PhUSE US Connect 2021
Paper for Presentation AS01

Managing Multiple Versions of R in a Clinical Environment

Phil Bowsher, RStudio; Boston, USA,
Sean Lopp, RStudio; Boston, USA,
Alex Gold, RStudio; Boston, USA

Abstract

Interest in using R for clinical trials has increased significantly and there are a few questions that arise as the environments are being developed. One question asked by IT or the validation team is - *how to handle multiple versions of R?* This paper will offer some information on strategies for managing multiple versions of R in a regulated environment.

The PhUSE paper TT11 (How Do I Pick a R Package for My Clinical Workflow?) provides valuable information for which this paper builds on:

https://github.com/philbowsher/phuse-2019-r-packages/blob/master/How%20Do%20I%20Pick%20a%20R%20Package%20for%20My%20Clinical%20Workflow_.pdf

This paper will build off the information in the previously mentioned paper and will discuss approaches for managing multiple versions of R in a clinical environment to help IT stay abreast of changes.

Introduction

This paper is for clinical IT managers and R administrators that are implementing an R environment and have wondered how best to approach multiple versions of R. This is not a general introduction to R, but instead valuable information on how to approach R language upgrades and changes for clinical IT teams in using R and looking to understand how to handle multiple versions of R. This paper addresses the use of R on a server as opposed to individual laptops.

In many cases, organizations developing projects in R can take lessons from software development. Data science and analytic code are different from pure software development, so it is inappropriate to adopt software development practices in whole cloth, but there are many useful parallels.

Like in many software development use cases, the R developers often desire complete flexibility while IT wants almost complete stability. These two ideas can be challenging to balance. This challenge is especially keen when upgrading R and associated challenges, as it is the moment when things are most likely to break.

Why do we need multiple versions of R?

The open source data science ecosystem is developing very quickly. This means that new package and R versions often include new and exciting tools, techniques, computational methods, and syntax. Trying to keep R and package versions constant is to stand in opposition to those changes, slowing down the work of R users. This paper is to help *IT Admins understand strategies to be comfortable with and embrace these changes*, while also helping to understand how to transition code from development to something that's persistent and hardened (production).

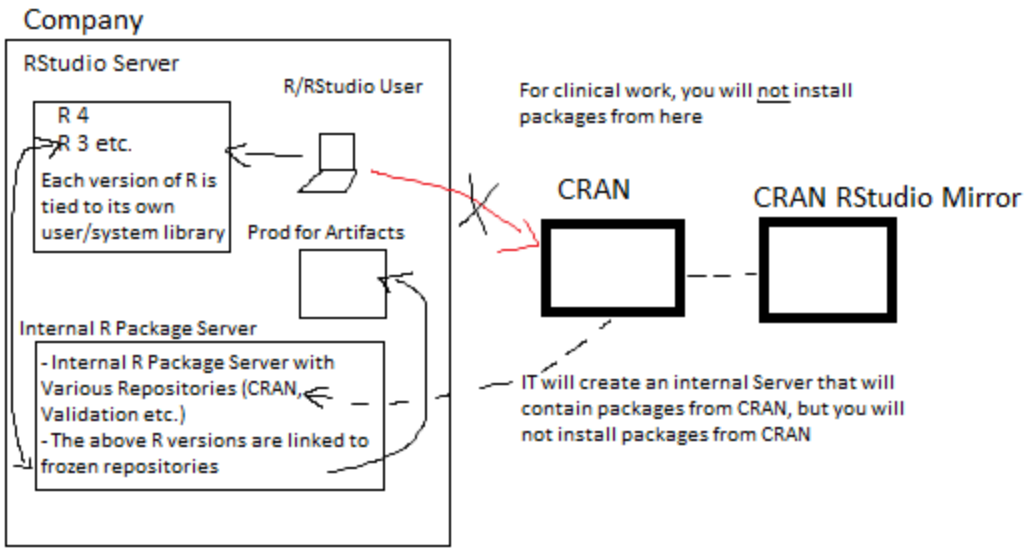
For clinical environments, it is recommended to follow a *risk-based approach* as defined here:

https://www.pharmar.org/white-paper/

Understanding how the above concept applies to multiple versions of R can be challenging and this paper hopes to add some helpful context.

Core Concepts to Understand

When it comes to using R, there are a few core ideas that are important to understand as outlined in the diagram below:



- IT should not upgrade R but rather install multiple versions of R on the same RStudio Server.
- An R package library is tied to a specific version of R, so different R versions have distinct package libraries.
- Users will not install packages from repositories (CRAN, BioConductor, PyPI, etc), but rather IT will bring repositories into the organization.
- Organizations will have multiple repositories (CRAN, validation etc.) aligning to different use cases -- e.g. an “open” repository for bleeding edge development and a “validated” repository for formal development.
    - Each repository can be frozen to a point in time, to ensure complete reproducibility of the package state at that point in time.
- Users will install packages from the frozen repositories states for the specific version of R they are using.
    - IT can set which frozen state is used for each version of R, most commonly by using the Renviron.site file.
- The Production environment will house the artifacts created (Shiny apps, Reports) and these will also use the packages from the frozen repositories. Each artifact should be housed in a sandboxed production environment on the Production server.

Installing Multiple Versions of R

Step 1: Create a R Version and Package Plan

In the clinical space, a key regulatory requirement for pharmaceutical companies is being able to reliably reproduce the tables, figures, listings (TFLs), and other data used in a submission to the FDA. The validation, IQ, OQ, PQ steps are codified in “standard operating procedures” (SOPs).

In the open source ecosystem there is a combination of products (RStudio), core software (R), and add-ons (packages); all of which receive updates at different time scales. It is important to have a plan on how to handle changes. These layers can have inter-dependencies as well. For example, newer RStudio versions can recommend newer R packages that the IDE relies on for specific functionality.

Change management plans must also account for the environment being implemented. For example, server implementations can be via local servers, cloud persistent servers, or docker containers.

It is a best practice not to upgrade R but rather to install multiple versions on the same server.

Before we discuss strategies for managing multiple versions of R, let us first detail the steps to implement multiple versions of R on a Server.

A new version of R is released via CRAN (The Comprehensive R Archive Network) about once a year, with patch releases coming out every 2-6 months. Below are the release dates for the last 3 years.

R 4.0.4 (February, 2021)
R 4.0.3 (October, 2020)
R 4.0.2 (June, 2020)
R 4.0.1 (June, 2020)
R 4.0.0 (April, 2020)
R 3.6.3 (February, 2020)
R 3.6.2 (December, 2019)
R 3.6.1 (July, 2019)
R 3.6.0 (April, 2019)
R 3.5.3 (March, 2019)
R 3.5.2 (December, 2018)
R 3.5.1 (July, 2018)
R 3.5.0 (April, 2018)
R 3.4.4 (March, 2018)

RStudio provides pre-compiled R binaries to help with the installation of different versions of R side by side in Linux environments (servers). The instructions below outline this process of installing R and can be used to install multiple versions:

https://docs.rstudio.com/resources/install-r/

The binaries at the repository below were built to be used side-by-side in RStudio:

https://github.com/rstudio/r-builds

Further information is here:

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

Files to customize R's behavior
When working with R, there are a *few files that are good to use and understand*. These are:

*Add the rspm stuff here on the rspm site

Renviron.site
Allows users to specify environment variables that should be available to any R session. The Renviron.site file is located at R_HOME/bin/R/etc. The system file is useful for specifying environment variables that should be available for all users accessing this version of R. For example, you might specify the PATH or set common ODBC settings.

User created .Renviron files
These files have the same format as the Renviron.site file but are placed in either a user home directory (to apply to all projects) or their project directory (to apply to only a specific project). Often user's will take advantage of these files to specify secrets such as passwords or API keys. For example, users may place a token as an environment variable in the `.Renviron` file. The easiest way to do so is using the usethis package:

https://usethis.r-lib.org/

```r
# install.packages(usethis)
usethis::edit_r_environ()
```
Then place the following inside.

```r
Software_API_KEY = "<my_api_key>"
```
Rprofile.site
Similar to the environment file, the site-wide R profile allows you to supply R code that will be executed prior to the R session starting for any user accessing this version of R. The file is located at R_HOME/bin/R/etc

The R profile is an easy way to set R specific options that can apply for all users and this will be important as we discuss below methods for managing multiple versions of R.

User created Rprofile.site files
Just like with the Renviron file, users can create their own default R profile by creating a file called .Rprofile and saving it in their home directory (to apply to all projects) or their project directory (to apply to only a specific project).

You can read more about these files here:

https://rviews.rstudio.com/2017/04/19/r-for-enterprise-understanding-r-s-startup/

https://support.rstudio.com/hc/en-us/articles/360047157094-Managing-R-with-Rprofile-Renviron-Rprofile-site-Renviron-site-rsession-conf-and-repos-conf

Environments

It is important to understand that not all clinical environments are the same. Some organizations will have a combination of stage, test, prod, data science labs, cloud images, and desktop users environments. For this paper, we discuss briefly managing multiple versions of R in development environments, but then devote the rest of the paper to Production.

Step 2: Pick versions of R

In both validated and research environments, *it is important for IT to align on specific versions of R* that they want to support and maintain. The main difference being that in research environments, more versions of R may be available and there may be a more regular process for allowing newer versions of R to be added. In validated environments, often the version of R is locked-down and very careful consideration is given to adding a new R version. Assume upgrading R will break your code. It is not uncommon for validated environments to have the same version of R for many years.

It is recommended not to upgrade R, but rather maintain multiple versions of R side by side. This strategy preserves past versions of R so you can manage upgrades and keep your code, apps, and reports stable over time.

Whenever you add a new version R, you have to install packages afresh for that new version of R. So adding a new version of R has two steps:

A. Add new version of R via the steps above
B. Reinstall/add packages

Step B can be done in a few ways. If you set the repository as described above, users can install packages themselves, with the package versions locked to the frozen repository set in the REnviron.site or RProfile.site. It is also possible to install an entire package set all at once using the renv package.

Development

Some pharmas have higher and lower environments, and different standards may apply to “data science work”, “early stage research”, and “clinical submissions”. For research and some data science work, managing multiple versions of R can be easier to approach as users can toggle to different versions and then use tools like renv to create reproducible environments. https://github.com/rstudio/renv

RStudio recommends organizations roll out new versions of R (and new package sets) every 4-6 months for most environments except for GxP, where new R versions are often only added every 1-2 years.

In most development environments, the controls around validation and qualification are less strict and IT R Admins will usually allow users the freedom to use an internal CRAN repository as outlined below.

Step 3: Create an Internal CRAN and/or a curated repository

For most clinical environments, it is best to have a dedicated team or R admin that will *create an internal repository of packages*. This repository often will replicate CRAN and allow users to install packages from the central and internally located location. This process helps IT to organize and centralize packages across the various clinical teams, departments and organizations. Moreover, many clinical environments are behind a firewall and offline so access to CRAN is not possible. A local R package repository can be used to share local packages as many drug development organizations will create internal packages such as the validation tests or report formatting routines. IT may also want to restrict package access or simply limit package downloads to a specific repository. These strategies are important for maintaining the validation and qualification processes as codified in the standard operating procedures created by the quality assurance personnel.

If there is any questions where packages are being installed from, a user can run the following code in the R console which will display the repository being used:

options('repos')

It is not recommended to install older versions of packages into your current R installation. There are a few potential issues that may arise with installing older versions of packages:

- You may be losing functionality or bug fixes that are only present in the newer versions of the packages.
- The older package version needed may not be compatible with the version of R you have installed.

Instead, if your old code relies on older packages, simply continue to use the old code with the older version of R. This is the beauty of having multiple R versions. If you want to bring old code up to date with a new version of R, expect that process to include upgrading packages and fixing your code.

Guidance documents for the use of affiliated R packages in regulated clinical trial environments can be found at the bottom of this page:

https://rstudio.com/solutions/pharma/

Advice on validating internal packages can be found on the schedule "R Package Validation Framework" by Ellis Hughes:

https://rinpharma.com/rpharma_2020/
https://www.youtube.com/watch?v=zEH-6Ik-5h8

Traditional Method to Install Packages by Users
Usually users understand what R packages are, but lack an understanding of R versions, repositories, mirrors and package versions. This is often because users learn R on desktop or laptop computers where they are the administrator and the understanding of such concepts is abstracted. If a user in this environment needs a package, they simply run:

install.packages("gt")

In a desktop environment, this will install a package from CRAN, the base R function install.packages is executed in R.

Behind the scenes, install.packages consults the current value of options('repos') to find the appropriate repository. By default, this value is set to RStudio's default CRAN mirror. It can also be configured inside the RStudio IDE's settings, or in an RProfile or REnviron file for persistence across users and projects.

To set it in an RProfile or REnviron, you would include a line like:
options(repos = c(CRAN = "http://cran.rstudio.com"))

After you specify the mirror to use, packages will be installed from that location every time install.packages is used. It is also possible (though inadvisable for reproducibility) to override the repositories and other parameters of install.packages with arguments to the function call.

When the base R function install.packages is executed, a connection is made over HTTP or HTTPs to the specified CRAN mirror. Then a file is downloaded and installed to your environment. The file type is dependent on your operating system. The file is unzipped/unpacked and installed.

Where are R packages installed?

To use the package and its functions, **library(<packagename>)** needs to be run in the active R session. Users generally have a poor mental model of what actually happens when this command runs.

R packages are installed into libraries, which are directories in the file system containing a subdirectory for each package installed there. **These libraries can be common/system libraries, user libraries, or project-level libraries**.

You can see all the library paths installed for your user by issuing the .libPaths() command in the R terminal:

```
.libPaths()
```

By default, both install.packages always installs to the first element of .libPaths(), and library looks for packages in the libraries that appear in .libPaths() in the order they appear.

Usually, two steps are required to use an R package:

The package is installed from a repository into a library using install.packages().
The package is loaded from the library for an analysis using library().

What is the difference between the system and user library?

This question is often asked by people managing a central linux server used by many people running r code.

Whether it is a system or a user library, it is tied to a specific version of R; and different versions of R will have different libraries and require different user libraries and package installations.

Users installing packages from an internal validated R repository, will install packages likely into their user library. Some organizations may decide to install the validated packages ahead of time for users, and these packages will be available as system packages, available to all users, as discussed below.

Production/Validated Environment
New Methods to Install/Use Packages by Users on a Server used in a GXP environment
Step 4: *Freeze Repository and tie it to the R version*
By now you should see that the library of installed packages is very important, and tied to the version of R. How do you ensure the correct packages are installed in that library? This is where the admin management of repositories comes into play.

However packages are installed, it is recommended to install from a frozen repository. By using a frozen repository specific to the R version, the same package version will be installed no matter who runs the install.packages command or when they do so.

In either case, you will begin by installing a version of R and setting the library to a frozen one:

Install a version of R. This results in a versioned system library:
/opt/R/3.4.4/lib/R/library

Create or edit the Rprofile.site file, to set the repo option for this version of R to point to a frozen repository.

```
# /opt/R/3.4.4/etc/Rprofile.site
local({
  options(repos = c(CRAN = "https://r-pkgs.example.com/cran/128"))
})
```

There are 2 common approaches for how to actually install the packages:

A. Allow users the freedom to install packages from a Frozen repository.

Users in this setup will login to RStudio and find an empty user library and a system library that only includes the base R required packages. Users will install the needed packages from the defined IT internal repository which often consist of validated packages. Users in this environment can use tools like renv to manage package dependencies specific to the work they are doing.

Often users don't know how to define or point to the validated repository of packages. *This is why IT sets the frozen repository ahead of time via the .Rprofile.site.* In this way, users can run the familiar install.packages function, but the source of the packages will be the validated repository, and not public CRAN.

B. Admins install packages via a Frozen Repository

This process is very similar to the one above, but instead of users doing their own installs, IT will do the installs ahead of time to a shared library. It is worth noting that it is impossible to stop users from running the install.packages command themselves. The way to stop users from installing unsanctioned packages or package versions is to only allow network access to a repository with the appropriate packages.

Run R as root, and install the desired baseline packages. Overtime, as requests for new packages come in, install them in the same way. Consistency is guaranteed because you are always installing from the same frozen repository.

sudo /opt/R/3.4.4./bin/R -e 'install.packages("ggplot2")'
Users access packages on the server without any need to install, e.g.: library(ggplot2)

(Optionally) Disable the user option to change the repository setting and discourage package installation.

```
# /etc/rstudio/rsession.conf
allow-r-cran-repos-edit=0
allow-package-installation=0
```

Docker
Docker can be used alongside the shared baseline strategy to ensure that rebuilding a Docker image always returns the same sets of packages. This process requires knowledge of Docker and other tools. The Phuse paper "R Clinical Cloud Strategy Developments" will discuss these topics in more detail. The key item that aligns to the topic above is either to include package installation in the Dockerfile which embeds the packages into the image. A second approach is to add appropriate R packages when the container is run. Regardless, the same advise from above applies: Install packages from a frozen repository that aligns to the version of R used.

Some R packages have requirements beyond the R code itself. For example, the RJava package requires a version of Java on the system. This will not be installed by the install.packages command and must be separately installed. This is one common use case for docker in clinical environments.

If you are installing packages as part of a Dockerfile, it is important to make sure you are setting the correct frozen repo in the install.packages command.

```
#  install from a versioned repo
RUN R -e 'install.packages(..., repo = "https://rpkgs.company.com/frozen/repo/123")'
```

Learn more here
pull in a manifest file and restore it
COPY renv.lock ./
RUN R -e 'renv::restore()'

Desktop Users
While it is a best practise to have users on a dedicated server, some organizations still have users using R on desktops. This can be for various reasons difficult to maintain for IT, however, the advice above still stands; *standardize on a version of R, and have a repository of validated/clinical packages and have users only use these packages.* Often the challenge is the user knowing how to change the default location for installing packages so user training may be required.

Many organizations manage a mix of RStudio Server and Windows RStudio desktop users. This document outlines 2 ways for corporate IT in managing Windows RStudio Desktop users:

A. Create an internal repository to centralize R packages across the entire organization. Packages in this repository will be managed, controlled and supported by corporate IT. RStudio desktop installations (open source or professional) will be configured to point to the internal repository for the Windows binaries of packages. The internal repository should be automatically versioned (frozen repositories). Each version of R supported internally will specify which version of the repository users will use via the Rprofile.site file.

Step 5: Production for Clinical Artifacts
It is recommended that statistical programmers write the code; then harden the result. Development environments (like RSP and RStudio Desktop) are called development environments because that is where code is going to be developed, which by definition indicates that this is a changing environment. Once code has completed development, it can be hardened and placed in a persistent environment.

There are a few key aspects of the production environment:
1) It allows each code artifact to run in a sandboxed environment with a set of packages particular to that artifact.
2) Package sets and versions of R do not change over time.

There are a variety of ways to create this production environment. RStudio Connect is a useful tool in this regard. It allows for the installation of multiple versions of R and the deployment of many pieces of content, each with its own unique package environment self-contained in an artifact-specific manifest.

It is a best practice to use some kind of infrastructure management tool to manage the server on which RStudio Connect sits and ensure the operating system and system libraries are reproducible as well. Docker is a common deployment target for RStudio Connect for this reason -- but other infrastructure-as-code tooling can serve the same role.

Validated Recap:

Step 1: Create a R Version and Package Plan

Step 2: Install R versions (separately) that are supported
(Don’t upgrade R but rather add new versions of R)

Step 3: Create an Internal CRAN and/or a curated repository (may be validated)

Step 4: Freeze Repository and align to your specific version(s) of R to that internal repo
The Validated repo is versioned and each version of R will specify which version of the repo to use

Step 5: Production for Clinical Artifacts
Deploy content that needs to be stable to a production environment like RStudio Connect. Do not rely on development tools like the RStudio IDE for long term production code.

Conclusion

This paper was to highlight strategies for managing multiple versions of R to help R statistical programmers. As interest in R for clinical trials increases by clinical statistical programmers, it is important to understand how this tooling can be used. The information outlined in this paper highlights practical approaches for IT to use.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Phil Bowsher
phil@rstudio.com