



Managing Packages and Environments

Childhood Cancer Data Lab

Software is called “soft” for a reason

- Software is always changing!
 - New versions can bring new features and fix bugs 🎉
 - But also remove features you relied on 😞
 - Or alter behavior in unexpected ways 😱

- Changes in one piece of software can break other software
 - Sometimes this is intentional! Operating systems are constantly updating to break hacking tools

Changes occur at every level of the computing “stack”

- Individual scripts/analyses
- Packages within R, Python, etc.
- Individual programs (Cell Ranger, Salmon, etc., but also R & Python)
- Operating system
- Hardware

We want to do our best to **track** and **document** versions of as many layers as possible.

Ideally, we would like to **freeze** versions, so we and anyone else can come back and know results will be the same!



The “analysis” layer

- We have already talked about tracking your changes with Git and GitHub
 - If you know which commit of your scripts you used to produce an analysis, you can point people right to that
 - “**tags**” and “**releases**” on GitHub can make this easier when you have a particular commit you want to share (but we won’t be covering that in this workshop)

The package layer

- Research software tools in bioinformatics (and beyond) are often published as packages for R or Python
 - Examples: Seurat, scanpy, tidyverse, pandas, Bioconductor packages
- This makes them generally easy to install and update as research progresses
 - BUT easy to update means things can change fast
 - New versions may change results, even for existing functions!
 - Newer isn't always better; sometime you want to stick with the old way
 - Dependencies on other packages may require specific versions of other packages

Documenting package versions in R

- `sessionInfo()` is your friend
 - or `sessioninfo::session_info()`

```
R version 4.1.2 (2021-11-01)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Monterey 12.4

Matrix products: default
LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Reso

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_

attached base packages:
[1] stats      graphics  grDevices  utils      datasets  met

other attached packages:
[1] magrittr_2.0.3 ggplot2_3.3.6 dplyr_1.0.9

loaded via a namespace (and not attached):
 [1] bslib_0.3.1      jquerylib_0.1.4 RColorBrewer_1
 [5] compiler_4.1.2  tools_4.1.2      digest_0.6.29
 [9] jsonlite_1.8.0  evaluate_0.15    lifecycle_1.0.
[13] gtable_0.3.0    pkgconfig_2.0.3  rlang_1.0.2
[17] rstudioapi_0.13 yaml_2.3.5       parallel_4.1.2
```

```
— Session info —
setting  value
version  R version 4.1.2 (2021-11-01)
os       macOS Monterey 12.4
system   x86_64, darwin17.0
ui       RStudio
language (EN)
collate  en_US.UTF-8
ctype    en_US.UTF-8
tz       America/New_York
date     2022-05-31
rstudio  2022.02.2+485 Prairie Trillium (desktop)
pandoc   2.17.1.1 @ /Applications/RStudio.app/Contents/MacOS/quarto/bin/ (via rmarkdown)

— Packages —
package * version date (UTC) lib source
bit      4.0.4   2020-08-04 [1] CRAN (R 4.1.0)
bit64    4.0.5   2020-08-30 [1] CRAN (R 4.1.0)
bslib    0.3.1   2021-10-06 [1] CRAN (R 4.1.0)
cli      3.3.0   2022-04-25 [1] CRAN (R 4.1.2)
colorspace 2.0-3   2022-02-21 [1] CRAN (R 4.1.2)
crayon   1.5.1   2022-03-26 [1] CRAN (R 4.1.2)
digest   0.6.29  2021-12-01 [1] CRAN (R 4.1.0)
dplyr    * 1.0.9   2022-04-28 [1] CRAN (R 4.1.2)
ellipsis 0.3.2   2021-04-29 [1] CRAN (R 4.1.0)
evaluate 0.15    2022-02-18 [1] CRAN (R 4.1.2)
```

But how do you recreate the same set of packages?

Installing packages based on `sessionInfo()` output could be very tedious!

Enter **renv**!

- **renv** is an R package for *tracking, freezing, and sharing* R environments, including all of the package versions that were installed.
- Each project can have its own environment, with its own set of packages
 - Different projects may require different versions of packages
 - **renv** can help manage these different sets of package versions
- When sharing a project/analysis, using **renv** allows everyone stay in sync with same packages and versions

How does renv work?

Rather than using the system R package library, renv creates a library for each project that R will use when running code for the project ("Project Library")

This renv-created library could be large, so we can't reasonably share the whole thing.

Instead, we create a file (`renv.lock`) that describes the library.

renv uses this file to track all of the packages we are using, and recreate the library with those packages as needed.



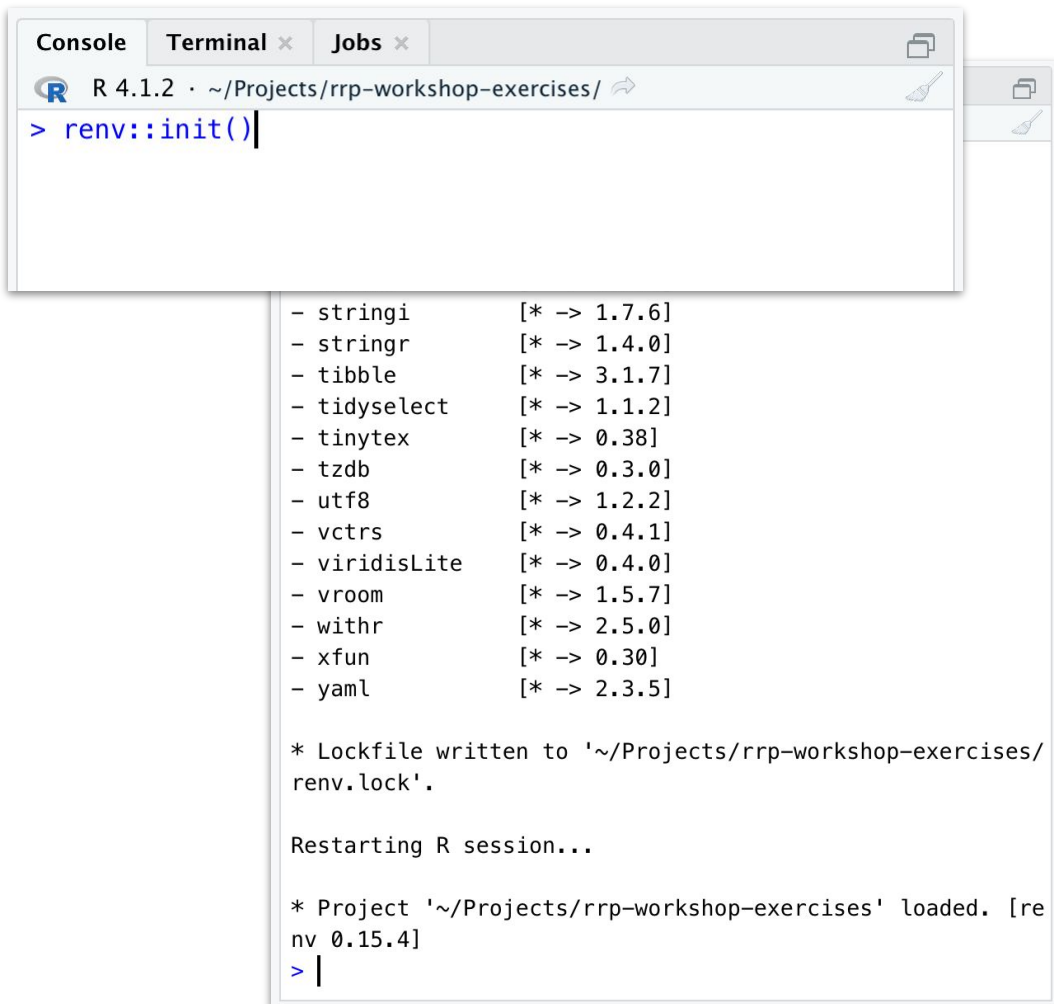
renv initialization

In the console, enter:

```
renv::init()
```

Lots of text will scroll by, and your R session will restart.

That's it! You are starting to track your R packages!



```
R 4.1.2 · ~/Projects/rrp-workshop-exercises/
> renv::init()

- stringi      [* -> 1.7.6]
- stringr     [* -> 1.4.0]
- tibble      [* -> 3.1.7]
- tidyselect  [* -> 1.1.2]
- tinytex     [* -> 0.38]
- tzdb        [* -> 0.3.0]
- utf8        [* -> 1.2.2]
- vctrs       [* -> 0.4.1]
- viridisLite [* -> 0.4.0]
- vroom       [* -> 1.5.7]
- withr       [* -> 2.5.0]
- xfun        [* -> 0.30]
- yaml        [* -> 2.3.5]

* Lockfile written to '~/Projects/rrp-workshop-exercises/
renv.lock'.

Restarting R session...

* Project '~/Projects/rrp-workshop-exercises' loaded. [re
nv 0.15.4]
> |
```

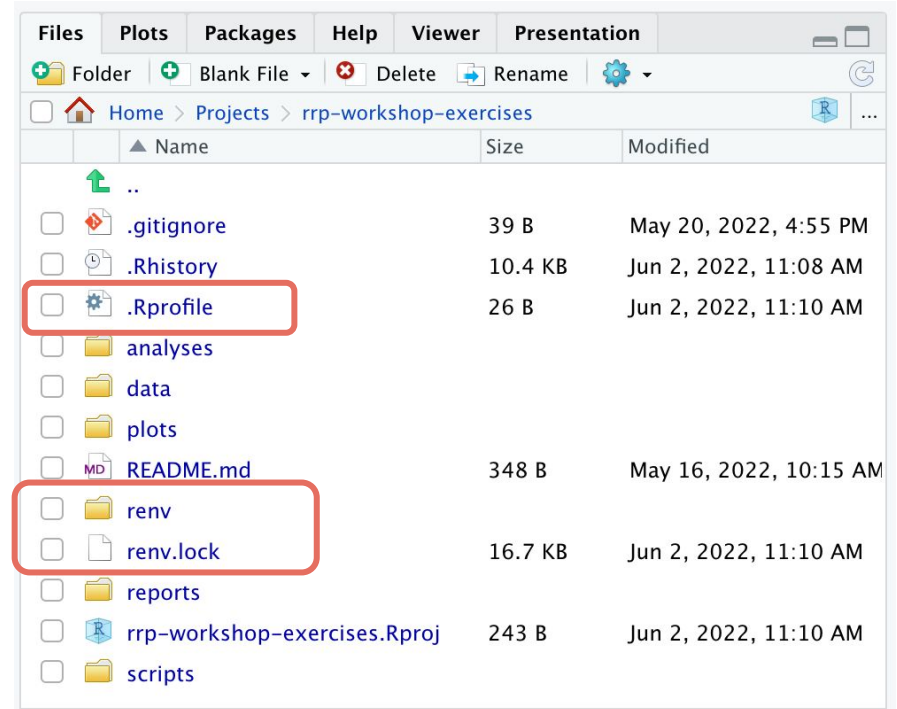
What did `renv::init()` do?

Added an `renv.lock` file, `renv/` folder and `.Rprofile` file
(or modified the one you had)

The `.Rprofile` file is run when R launches *for this project*, and it contains a command to configure `renv` on launch.

The `renv/` folder is where the Project Library and support files can be found

Newly installed packages for the project will be stored in this Project Library



The renv .lock file

Taking a snapshot creates or updates the renv .lock file at the base of your project.

This file records...

- Which packages are installed
- The package versions
- Where the packages came from

Do not edit this file manually!



```
1 {
2   "R": {
3     "Version": "4.1.2",
4     "Repositories": [
5       {
6         "Name": "CRAN",
7         "URL": "https://cran.rstudio.com"
8       }
9     ]
10  },
11  "Packages": {
12    "R6": {
13      "Package": "R6",
14      "Version": "2.5.1",
15      "Source": "Repository",
16      "Repository": "CRAN",
17      "Hash": "470851b6d5d0ac559e9d01bb352b4021",
18      "Requirements": []
19    },
20    "bit": {
21      "Package": "bit",
22      "Version": "4.0.4",
23      "Source": "Repository",
24      "Repository": "CRAN",
25      "Hash": "f36715f14d94678eea9933af927bc15d",
26      "Requirements": []
27    },
28    "bit64": {
29      "Package": "bit64",
30      "Version": "4.0.5",
31      "Source": "Repository",
32      "Repository": "CRAN",
33      "Hash": "9fe98599ca456d6552421db0d6772d8f",
34      "Requirements": [
35        "bit"
36      ]
37    },
38  },
39 }
```

Updating the renv .lock file

The screenshot shows the RStudio interface with the following components:

- Console:** Lists installed packages and their versions, such as `rappdirs` (0.3.3), `readr` (2.1.2), `renv` (0.15.4), `rlang` (1.0.2), `rmarkdown` (2.14), `rprojroot` (2.0.3), `sass` (0.4.1), `scales` (1.2.0), `stringi` (1.7.6), `stringr` (1.4.0), `tibble` (3.1.7), `tidyselect` (1.1.2), `tinytex` (0.38), `tzdb` (0.3.0), `utf8` (1.2.2), `vctrs` (0.4.1), `viridisLite` (0.4.0), `vroom` (1.5.7), `withr` (2.5.0), `xfun` (0.30), and `yaml` (2.3.5). It also shows the lockfile path: `~/Projects/rrp-workshop-exercises/renv.lock'`.
- Terminal:** Shows the message `Restarting R session...`.
- Environment:** Shows `Environment is empty`.
- Files/Plots/Packages/Help/Viewer/Presentation:** The `renv` menu is open, showing options: `Introduction to renv`, `Snapshot Library...` (highlighted with a red arrow), and `Restore Library...`.
- Project Library:** A table of installed packages with columns for Name, Description, Version, and Source.

Name	Description	Version	Source
base64enc	Tool-encoding	0.1-3	Repos
bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.0.4	Repos
bit64	A S3 Class for Vectors of 64bit Integers	4.0.5	Repos
bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.3.1	Repos
cli	Helpers for Developing Command Line Interfaces	3.3.0	Repos
clipr	Read and Write from the System Clipboard	0.8.0	Repos
colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	2.0-3	Repos
con11	A C++11 Interface for R	0.4.2	Repos

An “renv” menu now appears in the Packages pane

Use “Snapshot Library...” to update the renv .lock file to the current setup, e.g. after you update or install new packages

Alternatively, in the console enter:
`renv::snapshot()`

Restoring a library from an `renv.lock` file

When working on a new machine, or if someone else updated the `renv.lock` file, you may need to update the Project Library

- * Project '`~/Projects/rrp-workshop-exercises`' loaded. [renv 0.15.4]
- * The project library is out of sync with the lockfile.
- * Use `renv::restore()` to install packages recorded in the lockfile.

Follow the instructions! Enter `renv::restore()` in the console (or use the `renv` menu "Restore Library" option) to sync your Package Library with the recorded versions, installing any missing packages.

Which packages are included in `renv.lock`?

You might have many packages installed, but only use some in a given project.

`renv` tries to be smart about this, and only includes packages that it finds *used* within code in the project folder, or packages that are required by the packages that are used (so-called *dependencies*)

Sometimes `renv` misses a package (particularly for packages with optional dependencies), and you might need to create a file (we usually call ours `dependencies.R`) that only contains lines like:

```
library(missing_package)
```

which will force `renv` to include that package in the lockfile.

Other package management systems

- **renv** is pretty useful, but it only gets us so far... only R packages (and a bit of Python, in some situations)
 - For software outside R, other package management systems are required
- **conda** is one of the more popular and flexible package managers
 - Started as a Python package manager, but it can be used for any command line software
 - Like **renv**, you can create separate sets of software with different versions for different projects
 - LOTS of bioinformatics software available through **bioconda**
 - <https://bioconda.github.io/user/install.html>
- **Docker** and **Singularity** are another level up
 - “Containers” that include everything from the operating system up
 - Run one OS inside another, with *all the things* frozen to particular versions
 - Cloud platforms love containers...