

MANAGING R ENVIRONMENTS & LIBRARIES

Jacalyn Huband

Senior Computational Scientist
UVA Research Computing

Gladys Andino

Senior Computational Scientist
UVA Research Computing



9 Feb. 2023

Let me tell a story . . .

- Last week, I was working on an R program for a researcher.
- I copied the code into my account to debug it.
- I worked on it for days.
- I finally got it to run correctly – Yay!
- I copied the code back into the researcher's account and also into his colleague's account.
- The code worked in my account and in the colleague's account but not in the researcher's account.



Things that I checked...

Same copy of the code?

Same version of R?

Same data being read in?

Same package versions?

Things that I checked...

Same copy of the code?

Same version of R?

Same data being read in?

Same package versions?



Lesson Learned

- The R environment (i.e., packages and versions) is important for *reproducibility* and *portability*.
- Definitions:
 - *portability*: the ability of software to be transferred from one machine or system to another:
 - *reproducibility*: the ability to get consistent results when an experiment is repeated

Questions that we will explore today

- How can I figure out what my R environment is?
- What happens when I install packages in R?
- How can I ensure that my R environment is the same when I share my code? Or, even when I move to another institution/job?

WHAT IS MY R ENVIRONMENT?

The computer's operating system

The version of R

The version of my packages

The Operating System and Version of R

- To know what operating system and version of R you are using, you can type *version* at the R command prompt.
- If using SLURM on Rivanna, make sure that you include the version number when loading the R module.

```
> version
platform      x86_64-pc-linux-gnu
arch          x86_64
os            linux-gnu
system        x86_64, linux-gnu
status
major         4
minor        1.1
year         2021
month        08
...
```


Capturing Package Information

> `sessionInfo()`

R version 4.1.1 (2021-08-10)

Platform: x86_64-pc-linux-gnu (64-bit)

Running under: CentOS Linux 7 (Core)

Matrix products: default

BLAS/LAPACK:

/sfs/applications/202212/software/standard/compiler/gcc/7.1.0/
openblas/0.2.19/lib/libopenblas-r0.2.19.so

...

attached base packages:

[1] stats graphics grDevices utils datasets methods base

loaded via a namespace (and not attached):

[1] compiler_4.1.1

- To collect information about the current session, type *`sessionInfo()`* in the R console.
- Notice that this tells us what packages are attached (i.e., loaded) and what are available for other packages to use (“loaded via a namespace”).

Capturing More Package Information

```
> devtools::session_info()
```

```
– Session info –
```

```
setting value
```

```
version R version 4.1.1 (2021-08-10)
```

```
os CentOS Linux 7 (Core)
```

```
system x86_64, linux-gnu
```

```
ui X11
```

```
language (EN)
```

```
collate en_US.UTF-8
```

```
ctype en_US.UTF-8
```

```
tz America/New_York
```

```
date 2023-02-05
```

```
pandoc NA
```

```
– Packages –
```

```
package * version date (UTC) lib source
```

```
assertthat 0.2.1 2019-03-21 [2] CRAN (R 4.1.1)
```

```
backports 1.4.1 2021-12-13 [1] CRAN (R 4.1.1)
```

```
...
```

```
xml2 1.3.3 2021-11-30 [1] CRAN (R 4.1.1)
```

```
[1] /sfs/qumulo/qhome/jmh5ad/R/goolf/4.1
```

```
[2]
```

```
/sfs/applications/202212/software/standard/mpi/gcc/7.1.0/openmpi/3.1.4/R/4.1.1/
```

```
lib64/R/library
```

- To get more information (date of package installation and the repository that was used to install the package), use *session_info()* from *devtools*.
- If you have devtools installed, type *devtools::session_info()* at the R prompt.

Capturing More Package Information

```
> devtools::session_info()
```

```
– Session info –
```

```
setting value
```

```
version R version 4.1.1 (2021-08-10)
```

```
os CentOS Linux 7 (Core)
```

```
system x86_64, linux-gnu
```

```
ui X11
```

```
language (EN)
```

```
collate en_US.UTF-8
```

```
ctype en_US.UTF-8
```

```
tz America/New_York
```

```
date 2023-02-05
```

```
pandoc NA
```

```
– Packages –
```

```
package * version date (UTC) lib source
```

```
assertthat 0.2.1 2019-03-21 [2] CRAN (R 4.1.1)
```

```
backports 1.4.1 2021-12-13 [1] CRAN (R 4.1.1)
```

```
...
```

```
xml2 1.3.3 2021-11-30 [1] CRAN (R 4.1.1)
```

```
[1] /sfs/qumulo/qhome/jmh5ad/R/goolf/4.1
```

```
[2]
```

```
/sfs/applications/202212/software/standard/mpi/gcc/7.1.0/openmpi/3.1.4/R/4.1.1/  
lib64/R/library
```

The `devtools::session_info()` function also will show the location of your R libraries



Hands-on Activity: Capturing Package Information

- Start up R in the way that you are most comfortable (e.g., Rstudio).
 - In the console, type: `sessionInfo()`
 - Or, if you already have devtools installed, type: `devtools::session_info()`
 - Load your favorite package (e.g., `library(DESeq2)`)
 - Again type: `sessionInfo()`
 - What are the differences?

Hands-on Activity Results: What I see after loading DESeq2

```
> sessionInfo()
```

```
R version 4.1.1 (2021-08-10)
```

```
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
Running under: CentOS Linux 7 (Core)
```

```
Matrix products: default
```

```
BLAS/LAPACK: /sfs/applications/2021-08-10/softwares/blas-lapack/blas-r0.2.19.so
```

```
locale:
```

```
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
```

```
[7] LC_PAPER=en_US.UTF-8 LC_NAME=C
```

```
attached base packages:
```

```
[1] stats graphics grDevices utils datasets methods
```

```
loaded via a namespace (and not attached):
```

```
[1] compiler_4.1.1 tools_4.1.1
```

```
other attached packages:
```

```
[1] DESeq2_1.34.0
```

```
Summa
```

```
[6] GenomicRanges_1.46.1
```

```
Genom
```

```
loaded via a namespace (and not attached):
```

```
[1] KEGGREST_1.34.0
```

```
genefilter
```

```
LC_MONUS.
```

```
[8] colorspace_2.0-3
```

```
vctrs_0.5.1
```

```
ASUREMENT
```

```
[15] pillar_1.6.4
```

```
glue_1.6.0
```

```
[22] lifecycle_1.0.3
```

```
zlibbioc_1.0.6
```

```
[29] fastmap_1.1.0
```

```
parallel_4.0.2
```

```
[36] cachem_1.0.6
```

```
DelayedArray_0.18.0
```

LET'S TAKE A CLOSER LOOK AT R LIBRARIES

Libraries in R

- A library is simply a directory (i.e., folder) where the codes for packages are placed.
- When R is installed on any computer, it establishes a system library.
- You also can have user libraries.
- You can create a separate library to capture the packages for your working code.

What libraries do I have?

- Start up R in the way that you are most comfortable
- To see what R libraries you can access, type in the console:
`.libPaths()`

.libPaths()

```
> .libPaths()  
"/sfs/qumulo/qhome/jmh5ad/R/x86_64-pc-linux-gnu-library/3.5"  
"/sfs/qumulo/dev_apps/software/standard/compiler/gcc/7.1.0/R/3.5.1  
/lib64/R/library"  
>
```

Linux

```
> .libPaths()  
> "C:/Users/jmh5ad/Documents/R/win-library/3.4" "C:/Program  
Files/R/R-3.4.4/library"  
>
```

Windows

```
> .libPaths()  
> "/Library/Frameworks/R.framework/Versions/3.5/Resources/library"  
>
```

Mac

- If there are multiple libraries, R will search the libraries (in the order in which they appear) for a package.

.libPaths()

```
> .libPaths()  
"/sfs/qumulo/qhome/jmh5ad/R/x86_64-pc-linux-gnu-library/3.5"  
"/sfs/qumulo/dev_apps/software/standard/compiler/gcc/7.1.0/R/3.5.1  
/lib64/R/library"  
>
```

Linux

```
> .libPaths()  
> "C:/Users/jmh5ad/Documents/R/win-library/3.4" "C:/Program  
Files/R/R-3.4.4/library"  
>
```

Windows

```
> .libPaths()  
> "/Library/Frameworks/R.framework/Versions/3.5/Resources/library"  
>
```

Mac

- Notice that the libraries generally are associated with the major and minor version of the R program.

Important Fact

If you update the version of R on your computer, you will need to reinstall or update the packages that you have installed previously.

Installing Older Packages

- If you know the version of a package that you need to use, you can install it with *devtools::install_version()*. For example:

```
devtools::install_version("BiocManager", version = "1.30.19",  
                           repos = "http://cran.us.r-project.org")
```

CHECKING YOUR PACKAGES

What packages are already installed?

Notes

- To see what packages are installed, you can use the *installed.packages()* function.

R Code

```
> pkgs <- as.data.frame(installed.packages())  
> head(pkgs %>% select(Version, Priority))  
      Version Priority  
arrow      8.0.0    <NA>  
askpass      1.1    <NA>  
assertthat  0.2.1    <NA>  
backports   1.4.1    <NA>  
base64enc   0.1-3    <NA>  
BH          1.78.0-0  <NA>
```

What packages are already installed?

Notes

- To see what packages are installed, you can use the *installed.packages()* function.

R Code

```
> pkgs <- as.data.frame(installed.packages())  
> head(pkgs %>% select(Version, Priority))
```

	Version	Priority
arrow	8.0.0	<NA>
askpass	1.1	<NA>
assertthat	0.2.1	<NA>
backports	1.4.1	<NA>
base64enc	0.1-3	<NA>
BH	1.78.0-0	<NA>

The “Priority” indicates whether a package comes with R or has been installed by the user. (NA means installed by user.)

Installing Packages

- How you install a package depends on where the code for it resides.
 - With the CRAN repository, use `install.packages("somePackageName")`

```
install.packages('devtools')
```

- With the Bioconductor repository, use `BiocManager::install`

```
if (!require('BiocManager')) {  
  install.packages('BiocManager')  
}  
BiocManager::install('annotate')
```

- With Github, use `devtools::install_github`

```
devtools::install_github("mkearney/rtweet")
```


Comments on `install.packages()`

- In general, you only need to install a package once on your computer
- We do not recommend installing the packages every time that you run your code.
 - If a package is updated, you could break your code.
 - For reproducibility purposes, you may want to ensure that nothing changes after your code runs successfully.

Is there a way to capture the state of your packages after a successful run?

RENV

renv Overview

- renv is a package in R that manages packages to create reproducible environments.
- A benefit of renv is that it keeps track of the packages and package versions of individual codes.
- It produces a file with information for re-creating the package environment.

renv Steps

- Suppose your code worked with older packages and you want to preserve that environment. But, you want to use newer packages for new code development.
- Steps to do on your laptop:
 1. Create a folder for your project and place your code in the folder
 2. Open R or Rstudio and install the renv package (if not already installed)
 3. Initialize renv with the folder path
 4. Install any packages that your project needs
 5. Capture the environment by taking a “snapshot” of the project.
 6. Restore the environment as needed to run the code.

Step 1: Create a folder for your project

- Create a folder of move to the directory/folder where your code exists.

Step 2: Install the renv Package

- Open R or Rstudio and install renv

```
> install.packages("renv")
Installing package into
‘/sfs/qumulo/qhome/jmh5ad/R/goolf/4.1’
(as ‘lib’ is unspecified)

. . .

* DONE (renv)

The downloaded source packages are in
‘/tmp/RtmptSMp5T/downloaded_packages’
```

Step 3: Initialize renv

- Initialize renv for the directory where your code is located

```
> renv::init("~/renv_example")
* Initializing project ...
* Discovering package dependencies ... Done!
* Copying packages into the cache ... Done!
The following package(s) will be updated in the lockfile:

# CRAN =====
- renv      [* -> 0.16.0]
- rpart     [* -> 4.1.16]

The version of R recorded in the lockfile will be updated:
- R         [*] -> [4.1.1]

* Lockfile written to
'/sfs/qumulo/qhome/jmh5ad/renv_example/renv.lock'.
Restarting R session...
```

Step 3.5: Restart R

- You may need to exit out of your current session and restart R/Rstudio.
- Work with your code, installing packages as needed.

```
R version 4.1.1 (2021-08-10) -- "Kick Things"
Copyright (C) 2021 The R Foundation for Statistical Computing
> install.packages('rlang')
Retrieving 'https://cran.rstudio.com/src/contrib/rlang_1.0.6.tar.gz' ...
      OK [file is up to date]
Installing rlang [1.0.6] ...
      OK [built from source]
Moving rlang [1.0.6] into the cache ...
      OK [moved to cache in 0.11 seconds]
```


Step 4: Take a snapshot

- After installing any necessary packages and testing the code, take a snapshot.

```
> renv::snapshot()  
The following package(s) will be updated in the lockfile:  
  
# CRAN =====  
- rlang [* -> 1.0.6]  
  
Do you want to proceed? [y/N]: y  
* Lockfile written to  
'/sfs/qumulo/qhome/jmh5ad/renv_example/renv.lock'.
```

Step 5: Return to the code

- Anytime that you return to your code, set the working directory to the location of the library and run `renv::activate()`

```
> setwd("~/renv_example")  
> renv::activate()  
• Project '/sfs/qumulo/qhome/jmh5ad/renv_example'  
loaded. [renv 0.16.0]
```

Sharing Code

- If you want to share your code, include your files and the file *renv.lock* which is created when you initiated renv.
- The recipient can place the files in a folder, open R/RStudio in the folder (or set the working directory to the folder), and type: `renv::restore()`.
- When it asks if you want to activate, respond with 'Y'.

Sharing Code: Activating Project

```
> setwd("/scratch/jmh5ad/new_location")  
> renv::restore()
```

This project has not yet been activated.

Activating this project will ensure the project library is used during restore.

Please see `?renv::activate` for more details.

Would you like to activate this project before restore? [Y/n]: Y

* Project '/gpfs/gpfs0/scratch/jmh5ad/new_location' loaded. [renv 0.16.0]

* The project library is out of sync with the lockfile.

* Use `renv::restore()` to install packages recorded in the lockfile.

The following package(s) will be updated:

```
# CRAN =====
```

```
- rlang [* -> 1.0.6]
```

```
- rpart [* -> 4.1.16]
```

Do you want to proceed? [y/N]: y

Installing rlang [1.0.6] ...

OK [linked cache]

Installing rpart [4.1.16] ...

OK [linked cache]

Cautions with renv

- The package renv is great for maintaining a library for your code, especially if you have other codes that need different versions of packages.
- Although you can share your codes with other researchers, be aware that restoring a library may not work with different versions of R or with different operating systems.

PACKRAT

Packrat Overview

- Packrat is a package that renv hopes to replace.
- One advantage of packrat is that it maintains source code for the R packages. This means that your code will be
 - more portable and
 - reproducible

Packrat Steps

- Suppose you have code working on your laptop and you want to move it to Rivanna.
- Steps to do on your laptop:
 1. Install the packrat package in R
 2. Create a folder for your project
 3. Initialize packrat, using the folder name
 4. Install any packages that your project needs
 5. Capture the environment by taking a “snapshot” of the project.
 6. Bundle the environment in a compressed file that can be shared.

Step 1: Create a folder for your project

- Create a folder of move to the directory/folder where your code exists.

Step 2: Install the Packrat Package

- Open R or Rstudio and install Packrat

```
> install.packages("Packrat")
```

```
. . .
```

```
* DONE (Packrat)
```

```
The downloaded source packages are in  
  '/tmp/RtmpSMp5T/downloaded_packages'
```

Step 3: Initialize Packrat

- Initialize renv for the directory where your code is located

```
> packrat::init("~/packrat_example")
Initializing packrat project in directory:
- "~/packrat_example"

Adding these packages to packrat:

      _
packrat 0.9.0
randomForest 4.6-14

. . .
Snapshot written to
'/home/jmh5ad/packrat_example/packrat/packrat.lock'
Installing packrat (0.9.0) ... OK (built source) Installing
randomForest (4.6-14) ...
                                OK (built source) Initialization
complete!
Restarting R session...
```

Step 3.5: Restart R

- You may need to exit out of your current session and restart R/Rstudio.
- Work with your code, installing packages as needed.

```
R version 4.1.1 (2021-08-10) -- "Kick Things"
Copyright (C) 2021 The R Foundation for Statistical Computing
> install.packages('rlang')
Retrieving 'https://cran.rstudio.com/src/contrib/rlang_1.0.6.tar.gz' ...
      OK [file is up to date]
Installing rlang [1.0.6] ...
      OK [built from source]
Moving rlang [1.0.6] into the cache ...
      OK [moved to cache in 0.11 seconds]
```

Step 4: Take a snapshot

- After installing any necessary packages and testing the code, take a snapshot.

```
> packrat::snapshot()  
Snapshot written to  
'/home/jmh5ad/packrat_example/packrat/packrat.lock'
```

Step 5: Return to the code

- Anytime that you return to your code, set the working directory to the location of the library and run `renv::activate()`

```
> setwd("~/packrat_example")  
> packrat::restore()
```

Sharing the code: Compressing the Environment

- The bundle command will compress the environment and save it to a file. (It even appends a date to the file name.)
- The file can be archived and ported to other platforms.

```
> packrat::bundle(include.lib = TRUE)
The packrat project has been bundled at:
- "/home/jmh5ad/packrat_example/packrat/bundles/packrat_example-2023-02-06.tar.gz"
> |
```

Packrat Example: Recreate the Environ.

- To recreate the environment that I had on my laptop, I can share the bundled file, called a tarball. The receiver of my tarball needs to do two steps:
 - “Unbundle” the tarball
 - Run R from within the new directory.

Packrat Example: Recreate the Environ.

```
> packrat::unbundle("packrat_example-2023-02-06.tar.gz", where=".", restore=TRUE)
- Untarring 'packrat_example-2023-02-06.tar.gz' in directory '/gpfs/gpfs0/scratch/jmh5ad'...
- Restoring project library...
Installing packrat (0.9.0) ...
  OK (built source)
Installing randomForest (4.6-14) ...
  OK (built source)
Done! The project has been unbundled and restored at:
- "/gpfs/gpfs0/scratch/jmh5ad/packrat_example"
Warning messages:
1: R graphics engine version 14 is not supported by this version of RStudio. The Plots tab will be disabled until a newer version of RStudio is installed.
2: In restore(project = getwd(), restart = FALSE) :
  The most recent snapshot was generated using R version 4.0.2
> |
```

Packrat Example: Recreate the Environ.

- To ensure that the packrat environment is used, move to the associated folder and turn on packrat:
 - “setwd(“~/packrat_example”)
 - “packrat::on()

```
> .libPaths()
[1] "/home/jmh5ad/R/x86_64-pc-linux-gnu-library/4.0"
[2] "/library"
[3] "/usr/local/lib/R/site-library"
[4] "/usr/lib/R/site-library"
[5] "/usr/lib/R/library"

> setwd("~/packrat_example")
> packrat::on()
Packrat mode on. Using library in directory:
- "~/packrat_example/packrat/lib"
> .libPaths()
[1] "/home/jmh5ad/packrat_example/packrat/lib/x86_64-pc-linux-gnu/4.0.2"
[2] "/home/jmh5ad/packrat_example/packrat/lib-ext/x86_64-pc-linux-gnu/4.0.2"
[3] "/home/jmh5ad/packrat_example/packrat/lib-R/x86_64-pc-linux-gnu/4.0.2"
>
```

Packrat Summary

- Packrat is a tool for managing the R packages associated with a project.
 - It ensures that exact versions of packages are preserved.
 - It keeps project libraries separate.
 - It provides a way to make your code portable, by keeping the source code of the packages.

CONCLUSION

Save Your R Environment

- How?
 - Keep a list of the packages and versions that were used with your code; or
 - Use the *packrat* package to capture **everything**; or
 - Use the *renv* package to capture information for the given operating system and version of R.
- Why?
 - Packages will change – there is no guarantee that your code will work six months from now.
 - It will save you time (and headaches) in the long run.

Need more help?

Office Hours

Tuesdays: 3 pm - 5 pm

Thursdays: 10 am - noon

Website:

rc.virginia.edu

Or, submit a request for help through the web for at:

<https://www.rc.virginia.edu/form/support-request/>

Questions?

