PharmaSUG 2025 - Paper SD-220

Building Robust R Workflows: Renv for Version Control and Environment Reproducibility

Junze Zhang, Merck & Co., Inc.;
Joshua J. Cook, University of West Florida (UWF)

ABSTRACT

Reproducibility and consistency are vital in pharmaceutical analytics, where analyses must meet rigorous industry standards. Managing R environments effectively ensures consistent and reproducible results across projects, teams, and systems. This paper focuses on the use of renv, a powerful R package designed to achieve version control and environment reproducibility. We will demonstrate how renv allows users to create isolated project environments, lock package versions, and restore dependencies seamlessly across systems and teams, ensuring uniformity in analytical outputs. Attendees will learn practical workflows for initializing and managing renv environments, sharing them among team members, and resolving compatibility challenges. The session will also explore how renv integrates with version control systems, such as git, enabling teams to manage R environment changes alongside code revisions, thus reducing errors and enhancing collaboration. Additionally, we will showcase how renv can be combined with reporting tools like Quarto to create dynamic, reproducible outputs tailored for regulatory submissions with an emphasis on team collaboration. Through real-world examples and actionable insights, participants will leave equipped to harness renv for improved version control and workflow consistency in their pharmaceutical analytics projects. Whether working with clinical trial data or preparing regulatory reports, this session offers the tools needed to elevate R workflows to industry-leading standards. Join us to explore the transformative potential of renv for reproducible and efficient analytics in pharmaceutical research.

INTRODUCTION

Managing R environments effectively is critical in pharmaceutical analytics, where reproducibility and consistency are paramount. In pharmaceutical research, analyses must consistently adhere to stringent industry standards, making the management of computational environments a fundamental component of project success. A primary challenge encountered by analysts, data scientists, and clinical programmers alike is maintaining consistent R environments across multiple systems, teams, and stages of a project lifecycle. Failure to maintain reproducible environments can lead to significant delays, audit findings, or even regulatory rejections, highlighting the critical need for robust environment management solutions.

The R package renv addresses this challenge by providing robust tools for version control and reproducibility within R workflows. Through the use of isolated, reproducible project environments, renv enables analysts to control and precisely replicate software dependencies. It captures a detailed snapshot of the R environment in a single renv.lock file, facilitating reproducibility and reducing potential compatibility issues.

This paper introduces renv as a solution for achieving reliable and reproducible pharmaceutical analytics. Specifically, we demonstrate how renv allows teams to easily track, manage, and restore project-specific R environments through straightforward commands like renv::init(), renv::snapshot(), and renv::restore(). Additionally, we explore best practices for integrating renv with version control systems such as Git, emphasizing how committing the renv.lock file alongside code changes fosters greater transparency, collaboration, and error reduction.

Finally, we briefly illustrate how renv can seamlessly integrate with reporting tools such as Quarto, enabling teams to produce dynamic, reproducible reports suitable for regulatory submissions and collaborative review. Through practical, real-world examples and step-by-step workflows, this paper equips readers with the knowledge and strategies necessary to implement renv effectively, enhancing reproducibility, efficiency, and collaboration in pharmaceutical research. An overview of the topics covered in this paper is shown in Figure 1.

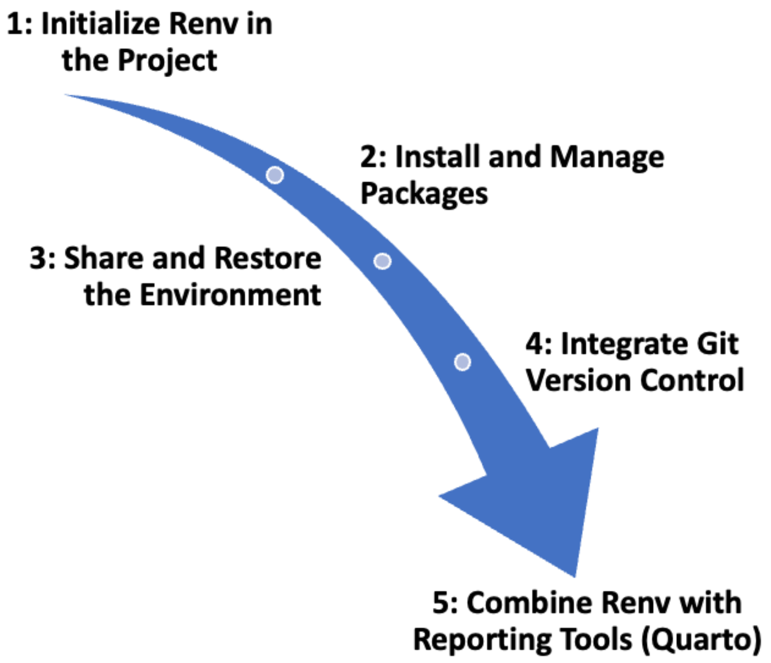

1: Initialize Renv in the Project
2: Install and Manage Packages
3: Share and Restore the Environment
4: Integrate Git Version Control
5: Combine Renv with Reporting Tools (Quarto)

Figure 1. An outline of the topics in this paper to introduce renv as a versatile tool for reproducible analyses.

A SMALL INTRODUCTION TO R AND RSTUDIO

This demonstration utilized **RStudio**® as the primary **Integrated Development Environment (IDE).** Official download links for **R** and **RStudio**® are linked below. As with any project in RStudio®, a primary folder should be setup that will be used as the working directory. This is where the R project (.Rproj) file will be housed, as well as any required subfolders. These examples use the following:

```
 C:\PharmaSUG2025\
```

Create subordinate folders for resources that will be required by markdown, such as data and images:

```
 C:\PharmaSUG2025\data\
 C:\PharmaSUG2025\images\
```

Within RStudio®, only four libraries are installed and imported-tidyverse, which is used for data wrangling and visualization, renv, which is used to create reproducible environments within R, rtrtf, to create production ready tables and figures in RTF format, and quarto, which is an open-source publishing system used to generate reports and other documents. Finally, the data set is imported.

```
packages <- c("tidyverse", "renv", "r2rtf", "quarto")

if (any(!sapply(packages, requireNamespace, quietly = TRUE))) {
  install.packages(packages[!sapply(packages, requireNamespace, quietly = TRUE)])
}

library(tidyverse)
library(renv)
library(r2rtf)
library(quarto)
```

Program 1. Install and import R packages

Using the “New File” button on the toolbar within RStudio®, an R script or Quarto Document can be created and saved to the primary folder with any filename, as shown in Figure 2.

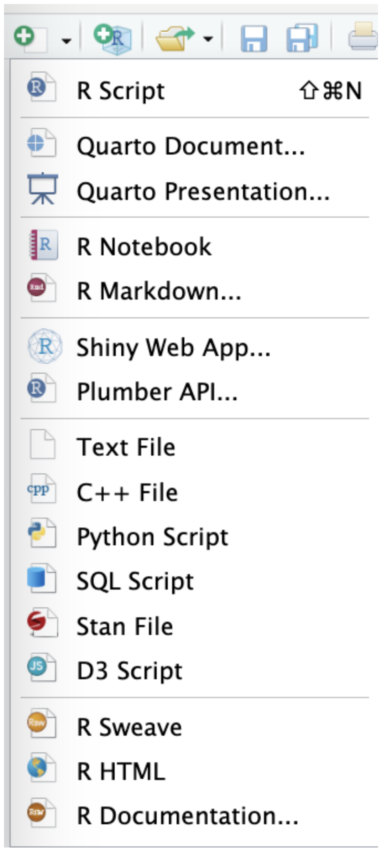


Figure 2. Example of options listed within the “New File” menu in RStudio®.

A TALE OF THE CREATOR AND THE CONSUMER

This guide is structured for two common roles in a clinical trial reporting team: 1. The **Environment Creator:** The person who sets up the project and prepares it for sharing, and 2. The **Environment Consumer:** The person who receives the project and needs to reproduce the exact environment.

SETTING UP A REPRODUCIBLE ENVIRONMENT (FOR ENVIRONMENT CREATORS)

1. QUICK START: CREATING A REPRODUCIBLE PROJECT

1.1 Create a new R project with renv:

- Open RStudio
- Click File > New Project > New Directory
- Check “Use renv with this project”
- Click “Create Project”

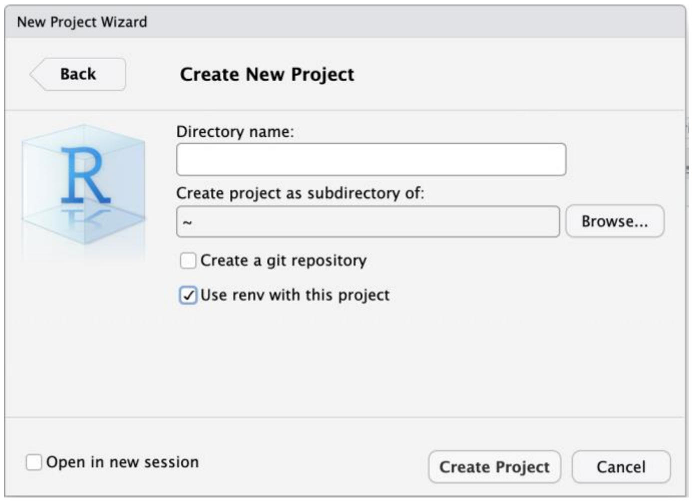
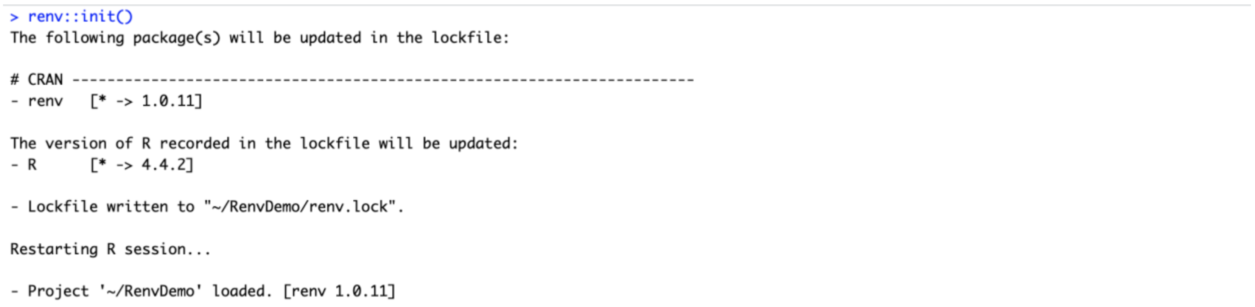


Figure 3. Creating a new project with renv enabled

1.2 Add renv to an existing project:

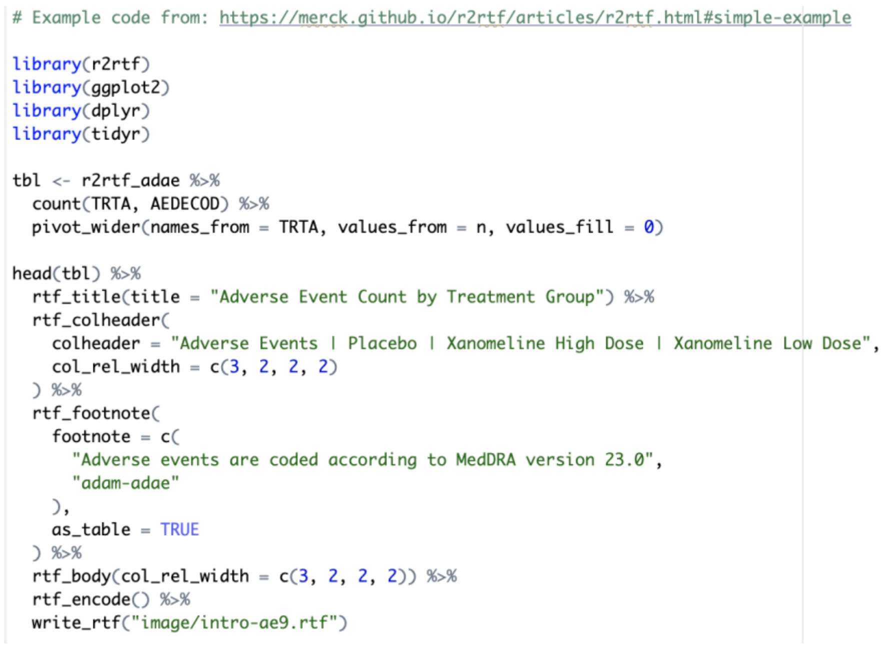
- Open your project in RStudio
- Run install.packages("renv") if not already installed
- Run renv::init()



```
> renv::init()
The following package(s) will be updated in the lockfile:

# CRAN -----------------------------------------------------------------------
- renv   [* -> 1.0.11]

The version of R recorded in the lockfile will be updated:
- R      [* -> 4.4.2]

- Lockfile written to "~/RenvDemo/renv.lock".

Restarting R session...

- Project '~/RenvDemo' loaded. [renv 1.0.11]
```

Figure 4. Initializing renv in an existing project

2. Development (e.g. Create RTF tables for clinical reporting):

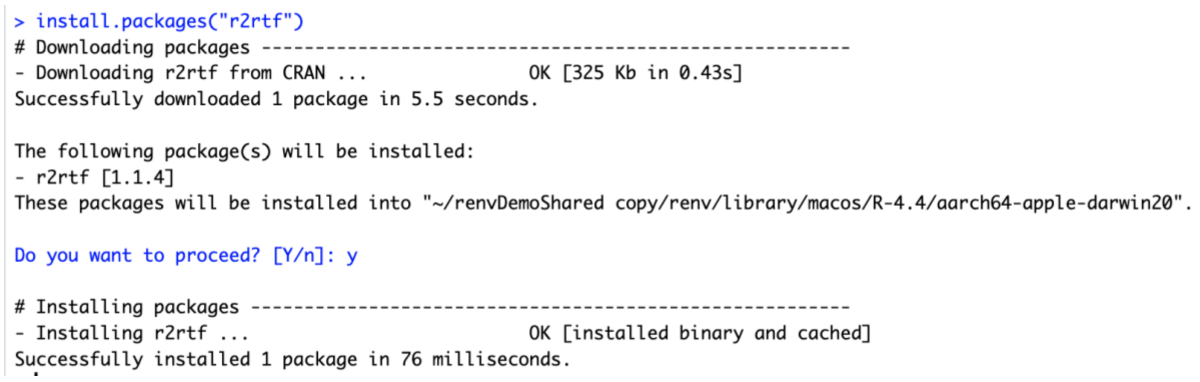


```r
# Example code from: https://merck.github.io/r2rtf/articles/r2rtf.html#simple-example

library(r2rtf)
library(ggplot2)
library(dplyr)
library(tidyr)

tbl <- r2rtf_adae %>%
  count(TRTA, AEDECOD) %>%
  pivot_wider(names_from = TRTA, values_from = n, values_fill = 0)

head(tbl) %>%
  rtf_title(title = "Adverse Event Count by Treatment Group") %>%
  rtf_colheader(
    colheader = "Adverse Events | Placebo | Xanomeline High Dose | Xanomeline Low Dose",
    col_rel_width = c(3, 2, 2, 2)
  ) %>%
  rtf_footnote(
    footnote = c(
      "Adverse events are coded according to MedDRA version 23.0",
      "adam-adae"
    ),
    as_table = TRUE
  ) %>%
  rtf_body(col_rel_width = c(3, 2, 2, 2)) %>%
  rtf_encode() %>%
  write_rtf("image/intro-ae9.rtf")
```

Figure 5. Example development code

3. Install required packages:

- Install packages as usual with install.packages() or renv::install()
- Example: install.packages("r2rtf") for clinical report generation
- Or use renv::install() without arguments to install all packages detected in your code

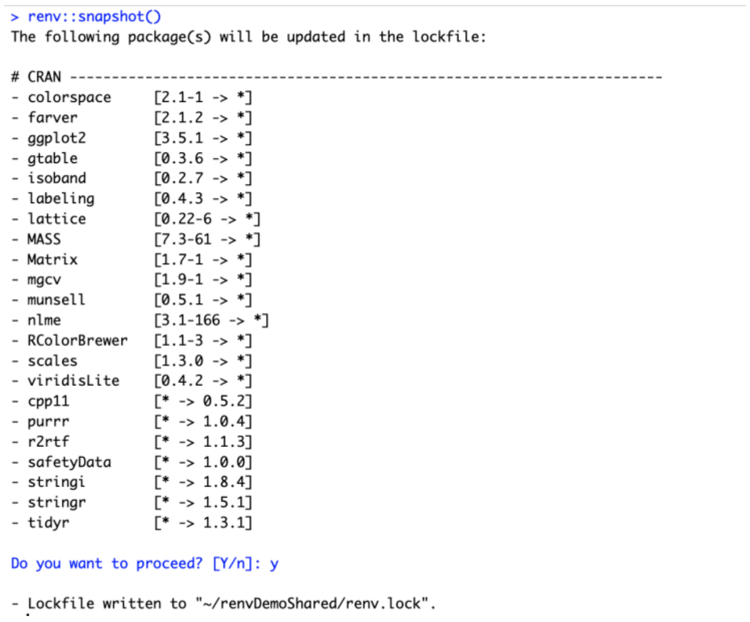


```
> install.packages("r2rtf")
# Downloading packages -------------------------------------------------------
- Downloading r2rtf from CRAN ...               OK [325 Kb in 0.43s]
Successfully downloaded 1 package in 5.5 seconds.

The following package(s) will be installed:
- r2rtf [1.1.4]
These packages will be installed into "~/renvDemoShared copy/renv/library/macos/R-4.4/aarch64-apple-darwin20".

Do you want to proceed? [Y/n]: y

# Installing packages --------------------------------------------------------
- Installing r2rtf ...                          OK [installed binary and cached]
Successfully installed 1 package in 76 milliseconds.
```

Figure 6. Installing specific packages with install.packages()

4. Create a snapshot:

- After installing all necessary packages and testing your code, run renv::snapshot()
- Commit the resulting renv.lock file to version control



```
> renv::snapshot()
The following package(s) will be updated in the lockfile:

# CRAN ---------------------------------------------------------------------
- colorspace     [2.1-1 -> *]
- farver         [2.1.2 -> *]
- ggplot2        [3.5.1 -> *]
- gtable         [0.3.6 -> *]
- isoband        [0.2.7 -> *]
- labeling       [0.4.3 -> *]
- lattice        [0.22-6 -> *]
- MASS           [7.3-61 -> *]
- Matrix         [1.7-1 -> *]
- mgcv           [1.9-1 -> *]
- munsell        [0.5.1 -> *]
- nlme           [3.1-166 -> *]
- RColorBrewer   [1.1-3 -> *]
- scales         [1.3.0 -> *]
- viridisLite    [0.4.2 -> *]
- cpp11          [* -> 0.5.2]
- purrr          [* -> 1.0.4]
- r2rtf          [* -> 1.1.3]
- safetyData     [* -> 1.0.0]
- stringi        [* -> 1.8.4]
- stringr        [* -> 1.5.1]
- tidyr          [* -> 1.3.1]

Do you want to proceed? [Y/n]: y

- Lockfile written to "~/renvDemoShared/renv.lock".
```

Figure 7. Creating a snapshot of your environment

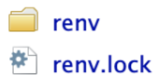
5. Share your project:

- Share your project directory (including the renv.lock file)
- Do NOT include the renv/library directory (it's automatically excluded from git)

UNDERSTANDING ENVIRONMENT SETUP IN DETAIL

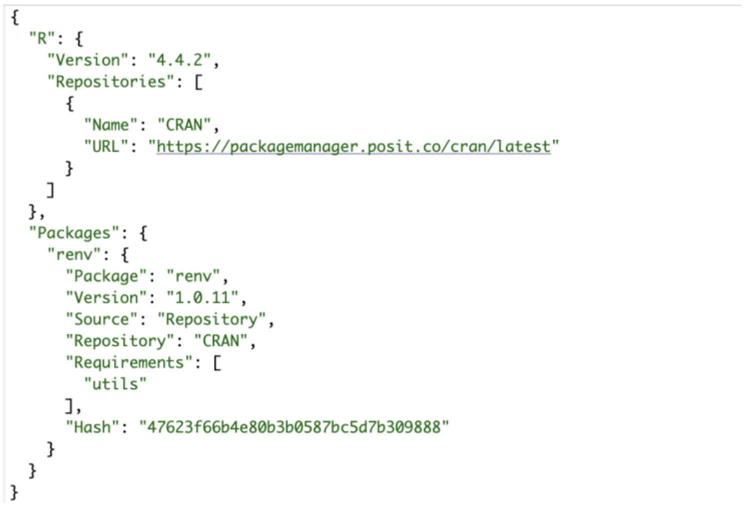
Renv initialization

Initialization sets up renv for the project. When you run renv::init(), it creates or updates several files, including renv/library, renv.lock, and .Rprofile. The lock file (renv.lock) is a JSON file detailing information about your environment, such as the R version, package sources, and package names and versions. The .Rprofile file runs automatically whenever you open the project, restoring the project environment, including library paths and renv settings. All installed packages are stored in renv/library, keeping them separate from other projects. These files work together to create an isolated and reproducible package environment.



renv
renv.lock

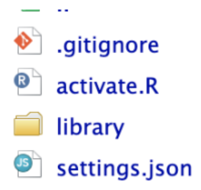
Figure 8. Files generated by renv::init()



```
{
  "R": {
    "Version": "4.4.2",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://packagemanager.posit.co/cran/latest"
      }
    ]
  },
  "Packages": {
    "renv": {
      "Package": "renv",
      "Version": "1.0.11",
      "Source": "Repository",
      "Repository": "CRAN",
      "Requirements": [
        "utils"
      ],
      "Hash": "47623f66b4e80b3b0587bc5d7b309888"
    }
  }
}
```

Figure 9. Default structure of renv.lock

Under the renv/ folder, renv/library is a project-specific library, it stores packages used in this project. It is machine-specific, and it can be rebuilt from lock file. You should not commit or include it when sharing. Fortunately, `renv` automatically creates a `.gitignore` file to exclude it.



```
.gitignore
activate.R
library
settings.json
```

Figure 10. Structure of the renv/ folder

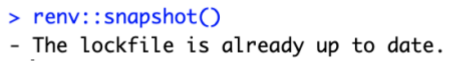
A More Efficient Way of Installing Packages

When installing packages with renv, they're placed in your project-specific library rather than your global R library. A significant advantage of renv is its global package cache. Instead of downloading the same package multiple times for different projects, renv stores a single copy in a central location and creates links to it from each project. This saves disk space and speeds up environment restoration.

Freezing in Time

The renv::snapshot() command is what makes reproducibility possible. Running renv::snapshot() will write the current state of your project's packages to renv.lock, essentially freezing your environment to that exact moment in time!

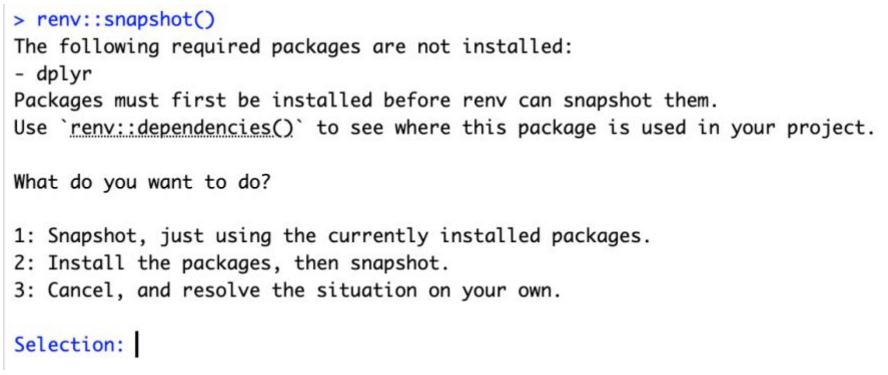
By default, renv::snapshot() uses "implicit" mode, it would scan your codebase and only include the packages that are actively used by your project. If you installed a package but never used it in your code, renv will not add it to the lock file. This behavior is also a benefit of using renv. When you run renv::init(), renv scans your project's R scripts (.R, .Rmd, etc.) looking specifically for package-loading functions such as library(), require(), and direct namespace references (pkg::function). It compiles a list of explicitly used packages as your project's dependencies. If your code conditionally loads packages, renv might not detect them, so it's important to explicitly reference packages you intend to manage.



```
> renv::snapshot()
- The lockfile is already up to date.
```

Figure 11. Snapshot behavior with unused packages

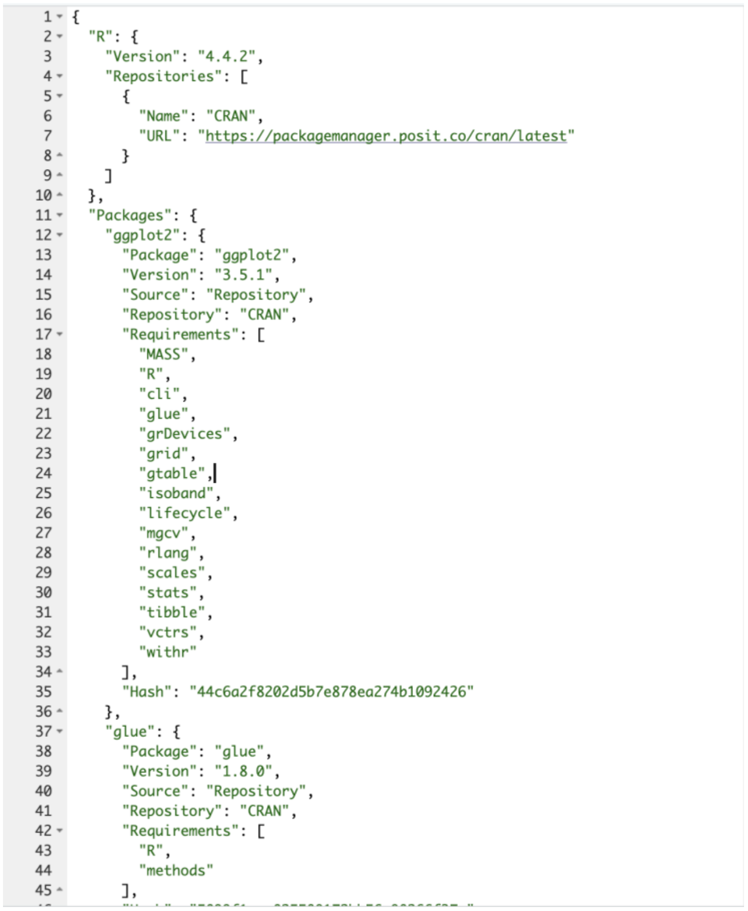
If your code references packages that aren't installed, `renv` will detect this discrepancy during snapshot, and give you three options and you can choose one of them:



```
> renv::snapshot()
The following required packages are not installed:
- dplyr
Packages must first be installed before renv can snapshot them.
Use `renv::dependencies()` to see where this package is used in your project.

What do you want to do?

1: Snapshot, just using the currently installed packages.
2: Install the packages, then snapshot.
3: Cancel, and resolve the situation on your own.

Selection: 
```

Figure 12. Snapshot identifying missing required packages

After installing new packages for your clinical trial reporting workflow, they're automatically added to the lockfile:



```
{
  "R": {
    "Version": "4.4.2",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://packagemanager.posit.co/cran/latest"
      }
    ]
  },
  "Packages": {
    "ggplot2": {
      "Package": "ggplot2",
      "Version": "3.5.1",
      "Source": "Repository",
      "Repository": "CRAN",
      "Requirements": [
        "MASS",
        "R",
        "cli",
        "glue",
        "grDevices",
        "grid",
        "gtable",
        "isoband",
        "lifecycle",
        "mgcv",
        "rlang",
        "scales",
        "stats",
        "tibble",
        "vctrs",
        "withr"
      ],
      "Hash": "44c6a2f8202d5b7e878ea274b1092426"
    },
    "glue": {
      "Package": "glue",
      "Version": "1.8.0",
      "Source": "Repository",
      "Repository": "CRAN",
      "Requirements": [
        "R",
        "methods"
      ],
```

Figure 13. Updated renv.lock after package installation

REPRODUCING AN ENVIRONMENT (FOR ENVIRONMENT CONSUMERS)

QUICK START: RESTORING A PROJECT ENVIRONMENT

1.Open the shared project:

- Open the R project in RStudio
- If you see a prompt asking to activate renv, click “Yes”

2.Restore the environment:

- Run renv::restore()
- Wait for all packages to be installed
- If prompted about package conflicts, choose the version in the lock file

3. Verify the environment:

- Run renv::status() to confirm all packages match the lock file
- Check that your R version is compatible

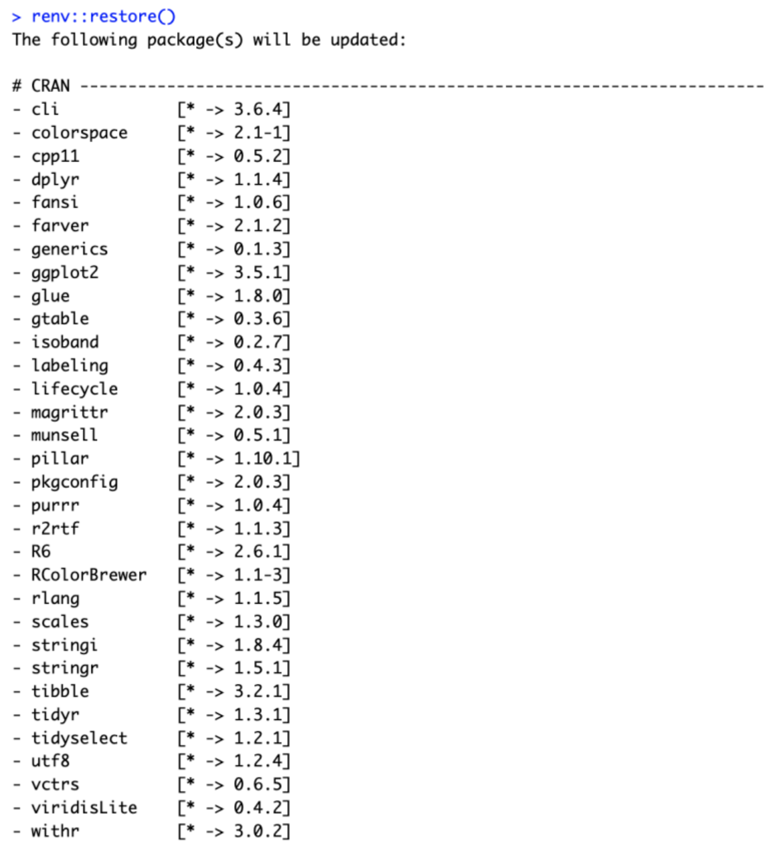
4.Run your analysis code:

- With the environment restored, your analysis code should run exactly as it did for the creator

UNDERSTANDING ENVIRONMENT RESTORATION

The restoration process

The renv::restore() command reads the renv.lock file and installs the exact package versions specified.



```
> renv::restore()
The following package(s) will be updated:

# CRAN ---------------------------------------------------------------------
- cli            [* -> 3.6.4]
- colorspace     [* -> 2.1-1]
- cpp11          [* -> 0.5.2]
- dplyr          [* -> 1.1.4]
- fansi          [* -> 1.0.6]
- farver         [* -> 2.1.2]
- generics       [* -> 0.1.3]
- ggplot2        [* -> 3.5.1]
- glue           [* -> 1.8.0]
- gtable         [* -> 0.3.6]
- isoband        [* -> 0.2.7]
- labeling       [* -> 0.4.3]
- lifecycle      [* -> 1.0.4]
- magrittr       [* -> 2.0.3]
- munsell        [* -> 0.5.1]
- pillar         [* -> 1.10.1]
- pkgconfig      [* -> 2.0.3]
- purrr          [* -> 1.0.4]
- r2rtf          [* -> 1.1.3]
- R6             [* -> 2.6.1]
- RColorBrewer   [* -> 1.1-3]
- rlang          [* -> 1.1.5]
- scales         [* -> 1.3.0]
- stringi        [* -> 1.8.4]
- stringr        [* -> 1.5.1]
- tibble         [* -> 3.2.1]
- tidyr          [* -> 1.3.1]
- tidyselect     [* -> 1.2.1]
- utf8           [* -> 1.2.4]
- vctrs          [* -> 0.6.5]
- viridisLite    [* -> 0.4.2]
- withr          [* -> 3.0.2]
```

Figure 14. renv::restore() process

During restoration, renv:

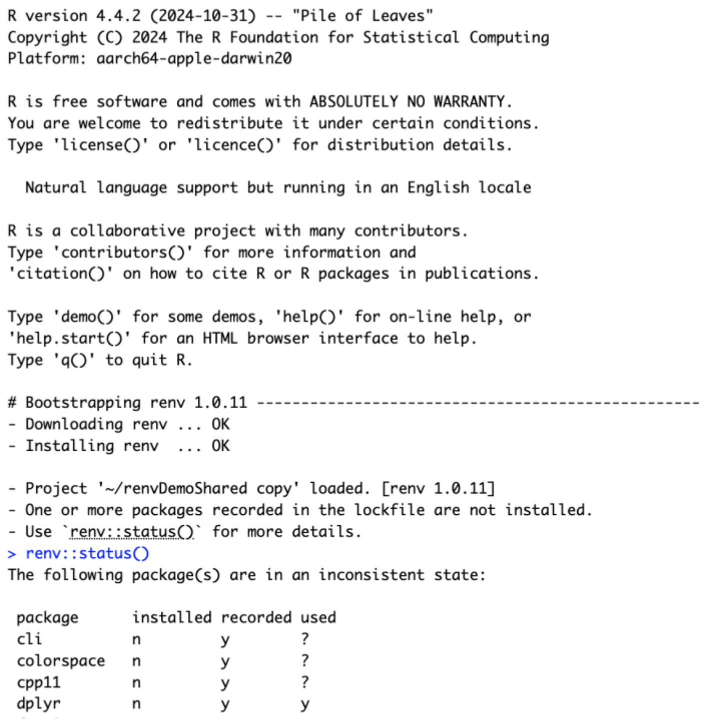
1. Reads the lock file to determine required packages
2. Checks if packages already exist in the cache
3. Downloads missing packages from the specified sources
4. Links packages from the cache to your project library

Importantly, renv performs “transactional” restores, meaning it only updates your library after all packages are successfully prepared. This prevents partial updates that could lead to inconsistent environments.

If a package is missing or a different version is detected, it installs the correct version from sources. If the package and version already exist in the renv/library, it skips reinstallation. If the package exists in the global renv cache, it links it to the project’s renv/library instead of redownloading it.

Monitoring environment status

The renv::status() command is essential for verifying that your environment matches the lock file.



```
R version 4.4.2 (2024-10-31) -- "Pile of Leaves"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

# Bootstrapping renv 1.0.11 --------------------------------------------------
- Downloading renv ... OK
- Installing renv  ... OK

- Project '~/renvDemoShared copy' loaded. [renv 1.0.11]
- One or more packages recorded in the lockfile are not installed.
- Use `renv::status()` for more details.
> renv::status()
The following package(s) are in an inconsistent state:

 package     installed recorded used
 cli         n         y        ?
 colorspace  n         y        ?
 cpp11       n         y        ?
 dplyr       n         y        y
```

Figure 15. Checking environment status with renv::status()

This output shows:

- The **"Installed"** shows whether the package is currently in their local library.
- The **"Recorded"** reflects if it is the one saved in the project's lock file (renv.lock), ensuring reproducibility and consistent package versions across teams.

The **"Used"** indicates whether a package is actually referenced in the project's code, helping to identify unnecessary dependencies.

renv::status() also alerts you if your R version differs from the one used to generate the lock file, which is crucial for complete reproducibility in clinical trial reporting.

INTEGRATION OF RENVWITH VERSION CONTROL SYSTEMS LIKE GITHUB

Integrating renv with version control systems such as GitHub significantly enhances collaboration, reproducibility, and transparency across pharmaceutical analytics teams. Leveraging GitHub alongside renv allows teams to effectively manage changes to both code and R environments, minimizing discrepancies and enhancing project consistency. Furthermore, using systems like GitHub allow for the utilization of multiple repository branches, which is supported by renv.

OPTION 1: USING GITHUB DESKTOP *(PREFERRED OPTION)*

GitHub Desktop (https://desktop.github.com/download/) is a cross-platform application designed to simplify the git process by allowing developers to contribute using a user interface (UI) instead of traditional terminal commands. This application eliminates the need to learn how to use a terminal and greatly simplifies the git syntax that developers need to learn. Thus, this is our recommended option for using version control systems like GitHub.

After downloading, open the GitHub Desktop app select an add option to create your repository:

- **Clone Repository:** if your repository is public and already uploaded to GitHub so you have access to clone information.
- **Create New Repository:** if you are starting a brand-new project.
- **Add Existing Repository:** if you have already started the project on your local computer but have not yet uploaded it to GitHub.

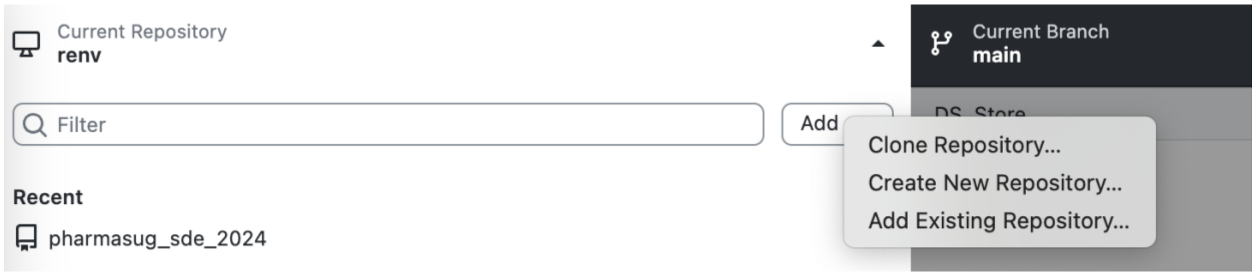


Figure 16. Options for creating repositories with GitHub Desktop.

After you have added or created your repository, you can proceed with normal scripting and coding as usual so long as it is saved in your repository somewhere. As mentioned previously, renv will create a .gitignore file automatically after initialization. Ensure this file is checked for committing, as it will exclude machine-specific directories such as renv/library. Once renv is initialized (renv::init()) and you've installed packages, run renv::snapshot() in R to save your environment.

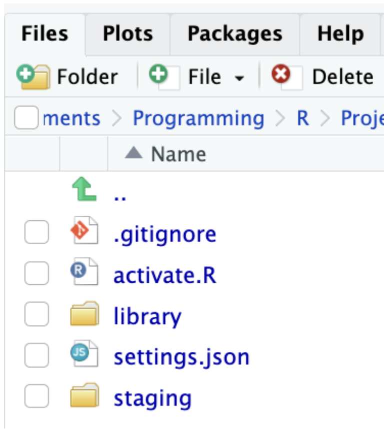
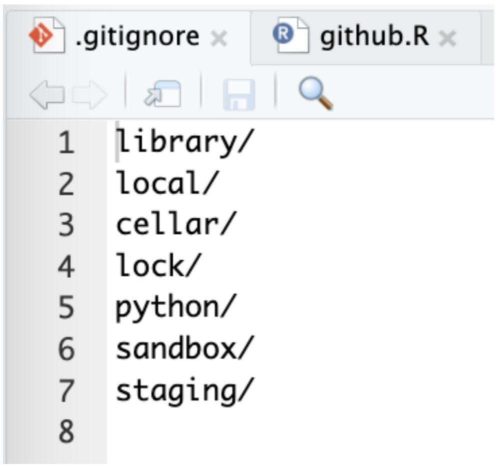


Figure 17. Generation of .gitignore.

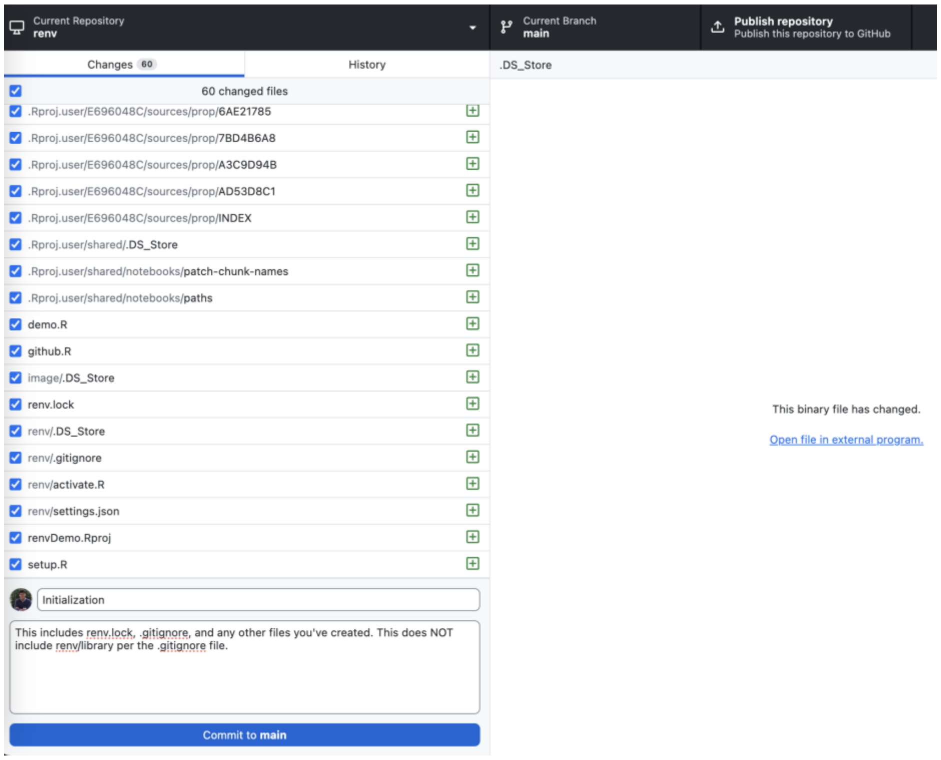


```
library/
local/
cellar/
lock/
python/
sandbox/
staging/

```

Figure 18. Contents of .gitignore.

Ensure this .gitignore file is committed to the repository to prevent unwanted files from being uploaded and always commit the renv.lock file to GitHub. This JSON file captures the exact state of your project's R environment, including R version, packages, and dependencies:

Switch to GitHub Desktop, and you will see the renv.lock file listed under changes (among anything else you have changed). Enter a meaningful commit message, such as "Commit renv.lock capturing current R environment," then click Commit. This is also where you can publish the repository to make it publicly available (and cloneable) online github.com for sharing.



.DS_Store

60 changed files

```
.Rproj.user/E696048C/sources/prop/6AE21785
.Rproj.user/E696048C/sources/prop/7BD4B6A8
.Rproj.user/E696048C/sources/prop/A3C9D94B
.Rproj.user/E696048C/sources/prop/AD53D8C1
.Rproj.user/E696048C/sources/prop/INDEX
.Rproj.user/shared/.DS_Store
.Rproj.user/shared/notebooks/patch-chunk-names
.Rproj.user/shared/notebooks/paths
demo.R
github.R
image/.DS_Store
renv.lock
renv/.DS_Store
renv/.gitignore
renv/activate.R
renv/settings.json
renvDemo.Rproj
setup.R
```

Initialization

This includes renv.lock, .gitignore, and any other files you've created. This does NOT include renv/library per the .gitignore file.

This binary file has changed.

Open file in external program.

Figure 19. The GitHub Desktop interface for committing and publishing repositories.

Whenever you install or update packages in your local repository, run renv::snapshot() in R. Switch to GitHub Desktop to commit the updated renv.lock file with a descriptive message, such as "Update renv.lock after installing new packages."

After committing changes locally, click the Push origin button in GitHub Desktop to upload your commits to GitHub (this takes the place of the publish repository button after you publish). Your collaborators can then fetch/pull these changes to their local repository to maintain consistency.

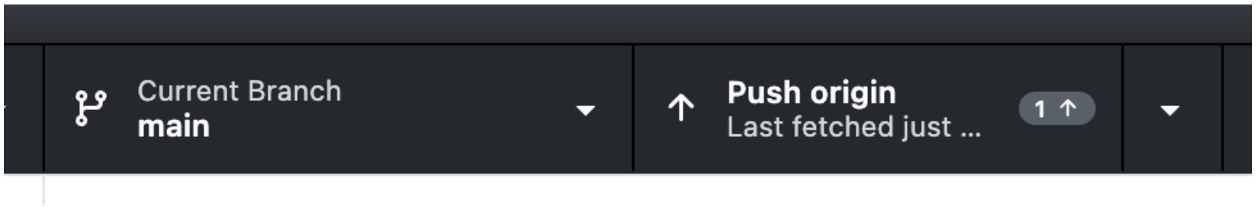


Figure 20. The GitHub Desktop interface for pushing commits.

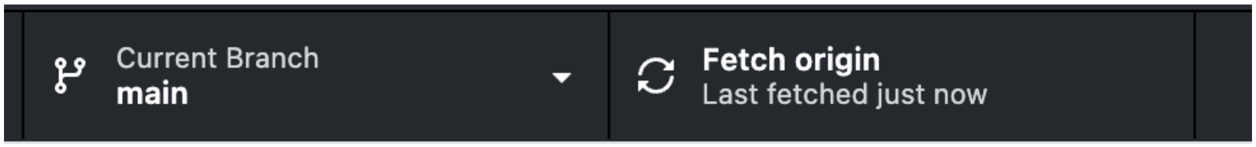
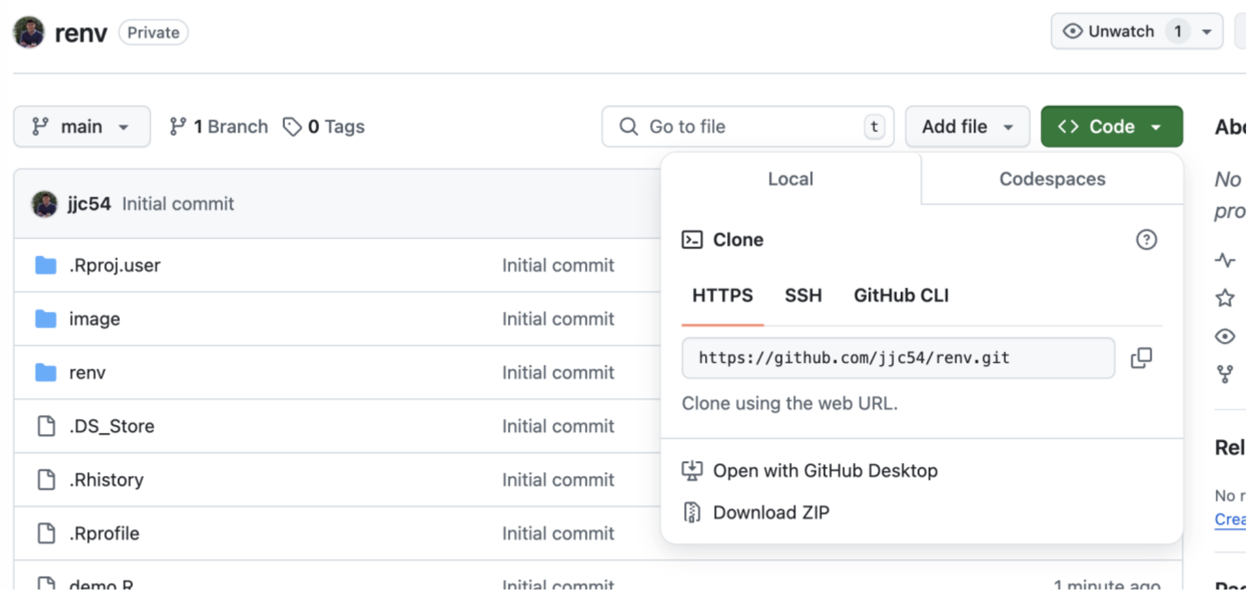


Figure 21. The GitHub Desktop interface for fetching/pulling commits.

Collaborators can then easily clone the repository using clone information from github.com through GitHub Desktop, open the project in R, and run:

```
renv::restore()
```



.Rproj.user	Initial commit
image	Initial commit
renv	Initial commit
.DS_Store	Initial commit
.Rhistory	Initial commit
.Rprofile	Initial commit

```
https://github.com/jjc54/renv.git
```

Clone using the web URL.

Figure 22. Repository cloning information from github.com.

This command restores the exact package versions defined in your committed renv.lock file, ensuring reproducibility across team members.

OPTION 2: USING TERMINAL COMMANDS

For those who are more comfortable with terminal commands, the following process can be followed. Begin by creating a new repository on GitHub or using an existing one. Initialize Git in your local project directory by entering the following terminal command:

```
git init
```

As mentioned previously, renv automatically creates a .gitignore file within your project directory to exclude unnecessary files, such as renv/library, which should not be shared as it is machine-specific:

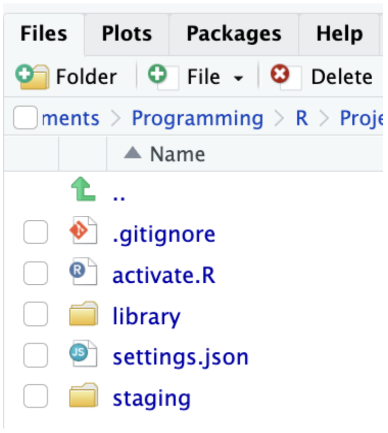
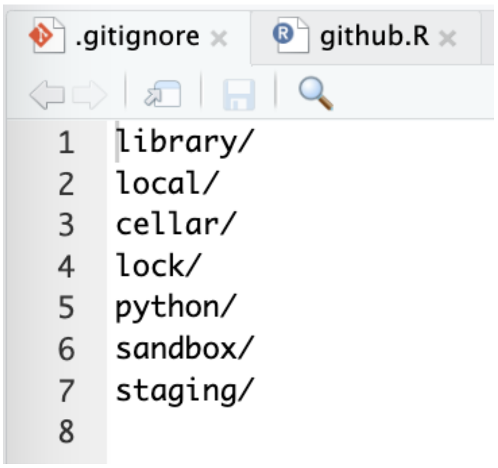


Figure 23. Generation of .gitignore.



```
library/
local/
cellar/
lock/
python/
sandbox/
staging/
```

Figure 24. Contents of .gitignore.

Ensure this .gitignore file is committed to the repository to prevent unwanted files from being uploaded and always commit the renv.lock file to GitHub. This JSON file captures the exact state of your project's R environment, including R version, packages, and dependencies:

```
git add renv.lock
git commit -m "Commit renv.lock capturing current R environment"
```

After installing or updating packages using {renv}, run renv::snapshot() to update the lock file to reflect current dependencies. Immediately commit these changes to git:

```
git add renv.lock
git commit -m "Update renv.lock after installing new packages"
```

To share changes with your team, push your commits to GitHub:

```
git remote add origin <your_repository_url>
git push -u origin main
```

Team members can replicate the exact R environment effortlessly by cloning the repository and running renvcommands:

```
git clone <repository_url>
cd <project_folder>
renv::restore()
```

This command restores the exact package versions defined in your committed renv.lock file, ensuring reproducibility across team members.

BEST PRACTICES (BOTH METHODS)

- Regularly snapshot and commit environment changes alongside code modifications.
- Clearly document package updates or environment changes in commit messages for transparency.
- Ensure the .gitignore is always respected to prevent unnecessary or sensitive files from being shared.

INTEGRATION OF RENVWITH QUARTO FOR DYNAMIC REPORTING

Integrating renv with Quarto enables clinical analytics teams to generate reproducible and dynamic reports, crucial for regulatory submissions and internal documentation. In this context, Quarto (often described as the next generation of R Markdown) emerges as a promising solution capable of bridging the gaps between different programming languages and enabling the efficient integration of diverse methodologies. By supporting extensive cross-language operations, Quarto allows teams to collaborate more effectively, streamlining the process of compiling and executing data projects. This paper aims to introduce readers to Quarto's potential to enhance interdisciplinary collaboration in data science, particularly focusing on its utility for reporting and visualization in the pharmaceutical and biotech industries. A full tutorial on Quarto is outside the scope of this paper, but supplementary readings are attached below.

Within your R project directory, create a new Quarto document (.qmd) using **RStudio®'s** interface (Figure 2).

Use renvto ensure consistent package environments for generating reports. Example packages might include r2rtf and tidyverse:

```
renv::install(c("r2rtf", "tidyverse"))
renv::snapshot()
```

Here's how your analysis can integrate into a Quarto document:

```
library(r2rtf)
library(tidyverse)

tbl <- r2rtf_adae %>%
  count(TRTA, AEDECOD) %>%
  pivot_wider(names_from = TRTA, values_from = n, values_fill = 0)

head(tbl) %>%
  rtf_title(title = "Adverse Event Count by Treatment Group") %>%
  rtf_colheader(
    colheader = "Adverse Events | Placebo | Xanomeline High Dose | Xanomeline Low Dose",
    col_rel_width = c(3, 2, 2, 2)
  ) %>%
  rtf_footnote(
    footnote = c(
      "Adverse events are coded according to MedDRA version 23.0",
      "adam-adae"
    ),
    as_table = TRUE
  ) %>%
  rtf_body(col_rel_width = c(3, 2, 2, 2)) %>%
  rtf_encode() %>%
  write_rtf("image/intro-ae9.rtf")
```

Program 2. RTF adverse event report in Quarto

Render your Quarto document using the render button within RStudio® or terminal command to produce reproducible and shareable reports:

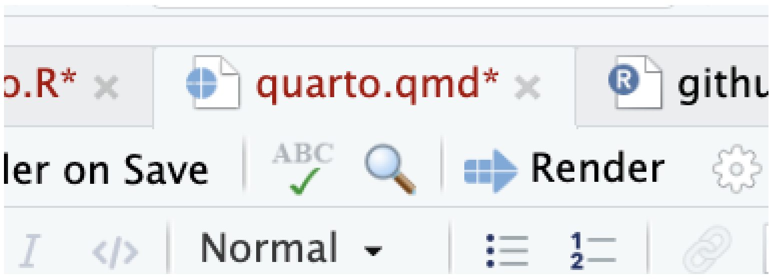
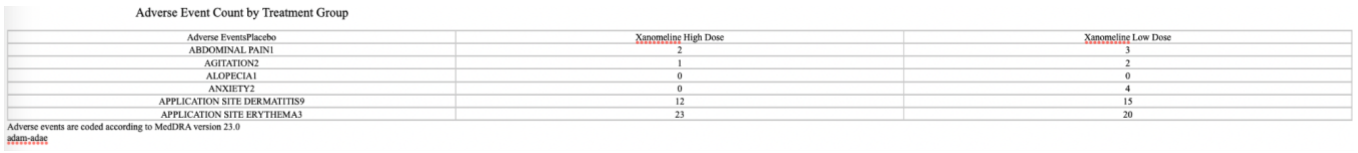


Figure 25. The render button within RStudio®.

```
quarto render your-report.qmd
```



Adverse Event Count by Treatment Group

Adverse EventsPlacebo	Xanomeline High Dose	Xanomeline Low Dose
ABDOMINAL PAIN1	2	3
AGITATION2	1	2
ALOPECIA1	0	0
ANXIETY2	0	4
APPLICATION SITE DERMATITIS9	12	15
APPLICATION SITE ERYTHEMA3	23	20

Adverse events are coded according to MedDRA version 23.0
adam-adae

Figure 26. The output.

This integrated workflow ensures robust, reproducible, and regulatory-ready reporting, facilitating collaboration and documentation within pharmaceutical analytics projects.

CONCLUSION

The renv package provides an essential foundation for managing reproducible and stable R environments across analytical workflows. When integrated with version control systems like GitHub and dynamic reporting tools like Quarto, renv significantly enhances transparency, efficiency, and reproducibility. Adopting this structured approach ensures regulatory compliance, improves team collaboration, reduces errors, and ultimately contributes to the robustness and integrity of pharmaceutical analytics projects.

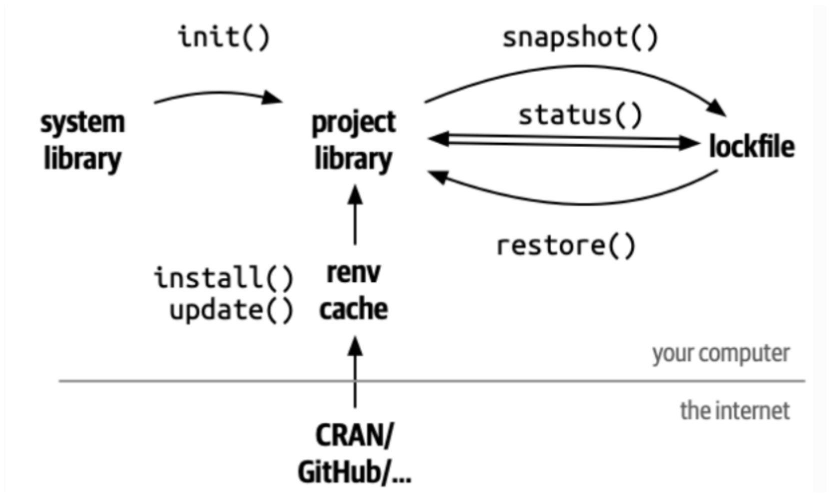


Figure 27. An overview of the entire process of using renv.

REFERENCES

Ushey, K. (2023). *Introduction to renv.* Retrieved from https://rstudio.github.io/renv/articles/renv.html

Cook, J. J. (2024). *An introduction to Quarto: A versatile open-source tool for data reporting and visualization*. PharmaSUG 2024 Conference Proceedings. Retrieved from https://pharmasug.org/proceedings/2024/DV/PharmaSUG-2024-DV-456.pdf

Zhang, Y., Wang, S., Ye, S., Kong, F., Lang, B., & Wang, B. (n.d.). “r2rtf”. Retrieved March 13, 2025, from https://merck.github.io/r2rtf/

ACKNOWLEDGMENTS

The authors thank PharmaSUG for their support of this project and for previous scholarships. Joshua J. Cook thanks his mentors, Kirk Paul Lafler (@sasNerd), and Dr. Achraf Cohen, for their ongoing mentorship and support. In addition, Junze Zhang expresses his sincere gratitude to Dr. Huei-Ling Chen for the inspiration that fueled this work. He also thanks Dr. Xiaohui Wang and Merck colleagues for their unwavering support.

RECOMMENDED READING

- *The R Project for Statistical Computing*
- *RStudio IDE*
- *Download GitHub Desktop*
- *Introduction to renv*
- *An introduction to Quarto: A Versatile Open-source Tool for Data Reporting and Visualization*
- *r2rtf – an R Package to Produce Rich Text Format (RTF) Tables and Figures*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Junze Zhang
junze.zhang@merck.com

Joshua J. Cook
jcook0312@outlook.com