Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

Reproducible development environments with rix

Bruno Rodrigues
Ministry of Research and Higher education, Luxembourg

Philipp Baumann
Plus Affiliation

Abstract

In order to create an analysis that is easily reproducible, it is not enough to write clean code and document it well. One must also make sure to list all the dependencies of the analysis clearly and ideally provide an easy way to install said dependencies. There are several tools that can be used to list dependencies and to make them easily installable by someone that wishes to reproduce a study, such as Docker, a containerization solution. This paper will present the Nix package manager, and an R package called **rix** that lowers Nix's learning curve for users of the R programming language.

Keywords: reproducibility, R, Nix.

1. Introduction: Reproducibility is also about software

Peng (2011) introduced the idea of reproducibility being on a continuum: on one of the ends of this continuum, we only have access to the paper describing the studies, which is not reproducible at all. Then, if in addition, to this paper we make the original source code of the analysis that was written to compute the results of the study available, reproducibility is improved, albeit only by a little. Adding the original data improves reproducibility yet again. Finally, if to all this we add what Roger Peng named the *linked and executable code and data*, we reach the gold standard of full replication.

What is this *linked and executable code and data*? Another way to name this crucial piece of the reproducibility puzzle is *computational environment*. The computational environment is all the software required to actually run the analysis. Here too, we can speak of a continuum. One could simply name and list the software used: for example, the R programming language. Sometimes, authors have the courtesy to also state the version of R used. Some authors go further, and also list the packages used, and ideally with their versions as well. Authors rarely state the operating system on which the analysis was done, even though it has been shown that running the same analysis with the same software but on different operating systems could lead to different results, as described in Bhandari Neupane, Neupane, Luo, Yoshida, Sun, and Williams (2019). Authors also only very rarely provide instructions to install the required tools and software in order to reproduce their studies.

Tools can be used at each of these steps to reach the gold standard of full replication. Let's first consider the task of listing the software used. R provides the `sessionInfo()` function whose output can be saved into a file. Below is an example output of `sessionInfo()`:

```
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: aarch64-unknown-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

Matrix products: default
BLAS:   /usr/lib/aarch64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/aarch64-linux-gnu/openblas-pthread/libopenblasp[...]

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

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

other attached packages:
[1] nnet_7.3-19  mgcv_1.9-0   nlme_3.1-163

loaded via a namespace (and not attached):
[1] compiler_4.3.2 Matrix_1.6-1.1 tools_4.3.2    splines_4.3.2
[5] grid_4.3.2     lattice_0.21-9
```

If an author provides this information, other people trying to reproduce the study (or the author him- or herself in the future) can read this file and see which version of R was used, and which packages (and their versions) were used as well. However, others would still need to install the correct software themselves. An alternative to this is instead to use the **renv** package which generates a so-called `renv.lock` file which also lists R and package versions. Here is an example of such an `renv.lock` file:

```
{
"R": {
  "Version": "4.2.2",
  "Repositories": [
  {
   "Name": "CRAN",
   "URL": "https://packagemanager.rstudio.com/all/latest"
  }
  ]
},
"Packages": {
  "MASS": {
    "Package": "MASS",
    "Version": "7.3-58.1",
    "Source": "Repository",
    "Repository": "CRAN",
    "Hash": "762e1804143a332333c054759f89a706",
    "Requirements": []
  },
  "Matrix": {
    "Package": "Matrix",
    "Version": "1.5-1",
    "Source": "Repository",
    "Repository": "CRAN",
    "Hash": "539dc0c0c05636812f1080f473d2c177",
    "Requirements": [
      "lattice"
    ]

    ***and many more packages***
```

This file lists every package alongside their versions and the repository from which they were downloaded. Generating this file only requires one to run the `renv::init()` function. Someone else can then restore the same package library by running `renv::restore()`. The exact same packages get installed in an isolated, project-specific, library which doesn't interfere with the other, main, library of the user. **renv** does not restore the R itself though, so installing the right version of R needs to be handled separately. Before continuing, it should be noted that other packages exist which provide similar functionality to **renv**: there is **groundhog** by Simonsohn and Gruson (2023) which makes it rather easy to install packages as they were on CRAN at a given date. For example, the code snippet below install the **purrr** and **ggplot2** packages as they were on April 4th, 2017:

```
groundhog.library("
    library(purrr)
    library(ggplot2)",
    "2017-10-04",
    tolerate.R.version = "4.2.2")
```

These packages also get installed in a project-specfic library so there is no interferences between these packages and other versions of the same packages that one might use for other projects. Because **groundhog** does not install R itself, users should either install the required version themselves, or they should use the `tolerate.R.version` argument as shown in the example above. Otherwise, **groundhog** would not continue with the installation of the packages. Another such package, developed by hong Chan and Schoch (2023) is **rang**, which also installs packages as they were on a given date. Yet another way to install packages as they were on a give date is to use the Posit Package Manager, which provides snapshots of CRAN. For example, to install the required packages for an analysis as they were on the 30th of June 2023, one could add the following line to the `.Rprofile` file:

```
options(repos = c(REPO_NAME = "https://packagemanager.posit.co/cran/
                                __linux__/jammy/2023-06-30"))
```

The `.Rprofile` file gets read by R when starting a new session, which means that every call to the `install.packages()` function will now install the packages from this snapshotted mirror.

The next step in reaching the gold standard of reproducibility would be to not only install the right packages used for the analysis, but also the right version of R. Of course, it would be possible to install the right version manually, but here too, there are tools that simplify the process such as rig by the R infrastructure team (2023). One could thus install the right version of R using rig allows to easily install different versions of R, one could use to install the right version to reproduce an analysis, and then use one of the listed packages above to install the right library of of R packages. However, this involves many manual steps and is thus error prone. It is also time-consuming to do so, especially if one wants to reproduce many studies.

The final step towards the gold standard of reproducibility is to package the right version of R and R packages inside a Docker image. Docker is a containerisation tool: using Docker it is possible to package some *data product* with its dependencies into an image. From this image, containers can be run to reproduce the data product with its exact dependencies. A statistical analysis, from the simplest to the most complex one, can be seen as such a data product that comes with many software dependencies. *Dockerizing* an analysis consists in first building an image: in the build step, the dependencies of the analysis have to be installed and this can be achieved using the tools mentioned above. The scripts to run the analysis and the data are also added to the image at build time. Instances of that image, called containers, can then be executed, which give access to the exact development environment originally used for the study. It is also possible to script the building of the analysis itself, such that in one single call to the `docker run` command, a complete analysis is reproduced.

The Rocker project initiated by Boettiger and Eddelbuettel (2017) provides many pre-built Docker to the R community of users. These images can be used as bases to build other images containing statistical analyses more easily than starting from a bare-bones image.

An optional step towards the gold standard is to use a build automation tool such as Make to run the whole analysis when the container is executed. Build automation tools make it easier to run arbitrary code in a series of well-defined steps. R programmers can use the **targets** by Landau (2021) as a build automation tool.

McDermott (2021) is an example of a scientific study that reached the gold standard of reproducibility. The author of this paper set up an accompagnying Github repository to the paper[1] containing all the instructions to install the required software and then run the analysis. If we take a closer look at this repository, we will notice that many of the tools previously mentioned were used to capture the compatutational environment and make it available to other researchers:

- Packages and their versions were listed and saved into an `renv.lock` file;
- A `Makefile` was used to run the whole analysis and compile the paper;
- A `Dockerfile` was used to provide the complete computational environment, including the right version of R and run the whole analysis easily.

As we go down this list, we get closer to the gold standard of a perfectly reproducible study.

However, reaching this gold standard is quite costly: one needs to learn a tool to deal with package versions, then a build automation tool such as make and ideally Docker should be added to the list. Docker is a tool that makes it possible to run arbitrary code in a completely controlled and isolated environment as a way to capture the complete underlying system libraries. Simply put, a `Dockerfile` provides a complete description of the computational environment as well as the required instructions to build the complete project. One can then use Docker to build a so-called *image* which contains all the required software and instructions to build the project. From this image, it is then possible to run a *container* which actually runs the analysis.

1.1. Prior art

R provides a very flexible implementation of the general GLM framework in the function `glm()` **?** in the **stats** package. Its most important arguments are

```
glm(formula, data, subset, na.action, weights, offset,
  family = gaussian, start = NULL, control = glm.control(…),
  model = TRUE, y = TRUE, x = FALSE, …)
```

where `formula` plus `data` is the now standard way of specifying regression relationships in R/S introduced in **?**. The remaining arguments in the first line (`subset`, `na.action`, `weights`, and `offset`) are also standard for setting up formula-based regression models in R/S. The arguments in the second line control aspects specific to GLMs while the arguments in the last line specify which components are returned in the fitted model object (of class '`glm`' which inherits from '`lm`'). For further arguments to `glm()` (including alternative specifications of starting values) see `?glm`. For estimating a Poisson model `family = poisson` has to be specified.

As the synopsis above is a code listing that is not meant to be executed, one can use either the dedicated `{Code}` environment or a simple `{verbatim}` environment for this. Again, spaces before and after should be avoided.
Finally, there might be a reference to a `{table}` such as Table 1. Usually, these are placed at the top of the page (`[t!]`), centered (`\centering`), with a caption below the table, column headers and captions in sentence style, and if possible avoiding vertical lines.

[1] https://github.com/grantmcdermott/skeptic-priors

Type	Distribution	Method	Description
GLM	Poisson	ML	Poisson regression: classical GLM, estimated by maximum likelihood (ML)
		Quasi	“Quasi-Poisson regression’ ’: same mean function, estimated by quasi-ML (QML) or equivalently generalized estimating equations (GEE), inference adjustment via estimated dispersion parameter
		Adjusted	“Adjusted Poisson regression’ ’: same mean function, estimated by QML/GEE, inference adjustment via sandwich covariances
	NB	ML	NB regression: extended GLM, estimated by ML including additional shape parameter
Zero-augmented	Poisson	ML	Zero-inflated Poisson (ZIP), hurdle Poisson
	NB	ML	Zero-inflated NB (ZINB), hurdle NB

Table 1: Overview of various count regression models. The table is usually placed at the top of the page (`[t!]`), centered (`centering`), has a caption below the table, column headers and captions are in sentence style, and if possible vertical lines should be avoided.

2. Illustrations

For a simple illustration of basic Poisson and NB count regression the `quine` data from the **MASS** package is used. This provides the number of `Days` that children were absent from school in Australia in a particular year, along with several covariates that can be employed as regressors. The data can be loaded by

```
R> data(mtcars)
```

and a basic frequency distribution of the response variable is displayed in **?@fig-quine**.

For code input and output, the style files provide dedicated environments. Either the “agnostic” `{CodeInput}` and `{CodeOutput}` can be used or, equivalently, the environments `{Sinput}` and `{Soutput}` as produced by `Sweave()` or **knitr** when using the `render_sweave()` hook. Please make sure that all code is properly spaced, e.g., using `y = a + b * x` and *not* `y=a+b*x`. Moreover, code input should use “the usual” command prompt in the respective software system. For R code, the prompt `R>` should be used with `+` as the continuation prompt. Generally, comments within the code chunks should be avoided – and made in the regular LaTeX text instead. Finally, empty lines before and after code input/output should be avoided (see above).

3. Summary and discussion

As usual...

Computational details

If necessary or useful, information about certain computational details such as version numbers, operating systems, or compilers could be included in an unnumbered section. Also, auxiliary packages (say, for visualizations, maps, tables, ...) that are not cited in the main text can be credited here.

The results in this paper were obtained using R~3.4.1 with the **MASS**~7.3.47 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at [https://CRAN.R-project.org/].

Acknowledgments

All acknowledgments (note the AE spelling) should be collected in this unnumbered section before the references. It may contain the usual information about funding and feedback from colleagues/reviewers/etc. Furthermore, information such as relative contributions of the authors may be added here (if any).

References

Bhandari Neupane J, Neupane RP, Luo Y, Yoshida WY, Sun R, Williams PG (2019). "Characterization of Leptazolines A–D, Polar Oxazolines from the Cyanobacterium Leptolyngbya sp., Reveals a Glitch with the "Willoughby–Hoye" Scripts for Calculating NMR Chemical Shifts." *Organic Letters*, **21**(20), 8449–8453.

Boettiger C, Eddelbuettel D (2017). "The R Journal: An Introduction to Rocker: Docker Containers for R." *The R Journal*, **9**, 527–536. ISSN 2073-4859. `doi:10.32614/RJ-2017-065`. Https://doi.org/10.32614/RJ-2017-065.

hong Chan C, Schoch D (2023). "rang: Reconstructing reproducible R computational environments." *PLOS ONE*. `doi:10.1371/journal.pone.0286761`. URL https://github.com/gesistsa/rang.

Infrastructure R (2023). *rig: The R Installation Manager*. URL https://github.com/r-lib/rig.

Landau WM (2021). "The targets R package: A dynamic Make-like function-oriented pipeline toolkit for reproducibility and high-performance computing." *Journal of Open Source Software*, **6**(57), 2959.

McDermott GR (2021). "Skeptic priors and climate consensus." *Climatic Change*, **166**(1-2), 7.

Peng RD (2011). "Reproducible research in computational science." *Science*, **334**(6060), 1226–1227.

Simonsohn U, Gruson H (2023). *groundhog: Version-Control for CRAN, GitHub, and GitLab Packages.* R package version 3.1.2, https://github.com/CredibilityLab/groundhog, URL https://groundhogr.com/.

More technical details

Appendices can be included after the bibliography (with a page break). Each section within the appendix should have a proper section title (rather than just *Appendix*).
For more technical style details, please check out JSS's style FAQ at [https://www.jstatsoft.org/pages/view/style#frequently-asked-questions] which includes the following topics:

- Title vs. sentence case.
- Graphics formatting.
- Naming conventions.
- Turning JSS manuscripts into R package vignettes.
- Trouble shooting.
- Many other potentially helpful details…

Using BibTeX

References need to be provided in a BibTeX file (`.bib`). All references should be made with `@cite` syntax. This commands yield different formats of author-year citations and allow to include additional details (e.g.,pages, chapters, …) in brackets. In case you are not familiar with these commands see the JSS style FAQ for details.
Cleaning up BibTeX files is a somewhat tedious task – especially when acquiring the entries automatically from mixed online sources. However, it is important that informations are complete and presented in a consistent style to avoid confusions. JSS requires the following format.

- item JSS-specific markup (`\proglang`, `\pkg`, `\code`) should be used in the references.
- item Titles should be in title case.
- item Journal titles should not be abbreviated and in title case.
- item DOIs should be included where available.
- item Software should be properly cited as well. For R packages `citation("pkgname")` typically provides a good starting point.

Affiliation:

Bruno Rodrigues
Department of Statistics
18, Montée de la Pétrusse
Luxembourg Luxembourg
E-mail: `bruno@brodrigues.co`
URL: https://www.brodrigues.co

Philipp Baumann

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
`doi:10.18637/jss.v000.i00` *Accepted:* yyyy-mm-dd