Up to Bat with Packrat

Introduction to Packrat

R is a powerful programming language for statistical computing with many packages and tools.

The goal of this article is to arm you with tools and techniques for using packrat.

When you or someone else wants to run your project months or years in the future, there will be no need to figure out what versions of packages were used and the source as Packrat will retain the information.

- https://rstudio.github.io/packrat/ (https://rstudio.github.io/packrat/)
- https://cran.r-project.org/web/packages/packrat/index.html (https://cran.r-project.org/web/packages/packrat/index.html)
- https://rpubs.com/jjallaire/packrat-useR-2014 (https://rpubs.com/jjallaire/packrat-useR-2014)

We'll cover the topics below:

1. **How would you reproduce this analysis?**
2. **Motivation - Reproducible Research**
3. **My Take**
4. **Quick Notes on Managing Libraries**
5. **Packrat - What it is?**
6. **Packrat - Installing**
7. **Packrat Fundamentals**
8. **Packrat Commands & Functions**
9. **Anatomy of a Packrat Project**
10. **Packrat Tips**
11. **Quick start - Stand Alone Full Mode Demo**
12. **Quick start - Stand Alone Lite Mode Demo**
13. **RStudio Projects**
14. **Using Packrat with RStudio**
15. **RStudio Packrat Notes**
16. **Demo - RStudio Full Mode**
17. **What if I Want to Use my Own Local Repository?**
18. **Packrat Bundle Sharing**
19. **Collaboration**
20. **Packrat & Version Control**
21. **Packrat Notes**

22. **Other Package Tools Used for Managing Packages & Package Dependencies**
23. **Other Tools**
24. **Interesting**

How would you reproduce this analysis?

- http://spark.rstudio.com/taxiDemoH2O.nb.html# (http://spark.rstudio.com/taxiDemoH2O.nb.html#)

What items were used?

1. Data?

- The NYC taxi data contain over 1 billion records describing pickups and drop offs
- Spark DataFrames & H2O Frames
- Hadoop - Cloudera / Hortonworks?
- Databases?

2. Packages?

- library(sparklyr)
- library(tidyverse)
- library(leaflet)
- library(rsparkling)
- library(h2o)
- library(DT)
- library(rpart)

3. Other software?

- Environment Variables
- Spark
- RStudio/R
- R version
- Operating system

4. Platform/hardware?

- Hardware
- Machine Architecture
- Platform size and compute
- Cluster
- Cloud environment - AWS, Azure, Google

What steps were followed?

- sqlvis_raster.R
- sqlvis_histogram.R
- taxiDemoH2O.Rmd

When was it done?

What was the purpose?

https://github.com/rstudio/sparkDemos/blob/master/prod/presentations/sparkSummitEast/taxiDemoH2O.Rmd (https://github.com/rstudio/sparkDemos/blob/master/prod/presentations/sparkSummitEast/taxiDemoH2O.Rmd)

How much time, effort, money, people would it take to repeat the analysis above?

What is Reproducibility?

“Reproducibility is the ability of an entire analysis of an experiment or study to be duplicated, either by the same researcher or by someone else working independently, whereas reproducing an experiment is called replicating it”

- https://en.wikipedia.org/wiki/Reproducibility (https://en.wikipedia.org/wiki/Reproducibility)
- http://biorxiv.org/content/early/2016/07/29/066803 (http://biorxiv.org/content/early/2016/07/29/066803)
- http://t-redactyl.io/blog/2016/10/a-crash-course-in-reproducible-research-in-r.html (http://t-redactyl.io/blog/2016/10/a-crash-course-in-reproducible-research-in-r.html)
- http://ivory.idyll.org/blog/2017-pof-software-archivability.html (http://ivory.idyll.org/blog/2017-pof-software-archivability.html)
- http://blog.jom.link/ten_rules_reproductible_research.html (http://blog.jom.link/ten_rules_reproductible_research.html)

Should everything be reproducible?

Can we reproduce these reports?

https://blogs.wsj.com/dailyshot/2017/04/23/wsjs-daily-shot-kickstarting-americas-productuvity/ (https://blogs.wsj.com/dailyshot/2017/04/23/wsjs-daily-shot-kickstarting-americas-productuvity/)

Can we even access data behind the plots?

If you have an enterprise environment for reproducible research, then please give your R admin/platform manager a high five for me.

- https://ropensci.org/blog/2014/06/09/reproducibility/ (https://ropensci.org/blog/2014/06/09/reproducibility/)

“It is difficult to find a counter-argument to these claims, but arguing that reproducibility is laudable in general glosses over the fact that for each research group it is a significant amount of work to make their research (easily) reproducible for independent scientists.”"

Currently, reproducibility is hard work and it might help to acknowledge that exact repeatability has a half life of utility.

- http://ivory.idyll.org/blog/2017-pof-software-archivability.html (http://ivory.idyll.org/blog/2017-pof-software-archivability.html)

After a review of the options (packrat, docker, git, etc.) it seems there is no one option as of yet that satisfies many of the needs of reproducibility and many offer 2 of 3 benefits/features below:

1. Open Source & Part of Data Science Ecosystem
2. Supports Reproducibility
3. Functional / User Friendly / Ease of Use

Motivation - Reproducible Research

- https://cran.r-project.org/web/views/ReproducibleResearch.html (https://cran.r-project.org/web/views/ReproducibleResearch.html)

R packages change over time, and often they are not backwards compatible...and there are multiple dimensions:

- Reproducible Research across time (running the same analysis again years later)
- Reproducible Research across space (moving code from a desktop to a server, or between the systems of collaborators)

Freezing projects can help by confining arbitrary combinations of R packages with a guarantee of being able to use them in the future.

Why Bother?

The rapid rate at which data science environments are changing, it can very easily become the case that scripts run in a specific environment no longer properly work after a just short period of time. That puts a clock on the reproducibility of your project, which can limit its impact and usefulness.

Have you ever asked?

1. Where even is that R script?
2. This code totally worked the last time I used it, and it simply doesn't now. What changed?
3. Is this the code I used for that conference poster, or was it another version? Wasn't there some code in here for sequence plots? Where did that go?

Does your old code look more like a series of misunderstandings and cryptic notes than clean, reproducible analyses?

"Common Problem: Let's say you coded a project in R in 2014 using the package dplyr and you called the project 2014project. Over time packages get updated, sometimes changing syntax and function names, let's say some major syntax changes occurred in 2015 to dplyr and you used the latest version of dplyr in other projects in 2015. Now you come to 2016 and you want to re-run the code in the 2014project, but you find a bunch of errors because in 2015 dplyr had those syntax changes, such that the syntax you used in 2014 no longer computes..."

- http://lukesingham.com/using-package-management-in-r/ (http://lukesingham.com/using-package-management-in-r/)

Once you start working with R, you'll quickly find out that R package dependencies can cause a lot of headaches and unfortunately, private libraries don't travel well; like all R libraries, their contents are compiled for your specific machine architecture, operating system, and R version.

So what are people doing today?

1. Nothing or Capturing Session & Sys Info - maybe annotating, or commenting on the code. Good annotations can help users determine if issues are code issues, are package related (and can therefore be addressed with packrat), or are (rarely) issues with versions of R. R version errors are harder to fix, and are not addressed by the packrat package. They might store the results of sessionInfo() or devtools::session_info() for the analysis and Sys.info() too. The information from sessionInfo() is used to recreate the analysis environment if needed to repeat down the road.
2. sessionInfo, R Markdown & Git - Users capture sessionInfo() and use R Markdown to help structure the analyses and output. At a minimum, one would hope for all of the appropriate data, a description of the software used for analysis or the software itself, and some metadata describing the data and how it was processed. Git and Github (or insert your favorite version control and code-sharing tooling here) is used to track changes and share projects with collaborators.

Here is a R Markdown example with sessionInfo:

- https://rpubs.com/jesuscastagnetto/caret-knn-cancer-prediction (https://rpubs.com/jesuscastagnetto/caret-knn-cancer-prediction)
- http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/ (http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/)
3. R Markdown, Git and Packrat - All of 2 above is used with packrat in usually two fashions:

- Packrat lite - use *packrat::.snapshotImpl(snapshot.sources = False, ".")* to generate the packrat.lock file to list the precise package versions that were used to satisfy dependencies, including dependencies of dependencies. Packages are not stored.
- https://github.com/rstudio/packrat/blob/master/man/snapshotImpl.Rd (https://github.com/rstudio/packrat/blob/master/man/snapshotImpl.Rd)
- Packrat Full - Using packrat by storing your package dependencies in your private package library.

People at this level will usually also mirror the cran repo and use a local repo.

"R package dependencies can be frustrating. Have you ever had to use trial-and-error to figure out what R packages you need to install to make someone else's code work-and then been left with those packages globally installed forever, because now you're not sure whether you need them? Have you ever updated a package to get code in one of your projects to work, only to find that the updated package makes code in another project stop working?"

- https://rstudio.github.io/packrat/ (https://rstudio.github.io/packrat/)

"Because of how painless it makes sharing reproducible projects—either openly with the public, with collaborators, or just between different machines—Packrat is an essential tool for R."

- http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/ (http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/)

"Depending on the level of experience, R users may not have ran into this issue before but it is a persistent problem with the R system. Due to the dynamic and open nature of the software, changes and improvements to packages can tweek the way that certain functions interact, making old code buggy or obsolete. Packrat is an attempt to control for this."

- https://cereo.wsu.edu/2017/02/23/packrat-package-managing-package-versions/ (https://cereo.wsu.edu/2017/02/23/packrat-package-managing-package-versions/)

4. Docker, R Markdown, Git and sometimes packrat too - Usually all of 1-3 above and now Docker and other tools like chef and travis-ci.

- http://www.baselr.org/presentations/2017/03/BaselR_-_Reproducible_analysis_-_Joris_Muller_-_20170307.pdf (http://www.baselr.org/presentations/2017/03/BaselR_-_Reproducible_analysis_-_Joris_Muller_-_20170307.pdf)
- https://ropensci.org/blog/2014/11/10/open-data-growth.html (https://ropensci.org/blog/2014/11/10/open-data-growth.html)
- http://blog.jom.link/ten_rules_reproductible_research.html (http://blog.jom.link/ten_rules_reproductible_research.html)

My Take

Words like "reproducibility", "portability" , "repeatability", "replicability" and "unit testing" are buzzing big time.

Managing reproducibility is like managing security. A good security program balances risk, business impact and cost using various controls and determines at what level to implement the strategies.

- https://rpubs.com/cosmicBboy/nyc-r-stats-pegged1 (https://rpubs.com/cosmicBboy/nyc-r-stats-pegged1)

By adopting packrat, you can:

- Improve knowledge transfer among team
- Maintain a reproducible workflow
- Provide a dependency management system for R

Some limitations include:

- Necessary Time Investment and extra steps in workflow
- Learning curve
- Breaking old habits
- *Maybe Extra IT workload

Quick Notes on Managing Libraries

Many people use the standard R install and use common/system libraries or user libraries, and download from CRAN to these locations. Some limit package installs from CRAN using encrypted HTTPS connections as explained here:

https://support.rstudio.com/hc/en-us/articles/206827897-Secure-Package-Downloads-for-R (https://support.rstudio.com/hc/en-us/articles/206827897-Secure-Package-Downloads-for-R)

Common/system libraries & user information is here:

Managing user vs system libraries:

- https://support.rstudio.com/hc/en-us/articles/215733837-Managing-libraries-for-RStudio-Server (https://support.rstudio.com/hc/en-us/articles/215733837-Managing-libraries-for-RStudio-Server)
- https://cran.r-project.org/doc/manuals/r-release/R-admin.html#Managing-libraries (https://cran.r-project.org/doc/manuals/r-release/R-admin.html#Managing-libraries)

If you have mutiple versions of R, you can read more here:

https://support.rstudio.com/hc/en-us/articles/200486138-Using-Different-Versions-of-R (https://support.rstudio.com/hc/en-us/articles/200486138-Using-Different-Versions-of-R)

General package information is here:

https://support.rstudio.com/hc/en-us/articles/200486508-Building-Testing-and-Distributing-Packages (https://support.rstudio.com/hc/en-us/articles/200486508-Building-Testing-and-Distributing-Packages)

Updating libraries with versions of R:

- http://rpubs.com/wch/275466 (http://rpubs.com/wch/275466)

Upgrading R:

The advice is usually not to, and to install the new version side-by-side with the old version.

https://support.rstudio.com/hc/en-us/articles/215488098-Installing-multiple-versions-of-R (https://support.rstudio.com/hc/en-us/articles/215488098-Installing-multiple-versions-of-R)

http://docs.rstudio.com/ide/server-pro/r-versions.html#installing-multiple-versions-of-r (http://docs.rstudio.com/ide/server-pro/r-versions.html#installing-multiple-versions-of-r)

http://docs.rstudio.com/ide/server-pro/r-versions.html#managing-upgrades-of-r (http://docs.rstudio.com/ide/server-pro/r-versions.html#managing-upgrades-of-r)

https://support.rstudio.com/hc/en-us/articles/218004217-Building-R-from-source (https://support.rstudio.com/hc/en-us/articles/218004217-Building-R-from-source)

You may also take a look at:

- https://github.com/r-lib/crancache (https://github.com/r-lib/crancache)

Packrat - What it is?

Packrat is an R package that implements a dependency management system for R.

Packrat creates a special kind of directory - a private package library for a given R project or directory.

Within the packrat directory, any libraries you install are isolated.

```
readLines(system.file("DESCRIPTION", package = "packrat"))[c(3,9,10)]
```

```
## [1] "Title: A Dependency Management System for Projects and their R Package"
## [2] "    on in an isolated, portable, and reproducible way."
## [3] "License: GPL-2"
```

The last 'snapshotted' state can be used to save and restore the state of the private library

Packrat captures all source code required to reproduce configurations and requires no changes to CRAN.

- https://www.rstudio.com/resources/webinars/managing-package-dependencies-in-r-with-packrat/ (https://www.rstudio.com/resources/webinars/managing-package-dependencies-in-r-with-packrat/)

Packrat - Installing

Packrat is on CRAN, so you can install it with:

```
# install.packages("packrat")
```

- https://blog.rstudio.org/2014/09/05/packrat-on-cran/ (https://blog.rstudio.org/2014/09/05/packrat-on-cran/)

Or you can install using devtools:

```
# install.packages("devtools")
# devtools::install_github("rstudio/packrat")
```

Once you initialize Packrat on that directory, it will keep track of which version of R is used as well as which packages are used in your scripts, including their version information and source code. This makes your project completely self-contained.

Packrat Fundamentals

Packrat folders are able to keep track of your dependencies due to the presence of a lock file, which simply keeps track of what libraries you installed and their versions.

init will find any packages used by the scripts in your project and download the source code for the version used. It will also store the version of R and a copy of itself, so Packrat does not need to already be installed on other machines in order for your project to be used.

- http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/ (http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/)
- Packrat uses the "::" syntax to call commands as it is a way to specify exactly which packages commands are being used.

Packrat Commands & Functions

- https://rstudio.github.io/packrat/commands.html (https://rstudio.github.io/packrat/commands.html)
- packrat::init("~/path/to/repo") : Create a packrat project, initializes an isolated package library for the current project
- packrat::bootstrap has been renamed to packrat::init()
- packrat::snapshot() : packages in use and stores those packages, saves all changes within the packrat project library since the last snapshot
- packrat::restore() : Restore the directory to the last snapshotted state, deletes all changes within the packrat project library since the last snapshot
- packrat::install_github()
- packrat::bundle() : creates a tarball of the project, the packages and its dependencies
- packrat::unbundle((bundle = "path//to//the//bundle", where = ".") : unbundles (restores) a project within a subdirectory of another (conveniently empty) project
- packrat::status()
- packrat::clean()
- packrat::packrat_mode() : capabilities for easily moving in and out of Packrat projects
- packrat::disable()
- ?"packrat-options"
- packrat::set_opts() : Get/set project-specific settings
- packrat::get_opts()
- packrat::off() : lets you temporarily step out of "packrat mode", which means, that you will now use your global user library instead of the packrat project library
- packrat::on() : lets you go back into "packrat mode"

Other helpful commands/functions:

- getwd() : not packrat function but tells you which directory
- .libPaths() : You can see all the library paths installed for your user by issuing this command in the console

Anatomy of a Packrat Project

https://rpubs.com/cosmicBboy/nyc-r-stats-pegged1 (https://rpubs.com/cosmicBboy/nyc-r-stats-pegged1)

Within packrat, each folder is essentially its own environment, with its own packages. When you use Packrat via the RStudio IDE integration, it will actually be its own project.

You will need to have all the data and scripts related to your project in its own directory. How these files are named and organized is up to you.

- .Rprofile , Directs R to use the private package library (when it is started from the project directory).
- packrat/lib/ , Private package library for this project.

- packrat/src/ , Source packages of all the dependencies that packrat has been made aware of.
- packrat/packrat.lock , Lists the precise package versions that were used to satisfy dependencies, including dependencies of dependencies.
- packrat/packrat.opts , Project-specific packrat options.

Packrat Tips

- *packrat::init()*: This will allow packrat to find the packages used by scanning for `library()` or `require()` functions in the scripts. Furthermore, when some packages are added when writing R code, packrat will automatically add it.
- request users call devtools::install_github() directly : See https://github.com/rstudio/packrat/issues/362 (https://github.com/rstudio/packrat/issues/362)
- When a user opts in to using packrat with an RStudio project, one of the things packrat automatically does is create (or modify) a project-specific .Rprofile. Packrat uses the .Rprofile to ensure that each time the project opens, Packrat mode is turned on.
- Once you are working within a packrat session there are some useful commands to know. One is sessionInfo() which shows what versions of things you have loaded. devtools::session_info() can also be used and has more info.

http://blog.jom.link/implementation_basic_reproductible_workflow.html (http://blog.jom.link/implementation_basic_reproductible_workflow.html)

- There is also a way to install older versions of packages – this is useful if you want to create a new packrat project but you realize your current packages are too new. Information on how to do that can be found here : https://support.rstudio.com/hc/en-us/articles/219949047-Installing-older-versions-of-packages (https://support.rstudio.com/hc/en-us/articles/219949047-Installing-older-versions-of-packages)
- As long as you are in packrat_mode, when you run install.packages() or remove remove.packages() it will only be modifying your project folder.
- Installing local source packages - You may be working on a project with an R package that is not available on any external repository. Packrat can still handle this... With source packages, we expect these packages live in a local repository. A local repository is just a directory containing package sources. This can be set within a packrat project with: *packrat::set_opts(local.repos = "")*

Quick start - Stand Alone Full Mode Demo

Create a new directory that will contain all the .R scripts, CSV data, and other files that are needed.

```
# devtools::install_github("rstudio/packrat")
# setwd("~/packrat_test")
# library(packrat)
# packrat::init()
# install.packages("ggplot2")
# library(ggplot2)
# Run file
# packrat::snapshot()
# packrat::bundle()
```

```
# remove.packages('ggplot2')
# packrat::status()
# packrat::restore()
```

- https://rpubs.com/nishantsbi/221948 (https://rpubs.com/nishantsbi/221948)
- https://rstudio.github.io/packrat/walkthrough.html (https://rstudio.github.io/packrat/walkthrough.html)
- https://cereo.wsu.edu/2017/02/23/packrat-package-managing-package-versions/ (https://cereo.wsu.edu/2017/02/23/packrat-package-managing-package-versions/)

Quick start - Stand Alone Lite Mode Demo

The above example is a full example of using packrat with a simple analysis. What if you only wanted to capture the library and r info but not everything else - storing your packages and their dependencies in your private package library.

Then you can use Packrat lite by using *packrat::.snapshotImpl(snapshot.sources = False, ".")* to generate the packrat.lock file to list the R version used and precise package versions that were used to satisfy dependencies, including dependencies of dependencies. Packages are not stored. This will allow you to recreate the environment at a later date/time.

```
# devtools::install_github("rstudio/packrat")
# setwd("~/packrat_test_lite")
# Create/run Analysis file
# packrat::.snapshotImpl(snapshot.sources = FALSE, ".")
# packrat.lock is generated & has R version and package(s) information
```

RStudio Projects

If you are new to projects, please read this before taking a swing at packrat via the RStudio IDE :

- https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects (https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects)

RStudio projects make it straightforward to divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents. Many people as part of their workflow start new projects or revive old ones.

- https://blogs.uoregon.edu/rclub/2016/04/19/setting-up-a-lovely-new-project/ (https://blogs.uoregon.edu/rclub/2016/04/19/setting-up-a-lovely-new-project/)

Using Packrat with RStudio

- https://rstudio.github.io/packrat/rstudio.html (https://rstudio.github.io/packrat/rstudio.html)

You will need RStudio 0.98.945 or newer.

- File - New Project - New Directory - Empty Project - Use packrat with this project - Create Project

You are no longer in an ordinary R project; you're in a Packrat project with its own private package library and in "isolation".

To get the RStudio integrations, it is necessary to treat your project folder as an R Project.

When you create the project, RStudio runs the packrat::init() function to enter packrat mode, takes a snapshot of the package dependencies and places the binaries in the project folder under yourproject/packrat/src/.

init() runs restore() to apply the latest snapshot to the project folder. R will restart once this process is finished.

- http://t-redactyl.io/blog/2016/10/a-crash-course-in-reproducible-research-in-r.html (http://t-redactyl.io/blog/2016/10/a-crash-course-in-reproducible-research-in-r.html)

RStudio Packrat Notes

Auto-Snapshotting - RStudio works behind the scenes to fetch the package's sources and save them in Packrat. Packrat keeps track of any changes you make to your project dependencies without you needing to explicitly snapshot them yourself.

If you're bringing an existing Packrat project into RStudio, you don't need to do anything special to make it work. And if you're just starting a new project, you can start with Packrat right away.

If you have an existing RStudio project you'd like to bring under Packrat control, you can add it using the new Packrat section under Tools | Project Options.

Packages pane will show you the status of your project's private Packrat library.

RStudio tries to infer whether the appropriate action is a snapshot (i.e. update Packrat to match the library) or restore (i.e. update the library to match Packrat).

You can pick your library state or the Packrat state, but not some of each.

Remember, package removes aren't auto-snapshotted, so save your changes to Packrat once you've verified that your project's state is consistent after cleanup.

Demo - RStudio Full Mode

Let's try packrat::install_github("rstudio/rmarkdown"). Once the command finishes you'll see the Packages pane show the new packages.

What if I Want to Use my Own Local Repository?

A repository is a directory containg uninstalled R source files or platform-specific binaries. A repository cotains a PACKAGES file with important information about the repository's content.

A library is a directory containing installed R packages.

Though a repository and a library look very similar, they are two distinct entities.

Many people/organizations will also mirror the cran repo as explained here:

- https://cran.rstudio.com/mirror-howto.html (https://cran.rstudio.com/mirror-howto.html)
- https://cran.r-project.org/doc/manuals/R-admin.html#Setting-up-a-package-repository (https://cran.r-project.org/doc/manuals/R-admin.html#Setting-up-a-package-repository)

It is an easy way to grab all the packages and files from CRAN at once and then you set your repo to point to that new location as described here:

- http://docs.rstudio.com/ide/server-pro/r-sessions.html#cran-repositories (http://docs.rstudio.com/ide/server-pro/r-sessions.html#cran-repositories)

Other organizations do the above via Packrat.

- https://rstudio.github.io/packrat/custom-repos.html (https://rstudio.github.io/packrat/custom-repos.html)

- https://rstudio.github.io/packrat/limitations.html (https://rstudio.github.io/packrat/limitations.html)
- install a package that lives within a local repository with packrat::install_local() after following the steps above.

Setting Up a local repository info here:

- https://support.rstudio.com/hc/en-us/articles/115006298728-Package-Management-for-Offline-RStudio-Connect-Installations (https://support.rstudio.com/hc/en-us/articles/115006298728-Package-Management-for-Offline-RStudio-Connect-Installations)

Server Tips

- https://support.rstudio.com/hc/en-us/articles/200552316-Configuring-the-Server (https://support.rstudio.com/hc/en-us/articles/200552316-Configuring-the-Server)

External Libraries

You can add elements to the default LD_LIBRARY_PATH for R sessions (as determined by the R ldpaths script) by adding an rsession-ld-library-path entry to the server config file. This might be useful for ensuring that packages can locate external library dependencies that aren't installed in the system standard library paths. For example:

rsession-ld-library-path=/opt/local/lib:/opt/local/someapp/lib

Specifying R Version

By default RStudio Server runs against the version of R which is found on the system PATH (using which R). You can override which version of R is used via the rsession-which-r setting in the server config file. For example, if you have two versions of R installed on the server and want to make sure the one at /usr/local/bin/R is used by RStudio then you would use:

rsession-which-r=/usr/local/bin/R Note again that the server must be restarted for this setting to take effect.

Package Library Path

By default RStudio sets the R_LIBS_USER environment variable to ~/R/library. This ensures that packages installed by end users do not have R version numbers encoded in the path (which is the default behavior). This in turn enables administrators to upgrade the version of R on the server without reseting users installed packages (which would occur if the installed packages were in an R-version derived directory).

If you wish to override this behavior you can do so using the r-libs-user settings. For example:

r-libs-user=~/R/packages CRAN Repository

Finally, you can set the default CRAN repository for the server using the r-cran-repos setting. For example:

r-cran-repos=https://mirrors.nics.utk.edu/cran/ (https://mirrors.nics.utk.edu/cran/) Note again that the above settings should be specified in the /etc/rstudio/rsession.conf file (rather than the aforementioned rserver.conf file).

Packrat Bundle Sharing

My bundle is packrat_test-2017-04-24.tar.gz.

"Packrat, in essence, creates a large zip file with all of the libraries and settings used for a project. Users then send this entire file to their collaborators and collaborators load packages and libraries from that zip file. This ensures that the versions of packages used are the same across all collaborators."

- https://cereo.wsu.edu/2017/02/23/packrat-package-managing-package-versions/ (https://cereo.wsu.edu/2017/02/23/packrat-package-managing-package-versions/)

You can share your projects with “bundle”. Bundles help remove worry about conflicting package versions from person to person or computer to computer.

Example of a packrat bundle to reproduce : - https://ropensci.org/blog/2014/11/10/open-data-growth.html (https://ropensci.org/blog/2014/11/10/open-data-growth.html)

- You can push your RStudio project directory (packrat bundle) to a remote repo like Github. See here for my example:

https://github.com/philbowsher/Up-to-Bat-with-Packrat/blob/master/packrat_test-2017-04-24.tar.gz (https://github.com/philbowsher/Up-to-Bat-with-Packrat/blob/master/packrat_test-2017-04-24.tar.gz)

When you initialize a packrat project on github, it will let everyone clone it and call packrat::restore() to reproduce. The person starting with the bundle will also need to make sure their machine is able to build packages from source. For Windows, this means making sure you have Rtools:

- https://cran.rstudio.com/bin/windows/Rtools/ (https://cran.rstudio.com/bin/windows/Rtools/)

See here for other operating systems:

- https://support.rstudio.com/hc/en-us/articles/200486498-Package-Development-Prerequisites (https://support.rstudio.com/hc/en-us/articles/200486498-Package-Development-Prerequisites)

Via RStudio Connect

- https://support.rstudio.com/hc/en-us/articles/115003639788-Bundle-Downloads (https://support.rstudio.com/hc/en-us/articles/115003639788-Bundle-Downloads)

Collaboration

Packrat creates a large zip file with all of the libraries and settings used for a project. Users then send this entire file to their collaborators and collaborators load packages and libraries from that zip file. This ensures that the versions of packages used are the same across all collaborators.

You can send a “snapshot” or bundle (scripts plus private library). Collaborators use packrat::unbundle(“C:/phil/Misc/R_test/packrat_demo/packrat/bundles/packrat_demo-2017-02-21.tar.gz”, #location of the bundled project “C:/john/Misc/R_test/packrat_demo2”) #where we will unbundle the project.

Note: unbundling can take a few minutes!

There are two ways a bundled project can be “unbundled” :

- 1. If Packrat is installed on the target machine, the unbundle function can be used.
- 2. Otherwise, the bundled project can simply be untarred and un-gzipped using most file archiving projects

Packrat & Version Control

- If you’re collaborating using a version control system, Packrat will help keep your private libraries in sync. RStudio watches for changes to your Packrat lockfile. When a change from a version control system updates your Packrat lockfile, RStudio will prompt you to apply that change to your private library.

Using a packrat with Shiny Apps:

On the client, instead of running packrat::init() in the application directory, use:

packrat::.snapshotImpl(snapshot.sources = FALSE, “.”) packrat::set_opts(use.cache = TRUE)

Commit the results to Git

In your sandbox and production server, setup a directory as a global cache and ensure

R_PACKRAT_CACHE_DIR is set, see:

?packrat::set_opts

Setup a post-merge hook in your production / sandbox environment that runs:

packrat::init()

This will turn packrat on and create a unique library using the package described in the packrat/packrat.lock file committed in the Git repo. Packages will be added to the library from the global cache (if available) or else from CRAN.

Other:

- https://rstudio.github.io/shinytest/articles/ci.html (https://rstudio.github.io/shinytest/articles/ci.html)

Using a packrat with RStudio Connect:

RStudio Connect manages the deployment process through push-button publishing, but also includes support for bundle versioning and potentially staging - production promotions, see:

- https://support.rstudio.com/hc/en-us/articles/115003639788-Bundle-Downloads (https://support.rstudio.com/hc/en-us/articles/115003639788-Bundle-Downloads)
- https://support.rstudio.com/hc/en-us/articles/226871467-Package-management-in-RStudio-Connect (https://support.rstudio.com/hc/en-us/articles/226871467-Package-management-in-RStudio-Connect)
- https://support.rstudio.com/hc/en-us/articles/115006298728-Package-Management-for-Offline-RStudio-Connect-Installations (https://support.rstudio.com/hc/en-us/articles/115006298728-Package-Management-for-Offline-RStudio-Connect-Installations)
- http://docs.rstudio.com/connect/admin/package-management.html (http://docs.rstudio.com/connect/admin/package-management.html)
- https://support.rstudio.com/hc/en-us/articles/226871847-Process-management-in-RStudio-Connect (https://support.rstudio.com/hc/en-us/articles/226871847-Process-management-in-RStudio-Connect)
- https://github.com/rstudio/packrat/issues/291 (https://github.com/rstudio/packrat/issues/291)

Packrat Notes

- https://rstudio.github.io/packrat/limitations.html (https://rstudio.github.io/packrat/limitations.html)
- 1. http://lukesingham.com/using-package-management-in-r/ (http://lukesingham.com/using-package-management-in-r/)

When updating a package, without git, running packrat::restore() to downgrade a package requires the overwrite.dirty=TRUE option and seems to require switching off auto.snapshot. For a better understanding and control over packrat, turn off auto.snapshot.

- 2. https://bookdown.org/Tazinho/Tidyverse-Cookbook/rstudio-project-management.html (https://bookdown.org/Tazinho/Tidyverse-Cookbook/rstudio-project-management.html)

When you unbundle a project, which was created under an older (or newer) R version, then installed, packrat will warn you and install the older (or newer) R version when calling packrat::snapshot(). This is one among other reasons, why it is easier to work with packrat, while having access to the internet.

- 3. https://rpubs.com/cosmicBboy/nyc-r-stats-pegged1 (https://rpubs.com/cosmicBboy/nyc-r-stats-pegged1)

Unable to find source packages when restoring

> packrat::restore() Installing knitr (1.11) ... FAILED Error in getSourceForPkgRecord(pkgRecord, srcDir(project), availablePkgs, : Couldn't find source for version 1.11 of knitr (1.10.5 is current) Happens when there is a new version of a package on a R package repository like CRAN.

Solution 1: Use R's Installation Procedure

> install.packages() packrat::snapshot() Solution 2: Manually Download Source File

$ wget -P repo/packrat/src > packrat::restore() Python calls R

- 4. http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/ (http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/)

Note that because Packrat stores all of the packages used as well as their source code, projects using Packrat will require more disk space.

- 5. http://distancesampling.org/developer/develop/dev-packrat.html (http://distancesampling.org/developer/develop/dev-packrat.html)

Snapshot downloads the sources of the dependencies and installs them. This means you have to have C and Fortran compilers installed for this to work.

- 6. https://groups.google.com/forum/#!topic/packrat-discuss/RYgrosywULU (https://groups.google.com/forum/#!topic/packrat-discuss/RYgrosywULU)

Manually make sure that Rtools and the PATH are configured properly to match the packrat snapshot when transferring a package project to another machine.

- 7. http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/ (http://bconnelly.net/2014/07/creating-reproducible-software-environments-with-packrat/)

Snapshot() will make sure that each of the packages on which the project depends are available, which includes building them if necessary.

- 8. http://stackoverflow.com/questions/27282935/rstudio-packrat-and-knitr (http://stackoverflow.com/questions/27282935/rstudio-packrat-and-knitr)

If you activated Packrat for your project, and for some reason packrat::disable() did not clean out the .Rprofile generated, you may need to remove it manually.

- 9. https://github.com/rstudio/packrat/issues/31#issuecomment-208313921 (https://github.com/rstudio/packrat/issues/31#issuecomment-208313921) & http://stackoverflow.com/questions/36431714/loading-dependencies-from-package-internal-packrat-library (http://stackoverflow.com/questions/36431714/loading-dependencies-from-package-internal-packrat-library)

Why doesn't packrat support running in packages?There has been substantial progress on using packrat within the context of R package development, but there may yet be some edge cases to resolve (especially in the context of using packrat + R package + RStudio all together). Feel free to give it a whirl and raise an issue if you have trouble!

Other Package Tools Used for Managing Packages & Package Dependencies

- 1. https://cran.r-project.org/web/packages/miniCRAN/index.html (https://cran.r-project.org/web/packages/miniCRAN/index.html)
- 2. checkpoint Package: See https://mran.microsoft.com/rro/#repos (https://mran.microsoft.com/rro/#repos)
- https://cran.r-project.org/web/packages/checkpoint/index.html (https://cran.r-project.org/web/packages/checkpoint/index.html)
- 3. https://github.com/richfitz/remake (https://github.com/richfitz/remake)
- 4. Switchr - handy abstraction over .libPaths(), lib.loc and other lower level library settings. Helpful in creating and managing libraries.

Other Tools

- 1. Docker containers for projects
- https://hub.docker.com/r/rocker/ (https://hub.docker.com/r/rocker/)
- https://vsoch.github.io/rawr/2017/explore-drug-database/ (https://vsoch.github.io/rawr/2017/explore-drug-database/)

Docker allows you to write a recipe for what is important in your project and then you can distribute the resulting images. Docker works well on your local machine, on a cluster or on Amazon etc.

"With a syntax that is simpler than other provisioning tools (e.g. Chef, Puppet, Ansible) or Continuous Integration (CI) platforms (e.g. Travis CI, Shippable CI); users need little more than a basic familiarity with shell scripts and a Linux distribution software environment (e.g. Debian-based apt-get) to get started writing Dockerfiles."

- https://arxiv.org/pdf/1410.0846.pdf (https://arxiv.org/pdf/1410.0846.pdf)
- https://mattwilcox.net/web-development/vagrant-puppet-docker/ (https://mattwilcox.net/web-development/vagrant-puppet-docker/)
- https://superuser.com/questions/743448/is-there-any-reason-to-use-puppet-alongside-with-docker (https://superuser.com/questions/743448/is-there-any-reason-to-use-puppet-alongside-with-docker)
- 2. make

Make is a language-agnostic build management utility for *nix.

- http://opr.princeton.edu/workshops/Downloads/2017Jan_ReproducibleReasearchToolsPratt.pdf (http://opr.princeton.edu/workshops/Downloads/2017Jan_ReproducibleReasearchToolsPratt.pdf)
- https://www.rstudio.com/resources/videos/user-lightning-talks/ (https://www.rstudio.com/resources/videos/user-lightning-talks/)
- Watch first presentation
- http://will-landau.com/2016/06/14/workflow/ (http://will-landau.com/2016/06/14/workflow/)

Interesting

1. How to Set Up a Custom CRAN-like Repository http://rstudio.github.io/packrat/custom-repos.html (http://rstudio.github.io/packrat/custom-repos.html)
2. https://support.rstudio.com/hc/en-us/articles/216528108-Deploying-packrat-projects-to-Shiny-Server-Pro?mobile_site=true (https://support.rstudio.com/hc/en-us/articles/216528108-Deploying-packrat-projects-to-Shiny-Server-Pro?mobile_site=true)
3. https://groups.google.com/forum/#!forum/packrat-discuss (https://groups.google.com/forum/#!forum/packrat-discuss)
4. https://discuss.ropensci.org/t/reproducibility-in-r-package-building-with-travis-and-packrat/445 (https://discuss.ropensci.org/t/reproducibility-in-r-package-building-with-travis-and-packrat/445)
5. https://www.jasperginn.nl/shiny-server-series-pt1/ (https://www.jasperginn.nl/shiny-server-series-pt1/)
6. https://rviews.rstudio.com/2017/04/19/r-for-enterprise-understanding-r-s-startup/ (https://rviews.rstudio.com/2017/04/19/r-for-enterprise-understanding-r-s-startup/)
7. http://www.baselr.org/presentations/2017/03/BaselR_-_Reproducible_analysis_-_Joris_Muller_-_20170307.pdf (http://www.baselr.org/presentations/2017/03/BaselR_-_Reproducible_analysis_-_Joris_Muller_-_20170307.pdf)