RANG: RECONSTRUCTING REPRODUCIBLE R COMPUTATIONAL ENVIRONMENTS

A PREPRINT

Chung-hong Chan

GESIS Leibniz-Institut für Sozialwissenschaften

David Schoch

GESIS Leibniz-Institut für Sozialwissenschaften

March 7, 2023

ABSTRACT

A complete declarative description of the computational environment is often missing when researchers share their materials. Without such description, software obsolescence and missing system components can jeopardize computational reproducibility in the future, even when data and computer code are available. The R package rang is a complete solution for generating the declarative description for other researchers to automatically reconstruct the computational environment at a specific time point. The reconstruction process, based on Docker, has been tested for R code as old as 2005 and does not depend on the long term availability of any commercial service. The description generated by rang satisfies the definition of a reproducible research compendium and can be shared as such. In this contribution, we show how rang can be used to make otherwise unexecutable code, spanning from fields such as computational social science and bioinformatics, executable again. We also provide instructions on how to use rang to construct reproducible and shareable research compendium of current research. The package is currently available from CRAN (https://cran.r-project.org/web/packages/rang/index.html) and GitHub (https://github.com/chainsawriot/rang).

***K*eywords** R • reproducibility • docker

1 Background

"In some cases the polarization estimation will not work … This is NOT *a problem in the method, it is entirely dependent on the numpy version (and even the OS's). If you have different versions of numpy or even the same version of numpy on a different OS configuration, different networks will fail randomly. … [F]or instance, the 109th Congress will fail, but will work entirely normally on a different numpy version, which will fail on a different Congress network."*

~ excerpt of this README file

Other than bad programming practices (Trisovic et al. 2022), the main computing barrier to computational reproducibility is the failure to reconstruct the computational environment like the one used by the original researchers. This task looks trivially simple. But as computer science research has shown, this task is incredibly complex (Abate et al. 2015; Dolstra, Löh, and Pierron 2010). In the realm of R, that includes at least four aspects: a) operating system, b) system components such as `libxml2`, c) the exact R version, and d) what and which version of the installed R

packages. We will call them component a, b, c, d in the following sections. Any change in these four components can possibly affect the execution of any shared computer code. For example, the lack of the system component `libxml2` can impact whether the R package `xml2` can be installed on a Linux system. If the shared computer code requires the R package `xml2`, then the whole execution fails.

In reality, the impact of operating system is relative weak as mainstream, open source programming languages and their software libraries are usually cross platform. In modern computational research, Linux is the de-facto operating system in high performance computing environments. Instead, the impact of components b, c, and d is practically higher. Component d is probably the most volatile among them all. Software update with breaking changes might render existing shared code not executable anymore. Also, software obsolescence is a common place, especially when academic software is often not well maintained (Merow et al. 2023).

The DevOps (software development and IT operations) community has come up with a partial solution to this problem: containerization. In essence, to containerize is to deploy the software together with all the libraries and even the operating system in a virtualized environment. In this way, software dependency issues can be resolved within the tightly controlled virtualized environment and independent of what is installed on the host computer.

1.1 Existing solutions

`renv` (Ushey 2022) (and its derivatives such as `jetpack` and its predecessor `packrat`) takes a similar approach to Python's `virtualenv` and Ruby's `Gem` to pin down the exact version of R packages using a "lock file". `containerit` (Nüst and Hinz 2019) takes the current state of the computational environment and "containerize" it as a Dockerfile. But `containerit` does not pin down the exact version of R packages. `checkpoint` (Ooi, de Vries, and Microsoft 2022) depends on the availability of The Microsoft R Application Network (MRAN), which will be shut down on July 1st, 2023. `groundhog` (Simonsohn and Gruson 2023) used to be depending on MRAN but with a plan to switch to their home-grown R package repository.

These solutions are better for prospective usage, i.e. using them now to ensure the reproducibility of the current research for future researchers. `rang` mostly targets retrospective usage, i.e. using `rang` to reconstruct historical R computational environments which have not been completely declared. One can think of `rang` as an archaeological tool. In Section 3, `rang` is used to enable the reproducibility of published literature. However, one can still use `rang` for prospective usage and arguably with a longer term computational reproducibility than other solutions. In Section 4, `rang` is used to create an executable research compendium.

2 Basic usage

There are two important functions of `rang`: `resolve()` and `dockerize()`.

`resolve()` queries various web services from the r-hub project of the R Consortium for information about R packages at a specific time point that is necessary for reconstructing a computational environment, e.g. (deep) dependencies, system requirements, and R version. For instance, if there was a computational environment constructed on 2020-01-16 (called "snapshot date") with the several natural language processing R packages, `resolve()` can be used to resolve all the dependencies of these R packages. Currently, `rang` supports CRAN, Bioconductor, GitHub, and local packages.

```
library(rang)
graph <- resolve(pkgs = c("openNLP", "LDAvis", "topicmodels", "quanteda"),
                 snapshot_date = "2020-01-16")
graph
```

The resolved result is an S3 object called `rang`. The information contained in a `rang` object can then be used to construct a computational environment in a similar manner as `containerit`, but with the packages and R versions pinned on the snapshot date. Then, the function `dockerize()` is used to generate the `Dockerfile` and other scripts in the `output_dir`.

```
dockerize(graph, output_dir = "docker")
```

For R >= 3.1, the images from the Rocker project are used (Boettiger and Eddelbuettel 2017). For R < 3.1 but >= 1.3.1, a custom image based on Debian is used. As of writing, `rang` does not support R < 1.3.1, i.e. snapshot date earlier than 2001-08-31. There are two features of `dockerize()` that are important for future reproducibility.

1. By default, the container building process downloads source packages from CRAN and then compiles them from source. This step depends on the future availability of R packages on CRAN (which is extremely likely to be the case in the near future, given the continuous availability since 1997-04-23) [1]. However, it is also possible to cache (or archive) the source packages now. The archived R packages can then be used instead during the building process.

```
dockerize(graph, output_dir = "docker", cache = TRUE)
```

2. It is also possible to install R packages in a separate library during the building process to isolate all these R packages from the main library.

```
dockerize(graph, output_dir = "docker", cache = TRUE,
          lib = "anotherlibrary")
```

For the sake of completeness, the instructions for building and running the Docker container on Unix-like systems are included here.

```
cd docker
## might need to sudo
docker build -t rang .
## interactive environment
docker run --rm --name "rangtest" -ti rang
```

3 Case Studies

The following are some examples of how `rang` can be used to make otherwise shared, but unexecutable, R code runnable again. The examples were drawn from various fields spanning from political science, psychological science, and bioinformatics.

3.1 quanteda JOSS paper

The software paper of the text analysis R package `quanteda` was published on 2018-10-06 (Benoit et al. 2018). In the paper, the following R code snippet is included.

```
library("quanteda")
# construct the feature co-occurrence matrix
examplefcm <-
tokens(data_corpus_irishbudget2010, remove_punct = TRUE) %>%
tokens_tolower() %>%
tokens_remove(stopwords("english"), padding = FALSE) %>%
fcm(context = "window", window = 5, tri = FALSE)
# choose 30 most frequency features
topfeats <- names(topfeatures(examplefcm, 30))
# select the top 30 features only, plot the network
set.seed(100)
textplot_network(fcm_select(examplefcm, topfeats), min_freq = 0.8)
```

On 2023-02-08, this code snippet is not executable with the current version of `quanteda` (3.2.4). It is possible to install the "period appropriate" version of `quanteda` (1.3.4) using `remotes` on the current version of R (4.2.2). And indeed, the above code snippet can still be executed.

```
remotes::install_version("quanteda", version = "1.3.4")
```

[1] https://stat.ethz.ch/pipermail/r-announce/1997/000001.html

The issue is that installing `quanteda` 1.3.4 this way installs the latest dependencies from CRAN. `quanteda` 1.3.4 uses a deprecated (but not yet removed) function of `Matrix` (`as(<dgTMatrix>, "dgCMatrix")`). If this function were removed in the future, the above code snippet would not be executable anymore.

Using `rang`, one can query the version of `quanteda` on 2018-10-06 and create a Docker container with all the "period appropriate" dependencies. Here, the `rstudio` Rocker image is selected.

```
library(rang)
graph <- resolve(pkgs = "quanteda",
                 snapshot_date = "2018-10-06",
                 os = "ubuntu-18.04")
dockerize(graph, output_dir = "quanteda_docker",
          image = "rstudio")
```

The above code snippet can be executed with the generated container without any problem (Figure 1).

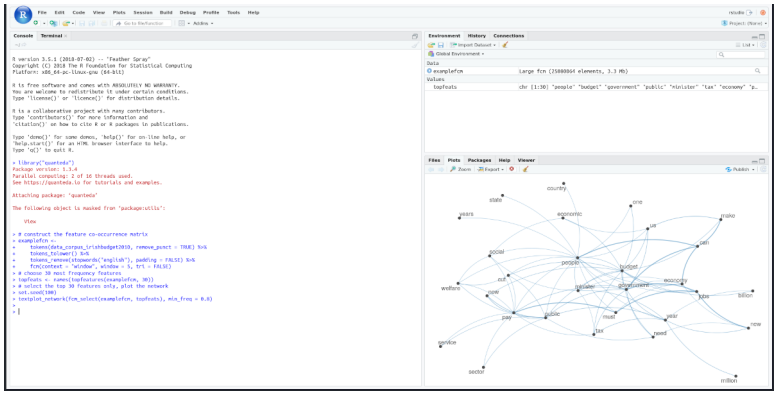

Figure 1: The code snippet running in a R 3.5.1 container created with rang

3.2 Psychological Science

Crüwell et al. (2023) evaluate the computational reproducibility of 14 articles published in *Psyhocological Science*. Among these articles, the paper by Hilgard et al. (2019) has been rated as having "package dependency issues".

All data and computer code are available from GitHub with the last commit on 2019-01-17 [2]. The R code contains a list of R packages used in the project as `library()` statements, including an R package on GitHub that is written by the main author of that paper. However, we identified one package (`compute.es`) that was not written in those `library()` statements but used with the namespace operator, i.e. `compute.es::tes()`. This undocumented package can be detected by `renv::dependencies()`, which is the provider of the scanning function of `rang`.

Based on the above information, one can run `resolve()` to obtain the dependency graph of all R packages on 2019-01-17.

```
## scan all packages
r_pkgs <- as_pkgrefs("vvg-2d4d")
## replace cran::hilgard with Github
r_pkgs[r_pkgs == "cran::hilhard"] <- "Joe-Hilgard/hilgard"
graph <- resolve(r_pkgs, snapshot_date = "2019-01-17")
```

When running `dockerize()`, one can take advantage of the `materials_dir` parameter to transfer the shared materials from Hilgard et al. (2019) into the Docker image.

```
dockerize(graph, "hilgard", materials_dir = "vvg-2d4d", cache = TRUE)
```

[2] https://github.com/Joe-Hilgard/vvg-2d4d

We then built the Docker and launch a Docker container. For this container, we changed the entry point from R to bash so that the container goes to the Linux command shell instead.

```
cd hilgard
docker build -t hilgard .
docker run --rm --name "hilgardcontainer" --entrypoint bash -ti hilgard
```

Inside the container, the materials are located in the `materials` directory. We used the following shell script to test the reproducibility of all R scripts.

```
cd materials
rfiles=(0_data_aggregation.R 1_data_cleaning.R 2_analysis.R 3_plotting.R)
for i in ${rfiles[@]}
do
    Rscript $i
    code=$?
    if [ $code != 0 ]
    then
        exit 1
    fi
done
```

All R scripts ran fine inside the container and the figures generated are the same as the ones in Hilgard et al. (2019).

3.3 Political Analysis

The study by Trisovic et al. (2022) evaluates the reproducibility of R scripts shared on Dataverse. They found that 75% of R scripts cannot be successfully executed. Among these failed R scripts is an R script shared by Beck (2019).

This R script has been "rescued" by the author of the R package `groundhog` (Simonsohn and Gruson 2023), as demonstrated in a blog post [3]. We were wondering if `rang` can also be used to "rescue" the concerned R script. The date of the R script, as indicated on Dataverse, is 2018-12-12. This date is used as the snapshot date.

```
## as_pkgrefs is automatically run in this case
graph <- resolve("nathaniel", snapshot_date = "2018-12-12")
dockerize(graph, output_dir = "nat", materials_dir = "nathaniel")
```

```
cd nat
docker build -t nat .
docker run --rm --name "natcontainer" --entrypoint bash -ti nat
```

Inside the container

```
cd materials
Rscript fn_5.R
```

The same file can also be "rescued" by `rang`.

3.4 Recover a removed R package: maxent

The R package `maxent` introduces a machine learning algorithm with a small memory footprint and was available on CRAN until 2019. A software paper was published by the original authors in 2012 (Jurka 2012). The R package was also used in some subsequent automated content analytic papers (e.g. Lörcher and Taddicken 2017). Despite the covert editing of the package by a staffer of CRAN [4], the package was removed from CRAN in 2019 [5]. We attempted

[3] http://datacolada.org/100

[4] https://github.com/cran/maxent/commit/9d46c6aad27a1f41a78907b170ddd9a586192be9

[5] https://cran-archive.r-project.org/web/checks/2019/2019-03-05_check_results_maxent.html

to install the second last (the original submitted version) and last (with covert editing) versions of `maxent` on R 4.2.2. Both of them didn't work.

Using `rang`, we are able to reconstruct a computational environment with R 2.15.0 (2012-03-30) to run all code snippets published in Jurka (2012) [6]. For removed CRAN packages, we strongly recommend querying the Github read-only mirror of CRAN instead (https://github.com/cran). It is because in this way, the resolved system requirements have a higher chance of being correct.

```
maxent <- resolve("cran/maxent", "2012-06-10")
dockerize(maxent, "maxentdir", cache = TRUE)
```

3.5 Recover a removed R package: ptproc

The software paper of the R package `ptproc` was published in 2003 and introduced multidimensional point process models (Peng 2003). But the package has been removed from CRAN for over a decade (at least). The only release on CRAN was on 2002-10-10. The package is still listed in the "Handling and Analyzing Spatio-Temporal Data" CRAN Task View [7] despite being uninstallable without modification on any modern R system (see below). As of writing, the package, as a tarball file (tar.gz), is still downloadable from the original author's website [8].

Even with this over-a-decade removal and new packages with similar functionalities have been created, there is evidence that `ptproc` is still being sought for. As late as 2017, there are blog posts on how to install the long obsolete package on modern R [9]. The package is extremely challenging to install on a modern R system because the package was written before the introduction of name space management in R 1.7.0 (Tierney 2003). In other words, the available tarball file from the original author's website does not contain a `NAMESPACE` file as all other modern R packages do.

The oldest version of R that `rang` can support, as of writing, is R 1.3.1. `rang` is probably the only solution available that can support the 1.x series of R (i.e. before 2004-10-04). Similar to the case of `maxent` above, a Dockerfile to generate a Docker image with `ptproc` installed can be generated with two lines of code.

```
graph <- resolve("ptproc", snapshot_date = "2004-07-01")
dockerize(graph, "~/dev/misc/ptproc", cache = TRUE)
```

Suppose we have an R script, extracted from Peng (2003), called "peng.R" like this:

```
require(ptproc)

set.seed(1000)
x <- cbind(runif(100), runif(100), runif(100))
hPois.cond.int <- function(params, eval.pts, pts = NULL, data = NULL, TT = NULL) {
    mu <- params[1]
    if(is.null(TT))
        rep(mu, nrow(eval.pts))
    else {
        vol <- prod(apply(TT, 2, diff))
        mu * vol
    }
}
ppm <- ptproc(pts = x, cond.int = hPois.cond.int, params = 50,
              ranges = cbind(c(0,1), c(0,1), c(0,1)))
fit <- ptproc.fit(ppm, optim.control = list(trace = 2), method = "BFGS")
summary(fit)
```

[6] On an interesting historical side note: The original paper reported —based on a benchmark— that "the algorithm is very fast; `maxent` uses only 135.4 megabytes of RAM and finishes in 53.3 seconds." On a modest computer in 2023 with a dockerized R 2.15.0, the benchmark finishes in 4 seconds.

[7] https://cran.r-project.org/web/views/SpatioTemporal.html

[8] https://www.biostat.jhsph.edu/~rpeng/software/

[9] https://blog.mathandpencil.com/installing-ptproc-on-osx and https://tomaxent.com/2017/03/16/Installing-ptproc-on-Ubuntu-16-04-LTS/

One can integrate `rang` into a BASH script to completely automate the batch execution of the above R script.

```
Rscript -e "require(rang); dockerize(resolve('ptproc', '2004-07-01'),
'pengdocker', cache = TRUE)"
docker build -t pengimg ./pengdocker
## launching a container in daemon mode -d
docker run -d --rm --name "pengcontainer" -ti pengimg
docker cp peng.R pengcontainer:/peng.R
docker exec pengcontainer R CMD BATCH peng.R
docker exec pengcontainer cat peng.Rout
docker cp pengcontainer:/peng.Rout peng.Rout
docker stop pengcontainer
```

The file `peng.Rout` contains the execution results of the script from inside the Docker container. As the random seed was preserved by the original author (Peng 2003), the above BASH script can perfectly reproduce the analysis [10].

3.6 Recover a removed Bioconductor package

Similar to CRAN, Bioconductor is a series of repositories of R package for bioinformatics and computational biology. Also similar to CRAN, packages can get removed over time.

The Bioconductor package `Sushi` has been deprecated by the original authors and is removed from Bioconductor version 3.16 (2022-11-02). `Sushi` is a data visualization tool for genomic data and was used in many online tutorials and scientific papers, including the original paper announcing the package by the original authors (Phanstiel et al. 2014).

`rang` has native support for Bioconductor packages since 0.2. We obtained the R script `"PaperFigure.R"` from the Github repo of `Sushi` [11], which generates the figure in Phanstiel et al. (2014). Similar to the above case of `ptproc`, we made a completely automated BASH script to run `"PaperFigure.R"` and get the generated figure out of the container (Figure 2). We made no modification to `"PaperFigure.R"`.

```
Rscript -e "require(rang); dockerize(resolve('Sushi', '2014-06-05'),
'sushidocker', no_rocker = TRUE, cache = TRUE)"
docker build -t sushiimg ./sushidocker
docker run -d --rm --name "sushicontainer" -ti sushiimg
docker cp PaperFigure.R sushicontainer:/PaperFigure.R
docker exec sushicontainer mkdir vignettes
docker exec sushicontainer R CMD BATCH PaperFigure.R
docker cp sushicontainer:/vignettes/Figure_1.pdf sushi_figure1.pdf
docker stop sushicontainer
```

4 Preparing research compendia with long-term computational reproducibility

The above six examples show how useful it is for `rang` reconstructing tricky computational environments which have not been completely declared in the literature. Although we position `rang` mostly as an archaeological tool, we also think that `rang` can also be used to prepare research compendia of current research. We can't predict the future but research compendia generated by `rang` would probably have long-term computational reproducibility.

To demonstrate this point, we took the recent paper by Oser et al. (2022). This paper was selected because 1) the paper was published in *Political Communication*, a high impact journal that awards Open Science Badges; 2) shared data and R code are available; and most importantly, 3) the shared R code is well-written. In the repository of this paper, we based on the materials shared by Oser et al. (2022) and prepared a research compendium that should have long-term computational reproducibility. The research compendium is similar to the Executable Compendium suggested by the Turing way.

[10] It is also important to note that the random number generator (RNG) of R has been changed several times over the course of the development. In this case, we are using the same generation of RNG as Peng (2003).

[11] https://github.com/PhanstielLab/Sushi/blob/master/vignettes/PaperFigure.R

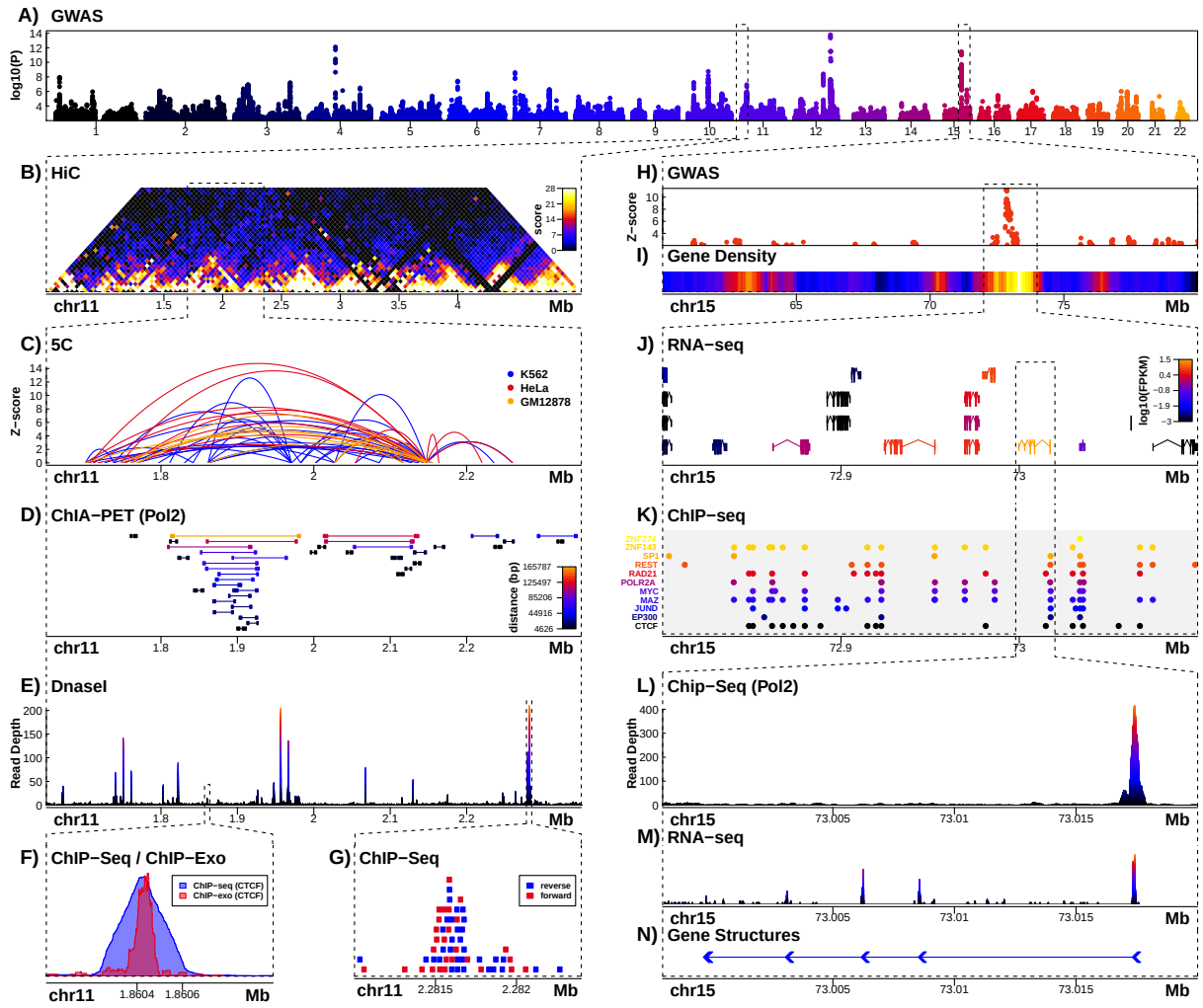

Figure 2: The figure from the batch execution of `PaperFigure.R` inside a Docker container generated by `rang`

The preparation of the research compendium is easy as `rang` can scan a materials directory for all R packages used [12].

```
require(rang)
## meta-analysis is the directory of all shared materials
cran_pkgs <- as_pkgrefs("meta-analysis")

## dmetar is an undeclared github package: MathiasHarrer/dmetar
cran_pkgs[cran_pkgs == "cran::dmetar"] <- "MathiasHarrer/dmetar"
x <- resolve(cran_pkgs, "2021-08-11", verbose = TRUE)
print(x, all_pkgs = TRUE)
dockerize(x, "oser_docker", materials_dir = "meta-analysis", cache = TRUE)
```

The above R script is saved as `oser.R`. The central piece of the executable compendium is the `Makefile`.

```
rmd = "rmarkdown::render('materials/README.Rmd', output_file = 'reproduced.html')"
resolve:
```

[12]We detected a minor issue in the code base that an undeclared Github package is used. But it can be easily solved, as in the Psychological Science example above.

```
    Rscript oser.R
build: oser_docker
    docker build -t oserimg oser_docker
render:
    docker run -d --rm --name "osercontainer" -ti oserimg
    docker exec osercontainer Rscript -e ${rmd}
    docker cp osercontainer:/materials/reproduced.html oser_README.html
    docker stop osercontainer
export:
    docker save oserimg | gzip > oserimg.tar.gz
rebuild: oserimg.tar.gz
    docker load < oserimg.tar.gz
all: resolve build render
    echo "finished"
```

With this `Makefile`, once can create the Dockerfile with `make resolve`, build the Docker image with `make build`, render the RMarkdown file inside the container with `make render`, export the built Docker image with `make export`, and rebuild the exported Docker image with `make rebuild`.

The structure of the executable compendium looks like this:

```
Makefile
oser.R
meta-analysis/
README.md
oser_docker/
oserimg.tar.gz
```

In this executable compendium, only the first four elements are essential. The directory `oser_docker` (116 MB) contains cached R packages, Dockerfile, and a further copy of the directory `meta-analysis/` to be transferred into the Docker image. That can be regenerated by running `make resolve`. However, having this directory preserved insures against the situations of some R packages used in the project are no longer available or any of the information providers used by `rang` for resolving the dependency relationships being not available. (Or in the rare circumstance of `rang` is no longer available.)

`oserimg.tar.gz` (667 MB) is a backup copy of the Docker image. This can be regenerated by running `make export`. Preserving this file insures against all the situations mentioned above, but also the situations of Docker Hub and the software repositories used by the dockerized operating system being not available. As a matter of fact, when `oserimg.tar.gz` is available it is possible to run `make rebuild` and `make render` even without internet access, provided that only Docker and make have been preinstalled. Of course, there is still an extremely rare situation where Docker (the program) itself is no longer available. However, it is possible to convert the image file for use on other containerization solutions such as Singularity [13].

Sharing of research artefacts less than 1G is not as challenging as it used to be. Zenodo, for example, allows the sharing of 50G of files. Therefore, sharing of the last two components of the executable compendium prepared with `rang` is at least possible on Zenodo. However, for data repositories with more restrictions on data size, sharing the executable compendium without the last two parts could be considered sufficient. For that, run `make all` will generate all the things needed for reproducing the analysis inside a container.

5 Conclusion

References

Abate, Pietro, Roberto Di Cosmo, Louis Gesbert, Fabrice Le Fessant, Ralf Treinen, and Stefano Zacchiroli. 2015. "Mining Component Repositories for Installability Issues." *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May. https://doi.org/10.1109/msr.2015.10.

[13] https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Beck, Nathaniel. 2019. "Estimating Grouped Data Models with a Binary-Dependent Variable and Fixed Effects via a Logit Versus a Linear Probability Model: The Impact of Dropped Units." *Political Analysis* 28 (1): 139–45. https://doi.org/10.1017/pan.2019.20.

Benoit, Kenneth, Kohei Watanabe, Haiyan Wang, Paul Nulty, Adam Obeng, Stefan Müller, and Akitaka Matsuo. 2018. "Quanteda: An R Package for the Quantitative Analysis of Textual Data." *Journal of Open Source Software* 3 (30): 774. https://doi.org/10.21105/joss.00774.

Boettiger, Carl, and Dirk Eddelbuettel. 2017. "An Introduction to Rocker: Docker Containers for R." *The R Journal* 9 (2): 527. https://doi.org/10.32614/rj-2017-065.

Crüwell, Sophia, Deborah Apthorp, Bradley J. Baker, Lincoln Colling, Malte Elson, Sandra J. Geiger, Sebastian Lobentanzer, et al. 2023. "What's in a Badge? A Computational Reproducibility Investigation of the Open Data Badge Policy in One Issue of Psychological Science." *Psychological Science*, February, 095679762211408. https://doi.org/10.1177/09567976221140828.

Dolstra, Eelco, Andres Löh, and Nicolas Pierron. 2010. "NixOS: A Purely Functional Linux Distribution." *Journal of Functional Programming* 20 (5–6): 577–615. https://doi.org/10.1017/s0956796810000195.

Hilgard, Joseph, Christopher R. Engelhardt, Jeffrey N. Rouder, Ines L. Segert, and Bruce D. Bartholow. 2019. "Null Effects of Game Violence, Game Difficulty, and 2D:4D Digit Ratio on Aggressive Behavior." *Psychological Science* 30 (4): 606–16. https://doi.org/10.1177/0956797619829688.

Jurka, P., Timothy. 2012. "Maxent: An r Package for Low-Memory Multinomial Logistic Regression with Support for Semi-Automated Text Classification." *The R Journal* 4 (1): 56. https://doi.org/10.32614/rj-2012-007.

Lörcher, Ines, and Monika Taddicken. 2017. "Discussing Climate Change Online. Topics and Perceptions in Online Climate Change Communication in Different Online Public Arenas." *Journal of Science Communication* 16 (02): A03. https://doi.org/10.22323/2.16020203.

Merow, Cory, Brad Boyle, Brian J. Enquist, Xiao Feng, Jamie M. Kass, Brian S. Maitner, Brian McGill, et al. 2023. "Better Incentives Are Needed to Reward Academic Software Development." *Nature Ecology & Evolution*, February. https://doi.org/10.1038/s41559-023-02008-w.

Nüst, Daniel, and Matthias Hinz. 2019. "Containerit: Generating Dockerfiles for Reproducible Research with r." *Journal of Open Source Software* 4 (40): 1603. https://doi.org/10.21105/joss.01603.

Ooi, Hong, Andrie de Vries, and Microsoft. 2022. *checkpoint: Install Packages from Snapshots on the Checkpoint Server for Reproducibility*. https://CRAN.R-project.org/package=checkpoint.

Oser, Jennifer, Amit Grinson, Shelley Boulianne, and Eran Halperin. 2022. "How Political Efficacy Relates to Online and Offline Political Participation: A Multilevel Meta-Analysis." *Political Communication* 39 (5): 607–33. https://doi.org/10.1080/10584609.2022.2086329.

Peng, Roger D. 2003. "Multi-Dimensional Point Process Models inR." *Journal of Statistical Software* 8 (16). https://doi.org/10.18637/jss.v008.i16.

Phanstiel, D. H., A. P. Boyle, C. L. Araya, and M. P. Snyder. 2014. "Sushi.r: Flexible, Quantitative and Integrative Genomic Visualizations for Publication-Quality Multi-Panel Figures." *Bioinformatics* 30 (19): 2808–10. https://doi.org/10.1093/bioinformatics/btu379.

Simonsohn, Uri, and Hugo Gruson. 2023. *groundhog: Version-Control for CRAN, GitHub, and GitLab Packages*. https://CRAN.R-project.org/package=groundhog.

Tierney, Luke. 2003. "The r Journal: Name Space Management for r." *R News* 3: 2–6.

Trisovic, Ana, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. 2022. "A Large-Scale Study on Research Code Quality and Execution." *Scientific Data* 9 (1). https://doi.org/10.1038/s41597-022-01143-6.

Ushey, Kevin. 2022. *renv: Project Environments*. https://CRAN.R-project.org/package=renv.