Simple Process for Achieving Full Reproducibility in R: A Docker and renv Strategy

R.G. Thomas

2025-06-14

This white paper presents a comprehensive approach to achieving reproducibility in R workflows by combining two powerful tools: renv for R package management and Docker for containerizing the computing environment. Together, these tools ensure that an R workflow runs identically across different systems with the same packages, R version, and system libraries as the original setup. The paper includes a practical case study demonstrating collaborative development using this approach, with an emphasis on testing procedures to maintain code quality.

Table of contents

0.1 Executive Summary

Reproducibility is key to conducting professional data analysis, yet in practice, achieving it consistently with R workflows can be quite challenging. R projects frequently break when transferred between computers due to mismatched R versions or package dependencies. This white paper describes a comprehensive approach to solving this problem by combining two powerful tools: **renv** for R package management and **Docker** for containerizing the computing environment. Together, these tools ensure that an R workflow runs identically across different systems with the same packages, the package versions, R version, and system libraries as the original setup.

################### # edits to here 2025-05-23 08:48 # ###################

0.2 Motivation

Imagine you've written code that you want to share with a colleague. At first glance, this may seem like a straightforward task—simply send the R files via email. However, ensuring that your colleague can run the code without errors and obtain the same results is often much more challenging than anticipated.

When sharing R code, several potential problems can arise:

- Different versions of R installed on each machine
- Mismatched R package versions
- Missing system dependencies (like pandoc or LaTeX)

2

- Missing supplemental files referenced by the program (bibliography files, LaTeX preambles, datasets, images)
- Different R startup configurations (.Rprofile or .Renviron)

A real-world scenario often unfolds like this:

1. You email your R Markdown file to your colleague, Joe
2. Joe attempts to run it with `R -e "source('peng1.Rmd')"`
3. R isn't installed on Joe's system
4. After installing R, Joe gets an error: "could not find function 'render'"
5. Joe installs the rmarkdown package
6. Now pandoc is missing
7. After installing pandoc, a required package is missing
8. After installing the package, external files are missing (bibliography, images)
9. And so on…

This cycle of troubleshooting can be time-consuming and frustrating, often resulting in the "it works on my machine" syndrome. Even when the code eventually runs, there's no guarantee that Joe will get the same results that you did.

To ensure true reproducibility, your colleague should have a computing environment as similar to yours as possible. Given the dynamic nature of open source software, this can be difficult to achieve through manual installation and configuration.

The approach outlined in this white paper offers a more robust solution. Rather than sending standalone text files, you can provide a complete, containerized environment that includes everything needed to run your analysis. With this approach, your colleague can run a simple command like:

```
docker run -v "$(pwd):/home/joe" -v "$(pwd)/output:/home/joe/output" \
username/penguins-analysis
```

This creates an identical R environment on their desktop, ready for them to run or modify your code with confidence that it will work as intended.

0.3 Introduction

0.3.1 The Challenge of Reproducibility in R

R has become a standard tool for data science and statistical analysis across numerous disciplines. However, as R projects grow in complexity, they often develop intricate webs of dependencies that can make sharing and reproducing analyses difficult. Some common challenges include:

- Different R versions across machines
- Incompatible package versions
- Missing system-level dependencies
- Operating system differences (macOS vs. Windows vs. Linux)
- Conflicts with other installed packages
- R startup files (.Rprofile, .Renviron) that affect code behavior

These challenges often manifest as the frustrating "it works on my machine" problem, where analysis code runs perfectly for the original author but fails when others attempt to use it. This undermines the scientific and collaborative potential of R-based analyses.

0.3.2 A Two-Level Solution

To address these challenges comprehensively, we need to tackle reproducibility at two distinct levels:

1. **Package-level reproducibility**: Ensuring exact package versions and dependencies are maintained
2. **System-level reproducibility**: Guaranteeing consistent R versions, operating system, and system libraries

The strategy presented in this white paper leverages **renv** for package-level consistency and **Docker** for system-level consistency. When combined, they provide a robust framework for end-to-end reproducible R workflows.

0.4 renv: Package-Level Reproducibility

0.4.1 What is renv?

renv (Reproducible Environment) is an R package designed to create isolated, project-specific library environments. Instead of relying on a shared system-wide R library that might change over time, renv gives each project its own separate collection of packages with specific versions.

0.4.2 Key Features of renv

- **Isolated project library**: renv creates a project-specific library (typically in `renv/library`) containing only the packages used by that project. This isolation ensures that updates or changes to packages in one project won't affect others.

4

- **Lockfile for dependencies**: When you finish installing or updating packages, `renv::snapshot()` produces a `renv.lock` file - a JSON document listing each package and its exact version and source. This lockfile is designed to be committed to version control and shared.

- **Environment restoration**: On a new machine (or when reproducing past results), `renv::restore()` installs the exact versions of packages specified in the lockfile. This creates an R package environment identical to the one that created the lockfile, provided the same R version is available.

0.4.3 Basic renv Workflow

The typical workflow with renv involves:

```
# One-time installation of renv
install.packages("renv")

# Initialize renv for the project
renv::init()  # Creates renv infrastructure

# Install project-specific packages
# ...

# Save the package state to renv.lock
renv::snapshot()

# Later or on another system...
renv::restore()  # Restore packages from renv.lock
```

While renv effectively handles package dependencies, it does not address differences in R versions or system libraries. This limitation is where Docker becomes essential.

0.5 Docker: System-Level Reproducibility

0.5.1 What is Docker?

Docker is a platform that allows you to package software into standardized units called containers. A Docker container is like a lightweight virtual machine that includes everything needed to run an application: the code, runtime, system tools, libraries, and settings.

0.5.2 Docker's Role in Reproducibility

While renv handles R packages, Docker ensures consistency for:

- **Operating system**: The specific Linux distribution or OS version
- **R interpreter**: The exact R version
- **System libraries**: Required C/C++ libraries and other dependencies
- **Computational environment**: Memory limits, CPU configuration, etc.
- **External tools**: pandoc, LaTeX, and other utilities needed for R Markdown

By running an R Markdown project in Docker, you eliminate differences in OS or R installation as potential sources of irreproducibility. Any machine running Docker will execute the container in an identical environment.

0.5.3 Docker Components for R Workflows

For R-based projects, a typical Docker approach involves:

1. **Base image**: Starting from a pre-configured R image (e.g., from the Rocker project)
2. **Dependencies**: Adding system and R package dependencies
3. **Configuration**: Setting working directories and environment variables
4. **Content**: Adding project files
5. **Execution**: Defining how the project should run

A simple Dockerfile for an R Markdown project might look like:

```
# Use R 4.1.0 on Linux as base image
FROM rocker/r-ver:4.1.0

# Set the working directory inside the container
WORKDIR /workspace

# Install renv and restore dependencies
RUN R -e "install.packages('renv', repos='https://cloud.r-project.org')"

# Copy renv lockfile and infrastructure
COPY renv.lock renv/activate.R /workspace/

# Restore the R package environment
RUN R -e "renv::restore()"

# Default command when container runs
CMD ["/bin/bash"]
```

A more comprehensive Dockerfile that includes additional tools and user setup might look like:

```
FROM rocker/r-devel
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get clean
RUN apt-get install pandoc vim git sudo -y
RUN Rscript -e 'install.packages("renv")'
COPY renv.lock renv.lock
RUN Rscript -e 'renv::restore()'
RUN groupadd --system joe
RUN useradd --system --gid joe -m joe
RUN usermod -aG sudo joe
RUN chown joe:joe -R /home/joe
RUN chown joe:joe -R /usr/local/lib/R/site-library
WORKDIR /home/joe/
RUN mkdir -p /home/joe/output
USER joe
CMD ["/bin/bash"]
```

This Dockerfile creates a consistent environment with a specific R version and packages, regardless of the host system.

0.6 Combining renv and Docker: A Comprehensive Approach

0.6.1 Why Use Both?

Using renv or Docker alone improves reproducibility, but combining them provides the most comprehensive solution:

- **Docker** guarantees the OS and R version
- **renv** guarantees the R packages and their versions
- **Together** they achieve end-to-end reproducibility from operating system to package dependencies

This combined approach creates a fully portable analytical environment that can be shared and will produce identical results across different computers.

0.6.2 Integration Strategy

The recommended workflow integrates renv and Docker in the following manner:

1. **Develop locally with renv**: Create your R project with renv to manage package dependencies.
2. **Snapshot dependencies**: Use `renv::snapshot()` to create a lockfile.
3. **Containerize with Docker**: Create a Dockerfile that uses a specific R version and incorporates the renv lockfile.
4. **Share both**: Distribute both the code (with lockfile) and the Docker configuration.
5. **Execute consistently**: Run analyses in the Docker container for guaranteed reproducibility.

This strategy ensures that your R Markdown documents and analyses will run identically for anyone who has access to your Docker container, regardless of their local setup.

0.7 Practical Example: Collaborative R Markdown Development with Testing

The following case study demonstrates how two developers can collaborate on an R Markdown project using renv and Docker to ensure reproducibility, with integrated testing procedures to maintain code quality.

0.7.1 Project Scenario

Two data scientists are collaborating on an analysis of the Palmer Penguins dataset. Developer 1 will set up the initial project structure and create a basic analysis. Developer 2 will extend the analysis with additional visualizations. They'll use GitHub for version control and DockerHub to share the containerized environment. Both will implement testing to ensure code quality before merging changes.

0.7.2 Step-by-Step Implementation

0.7.2.1 Developer 1: Project Setup and Initial Analysis

Step 1: Create and Initialize the GitHub Repository

Developer 1 creates a new GitHub repository called "penguins-analysis" and clones it locally:

```
git clone https://github.com/username/penguins-analysis.git
cd penguins-analysis
```

Step 2: Initialize renv for Dependency Management

```r
install.packages("renv")  # If not already installed
renv::init()  # Initialize renv for the project
```

This creates the necessary renv infrastructure, including an initial `renv.lock` file.

Step 3: Install Required R Packages

```r
install.packages("ggplot2")
install.packages("palmerpenguins")
install.packages("testthat")  # For testing
install.packages("devtools")  # For development utilities
renv::snapshot()  # Save package versions to renv.lock
```

Step 4: Create Initial R Markdown Analysis

Developer 1 creates a file named `peng1.Rmd` with the following content:

```
---
title: "Palmer Penguins Analysis"
author: "Developer 1"
date: "2025-06-14"
output: html_document
---



# Flipper Length vs. Bill Length

::: {.cell}

```{.r .cell-code}
ggplot(palmerpenguins::penguins, aes(x = flipper_length_mm, y = bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Flipper Length vs. Bill Length")
```

::: {.cell-output .cell-output-stderr}

```

Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```

```
```

:::

::: {.cell-output-display}
{fig-pos='H'}
:::
:::
```

Step 5: Create Tests for Analysis Functions

Developer 1 creates a test directory structure and initial tests:

```
mkdir -p tests/testthat
```

Then creates a file `tests/testthat.R`:

```
library(testthat)
library(palmerpenguins)

test_check("penguinanalysis")
```

And a test file `tests/testthat/test-data-integrity.R`:

```
context("Data Integrity")

test_that("penguins data is available and has expected dimensions", {
  expect_true(exists("penguins", where = "package:palmerpenguins"))
  expect_equal(ncol(palmerpenguins::penguins), 8)
  expect_gt(nrow(palmerpenguins::penguins), 300)
})

test_that("penguins data has required columns", {
  expect_true("species" %in% names(palmerpenguins::penguins))
  expect_true("bill_length_mm" %in% names(palmerpenguins::penguins))
  expect_true("flipper_length_mm" %in% names(palmerpenguins::penguins))
  expect_true("body_mass_g" %in% names(palmerpenguins::penguins))
})
```

Step 6: Create a Dockerfile

Developer 1 creates a Dockerfile that deliberately excludes the R Markdown file to ensure that Developer 2's local files are used when running the container:

```
# Use R 4.1.0 as base image
FROM rocker/r-ver:4.1.0

# Set the working directory inside the container
WORKDIR /workspace

# Install renv and restore dependencies
RUN R -e "install.packages('renv', repos='https://cloud.r-project.org')"

# Copy only the renv.lock and renv infrastructure
COPY renv.lock renv/activate.R /workspace/

# Restore the R package environment
RUN R -e "renv::restore()"

# Create test directory
RUN mkdir -p tests/testthat

CMD ["/bin/bash"]
```

Step 7: Build and Push the Docker Image

```
docker build -t username/penguins-analysis:v1 .
docker login
docker push username/penguins-analysis:v1
```

Step 8: Run tests before committing

Developer 1 runs the tests to make sure everything is working correctly:

```
R -e "devtools::test()"
```

Step 9: Commit and Push to GitHub

After confirming the tests pass, Developer 1 commits the project files:

```
git add .
git commit -m "Initial renv setup, Docker environment, and tests"
git push origin main
```

Step 10: Communicate with Developer 2

Developer 1 provides these instructions to Developer 2:

1. Clone the GitHub repository
2. Pull the prebuilt Docker image from DockerHub
3. Run the container interactively, mounting the local repository
4. Create a new branch for feature development
5. Extend the analysis in the peng1.Rmd file
6. Write tests for new functionality
7. Run tests to verify changes
8. Push changes back to GitHub
9. Create a pull request

0.7.2.2 Developer 2: Extending the Analysis

Step 1: Clone the Repository and Pull the Docker Image

```
git clone https://github.com/username/penguins-analysis.git
cd penguins-analysis
docker pull username/penguins-analysis:v1
```

Step 2: Create a Feature Branch

```
git branch body-mass-analysis
git checkout body-mass-analysis
```
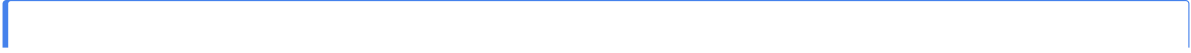
Step 3: Run Docker Interactively

Developer 2 runs the container with the local repository mounted:

```
docker run --rm -it -v "$(pwd):/workspace" -w /workspace username/penguins-analysis:v1 /bin/l
```

This approach: - Uses the renv-restored environment from the container - Allows Developer 2 to access and modify files directly from their local machine

Step 4: Extend the Analysis

Developer 2 modifies `peng1.Rmd` to add a second plot for body mass vs. bill length:

```
---
title: "Palmer Penguins Analysis"
author: "Developer 2"
date: "2025-06-14"
output: html_document
---



## Flipper Length vs. Bill Length

::: {.cell}

```{.r .cell-code}
ggplot(palmerpenguins::penguins,
       aes(x = flipper_length_mm, y = bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Flipper Length vs. Bill Length")
```

::: {.cell-output .cell-output-stderr}

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```


:::

::: {.cell-output-display}
{fig-pos='H'}
:::
:::

Body Mass vs. Bill Length

::: {.cell}

```{.r .cell-code}
ggplot(palmerpenguins::penguins,
       aes(x = body_mass_g, y = bill_length_mm)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Body Mass vs. Bill Length")
```
```

14

```
::: {.cell-output .cell-output-stderr}

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom point()`).
```

```r
context("Body Mass Analysis")

test_that("body mass data is valid", {
  expect_true(all(palmerpenguins::penguins$body_mass_g > 0, na.rm = TRUE))
  expect_true(is.numeric(palmerpenguins::penguins$body_mass_g))
})

test_that("body mass correlates with bill length", {
  # Calculate correlation coefficient
  correlation <- cor(
    palmerpenguins::penguins$body_mass_g,
    palmerpenguins::penguins$bill_length_mm,
    use = "complete.obs"
  )

  # Verify correlation is a numeric value (not NA)
  expect_true(!is.na(correlation))

  # Test that the correlation is positive
  expect_true(correlation > 0)
})
```

```r
R -e "devtools::test()"
```

```
git add peng1.Rmd tests/testthat/test-body-mass-analysis.R
git commit -m "Added body mass vs. bill length analysis with tests"
git push origin body-mass-analysis
```

```
git checkout main
git pull
docker run --rm -v "$(pwd):/workspace" -w /workspace username/penguins-analysis:v1 R -e "dev
```

0.7.3 Continuous Integration Extension

```yaml
# .github/workflows/r-test.yml
name: R Tests

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    container:
      image: username/penguins-analysis:v1

    steps:
    - uses: actions/checkout@v2

    - name: Run tests
      run: |
        R -e "devtools::test()"
```

0.7.4 Key Benefits Demonstrated in This Example

1.

2.

3.

4.

5.
6.
7.

0.8 Best Practices and Considerations

0.8.1 When to Use This Approach

-
-
-
-
-
-

0.8.2 Tips for Efficient Implementation

1.
2.

3.
4.
5.
6.
7.

0.8.3 Testing Strategies for R Analyses

1.
2.

3.

4.
5.
6.

0.8.4 Potential Challenges

-
-
-
-

-

0.9 Conclusion

0.10 References

1.
2.
3.
4.
5.
6.

0.11 Prerequisites

0.12 Step-by-Step Implementation

0.13 Key Takeaways

0.14 Further Reading