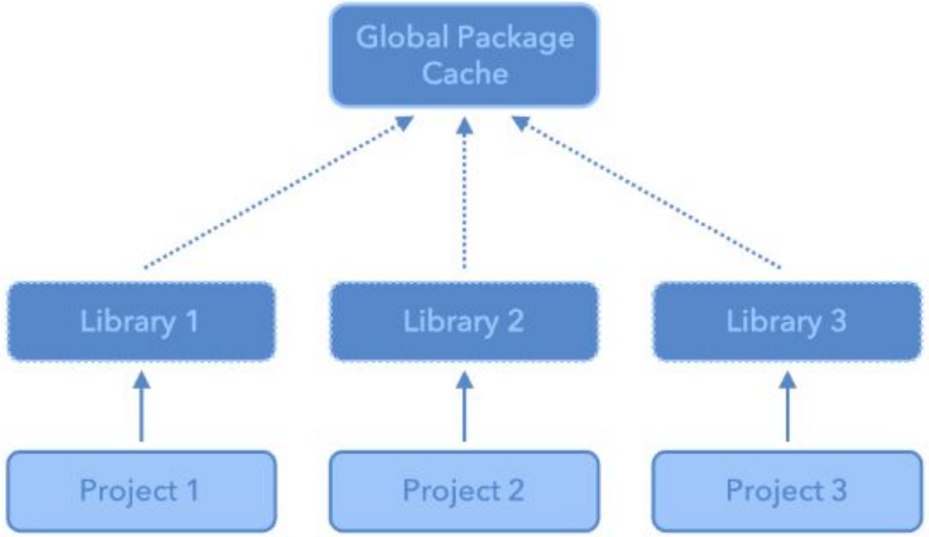
- **The problem:** You will be handling multiple projects in different points in time and with unique dependencies
- Every time you start a fresh project and use **install.packages()** you will grab the *latest version* of a given package from CRAN and re-install, *replacing* the package in your environment



`.libPaths` gets/sets the library trees within which packages are looked for. Each R session can use multiple library paths (user and system)

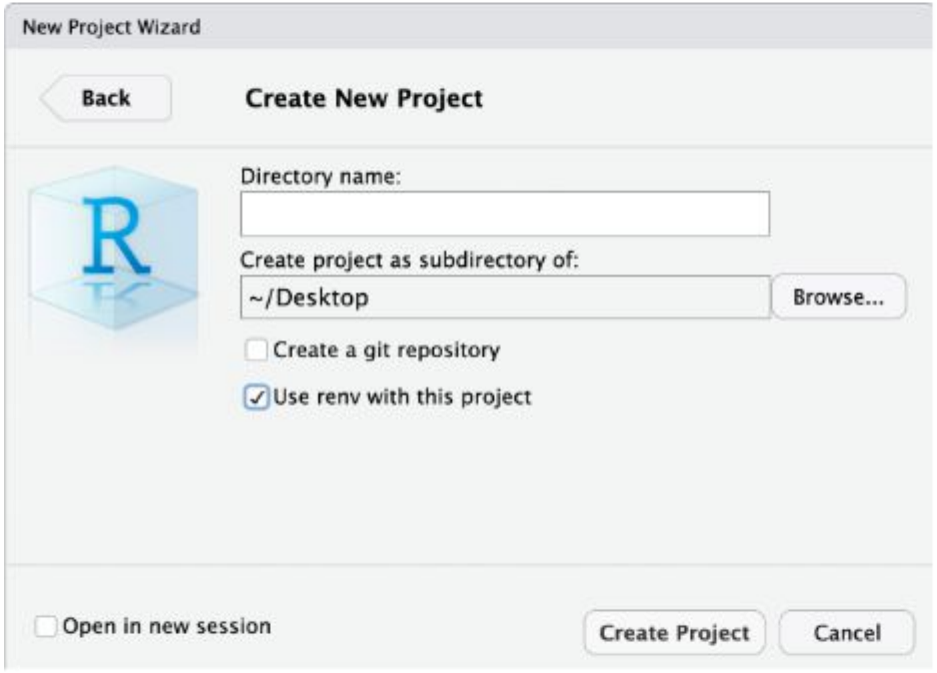
How does renv work? Life with renv

- Renv creates a local project library for each project, encapsulating dependencies so you can easil
  re-run results for each project using the original versions of packages



Using renv: starting a new project

When creating a new project using the new project wizard, make sure to select the option
Use renv with this project


New Project Wizard

Create New Project

Directory name:

Create project as subdirectory of: ~/Desktop

[ ] Create a git repository

[✓] Use renv with this project

[ ] Open in new session

Using renv: Enabling renv on an existing project

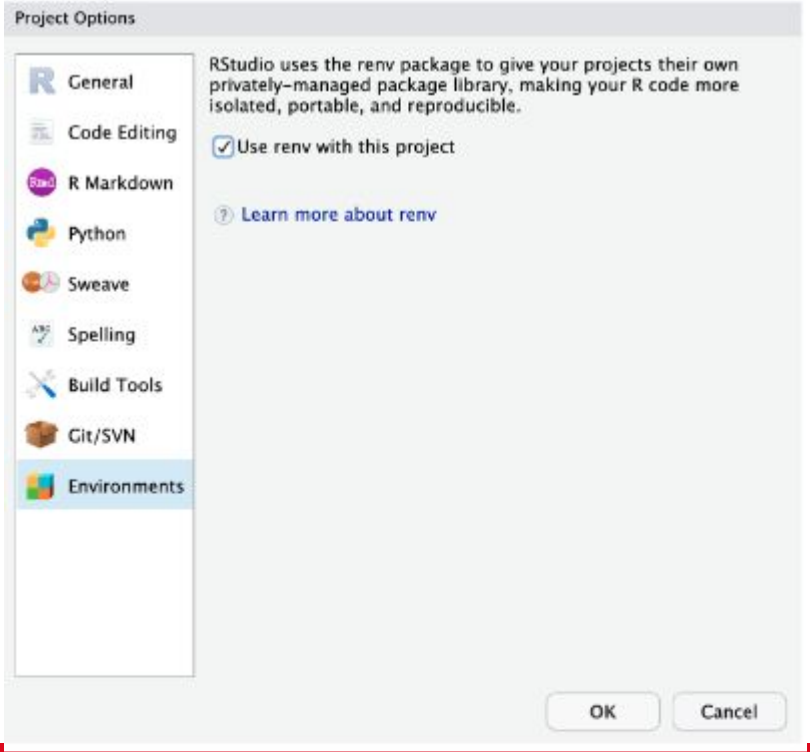
If you've started a project without renv but wish to add it:

Enable renv by clicking

Tools>Project Options…>Environments

…and selecting…

Use renv with this project



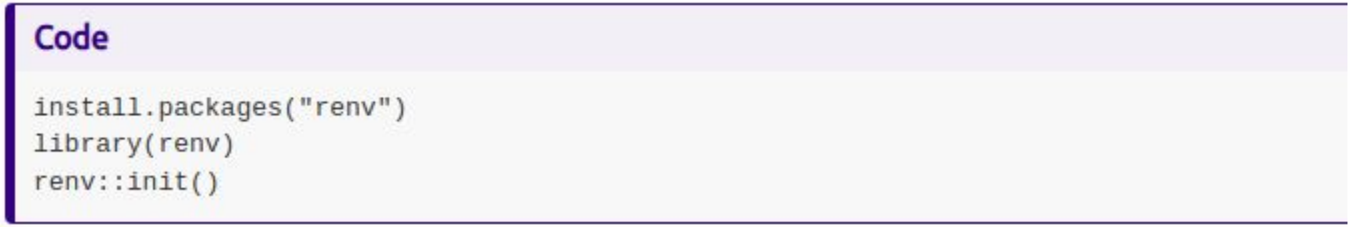
RStudio uses the renv package to give your projects their own privately-managed package library, making your R code more isolated, portable, and reproducible.

[✓] Use renv with this project

Learn more about renv

Using renv: Enabling renv via the R console

Another alternative is to install the renv package, then load it and run the **renv::init()** command in the console pane:



Code

```
install.packages("renv")
library(renv)
renv::init()
```

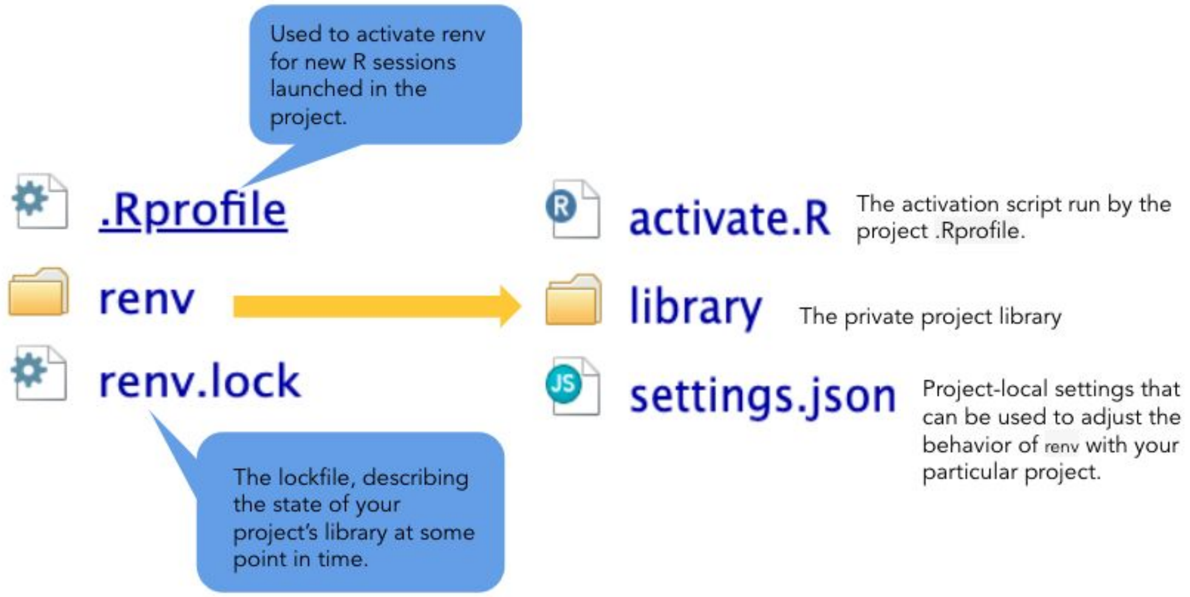
Enabling renv creates the initial **renv.lock** file:

- A JSON file in the project directory
- Records the precise versions of each installed package
- Other metadata such as package sources and checksums

Using renv: Enabling renv

Enabling renv also creates:

- An **.Rprofile** file
  - Activates renv for new R sessions
- An **renv folder** containing the files specified to the right


Used to activate renv for new R sessions launched in the project.
.Rprofile
renv
renv.lock
The lockfile, describing the state of your project's library at some point in time.
activate.R
The activation script run by the project .Rprofile.
library
The private project library
settings.json
Project-local settings that can be used to adjust the behavior of renv with your particular project.

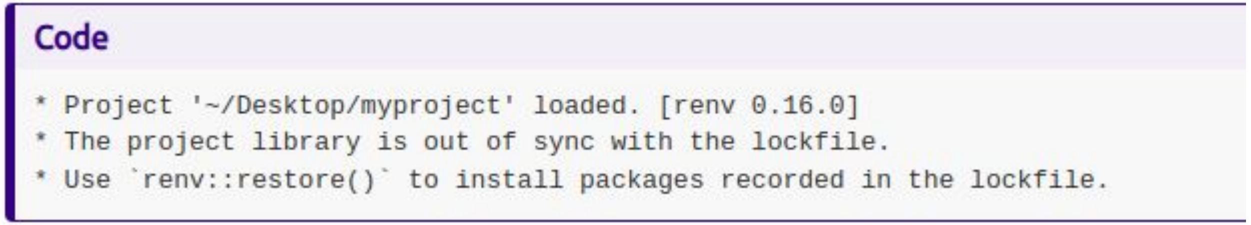
Using renv: Updating packages

What if you need to update packages for your project?

- Whenever you start using a new package (or otherwise change your project’s dependencies), run **renv::snapshot()** to update **renv.lock**
- If you're using **git** and start using renv, you will notice that renv creates several files and directories in addition to renv.lock
- When done, you should **commit** these files to your project’s git repository

Using renv: Restoring an existing project

- When you open a project for which renv has been set up, renv automatically runs and checks that the installed package versions match those of the project.
- If versions match, there is nothing to do
- If there are any mismatches, renv will print a warning resembling the following:


Code

```
* Project '~/Desktop/myproject' loaded. [renv 0.16.0]
* The project library is out of sync with the lockfile.
* Use `renv::restore()` to install packages recorded in the lockfile.
```

Using renv: Restoring an existing project

- If this happens, run **renv::restore()** from the console to download and install the package versions needed to match the project's requirements.
- For example, if the project uses **tidyverse 1.3.2** and you have an older version **tidyverse 1.3.1** installed, *renv will upgrade your RStudio installation to tidyverse 1.3.2*
- Conversely, if the project uses an *older version* of a package than you have installed, renv will attempt to download and install the older version for you.
- Don't worry about losing the newer version; *renv ensures that all versions of all packages remain installed on your computer, available for use by projects as needed*

Summary: How to collaborate using renv

- **Initialize** renv using **renv::init( )**
- **Share** project sources (data and code), and include **renv.lock, .Rprofile**, and **renv/activate.R** to ensure that collaborators download and install the right version of renv when starting the project
- When a collaborator opens the project, renv will **automatically** bootstrap and download the appropriate version of renv
- If updates are made, save them with **renv::snapshot( )**
- Collaborators can use **renv::restore( )** to restore the project library on their machine if needed

renv Caveats

renv is **not a panacea** for reproducibility; it is a tool that can help make projects reproducible by helping with one part of the overall problem: *R packages*. *There are a number of other pieces that renv doesn't currently provide much help with:*

- **R version:** renv tracks, but doesn't help with, the version of R used with the project
  - Tools like **rig** might help; they make it easier to switch between multiple versions of R on one system
- **Pandoc: rmarkdown** relies heavily on pandoc, *but pandoc is not bundled with the rmarkdown package*
  - Restoring rmarkdown from the lockfile is insufficient to guarantee exactly the same rendering of RMarkdown documents
  - The tools provided by the pandoc package might be useful
- **Operating system, versions of system libraries, compiler versions:** Keeping a 'stable' machine image is a separate challenge, but Docker is one popular solution.
  - See **vignette("docker", package = "renv")** for recommendations on how Docker can be used together with renv

renv and R project resources

- Creating and sharing reproducible environments with renv
- Personal R Administration: renv
- R for Data Science (2e): Workflow: scripts and projects (Hadley!)
- Introduction to renv (renv vignette)
- Using RStudio Projects (Posit)
- RStudio User Guide: renv (Posit)