PharmaSUG 2022 - Paper SI-187

Creating an Internal Repository of Validated R Packages

Phil Bowsher, RStudio PBC;
Devin Pastoor, RStudio PBC;
Cole Arendt, RStudio PBC

Abstract

The R in Pharma conference has highlighted over the years that many pharmaceutical organizations and the FDA are using R for various areas of drug development including late stage clinical trials reporting. Recent R package validation efforts in the clinical trials space have brought awareness to IT and quality teams about the rigor that goes into developing and maintaining public R packages. You can read more about these efforts in the PharmaSUG 2021 Paper, SI-203: **R Validation: Approaches and Considerations**:

https://www.pharmasug.org/proceedings/2021/SI/PharmaSUG-2021-SI-203.pdf

Introduction

Approaching validation uniformly for any programming language can be challenging as many organizations have different processes and levels of acceptance. This paper will highlight steps of **how organizations can set up an Internal Repository of Validated R Packages** and discuss best practices and advice for managing the repository as the use case develops overtime for clinical workflows.

Many IT and quality teams use the RStudio guidance documents that detail the use of affiliated R packages in regulated clinical trial environments:

https://www.rstudio.com/assets/img/validation-tidy.pdf

https://www.rstudio.com/assets/img/validation-shiny-rmd.pdf

These documents in total cover about 200 R packages built and maintained by RStudio. Organizations then add other public R packages based on a risk-based approach as documented here:

“A Risk-Based Approach For Assessing R Package Accuracy Within A Validated Infrastructure”

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

You can see case studies contributed by companies and individuals (Roche, Novartis & merck), who are implementing a risk-based approach to validate R packages at the link below:

https://github.com/pharmaR/case_studies/

In addition to RStudio and other public R packages, organizations will add internal packages. It is important to treat internal packages and UDFs with a similar risk-based approach as defined above. Internal functions and packages should include documentation, tests, release control, and other processes that are applied to validate external packages. In fact, because internal code is used by a smaller audience than most external open source packages, it is more important to validate internal UDFs than existing CRAN packages.

To help formalize this approach, Ellis Hughes is working on a R Package validation framework as highlighted at his 2020 R in Pharma conference talk here:

https://rinpharma.com/publication/rinpharma_145/

This process takes advantage of many existing tools for creating robust R packages outside of the validation space such as testthat, rmarkdown and roxygen2.

These documents, and the implemented risk-based approach, **allow organizations to establish a shared baseline of existing packages that are deemed validated for internal use**.

Once a set of packages has been approved through the processes outlined above, many IT organizations will make the collection packages available to users through an internal repository. The rest of this paper **will highlight the steps for creating an internal repository of validated R packages at an organization**. During the steps, we will discuss best practices for managing the repositories such as releases of packages being aligned with IT support for specific versions of R.

This paper is for clinical IT managers and R administrators that are implementing an R environment and have wondered how best to approach supporting controlled package requirements. 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.

R Package Strategy

One of the most important aspects of using R in pharma is having a package strategy that is understood by developers and statistical programmers. You can read more about this here:

https://environments.rstudio.com/reproduce.html

Key aspects are:

- Aligning on supported R versions
- Not upgrading R (instead, install multiple versions)
- Not installing packages from CRAN but using IT owned internal repositories
- Aligning historic R versions to frozen package snapshots

Step 1 - Installing RStudio Package Manager (RSPM)

It is important to understand that not all clinical environments are the same. Organizations will generally have some combination of stage, test, prod environments, as well as use other data science platforms,, 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.

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. Upgrades to R or packages may break existing code. It is not uncommon for validated environments to have the same version of R and packages available for many years to reproduce older analyses.

For step 1, you will need to install RStudio Package Manager (RSPM) using the linux command line as explained here:

https://docs.rstudio.com/rspm/admin/getting-started/installation/

Be sure to run the installation instructions for your Linux distribution. To install on linux, as of 2022-05-01, Ubuntu 18.04 would look like:

Terminal:

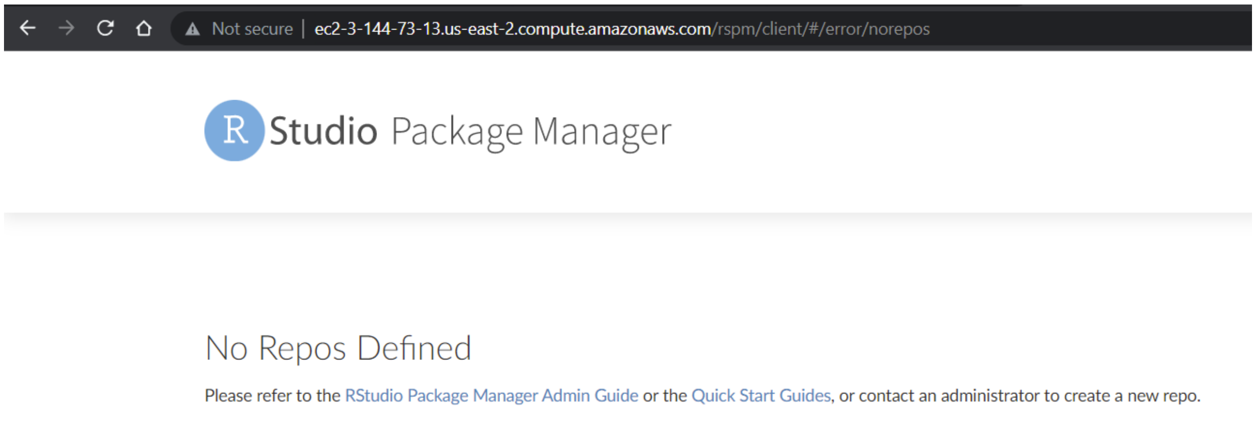
```
sudo apt update
sudo apt install gdebi-core
wget https://cdn.rstudio.com/package-manager/ubuntu/amd64/rstudio-pm_2022.04.0-7_amd64.deb
sudo gdebi rstudio-pm_2022.04.0-7_amd64.deb
```

These commands will install the RSPM into */opt/rstudio-pm*.

Below are the commands to install it vi aDocker:

```
mkdir -p ./rstudio-pm-demo/
docker run --privileged -it --rm --name=rstudio-pm -v $(pwd)/rstudio-pm-demo:/data/ -p 4242:4242 -e
RSPM_LICENSE ghcr.io/rstudio/rstudio-package-manager:late
```

After RSPM is installed, IT and users can view the software via a web browser, which provides you a user-friendly interface for viewing information about packages, package READMEs, metrics, etc. At this point though, no packages are yet available and will be installed in the steps below:


Not secure | ec2-3-144-73-13.us-east-2.compute.amazonaws.com/rspm/client/#/error/norepos

RStudio Package Manager

No Repos Defined

Please refer to the RStudio Package Manager Admin Guide or the Quick Start Guides, or contact an administrator to create a new repo.

For administrative changes, IT will use the Command-line Interface CLI.

https://docs.rstudio.com/rspm/admin/getting-started/installation/#getting-started-cli

STEP 2 - SERVING CURATED CRAN SUBSETS IN YOUR VALIDATED REPO

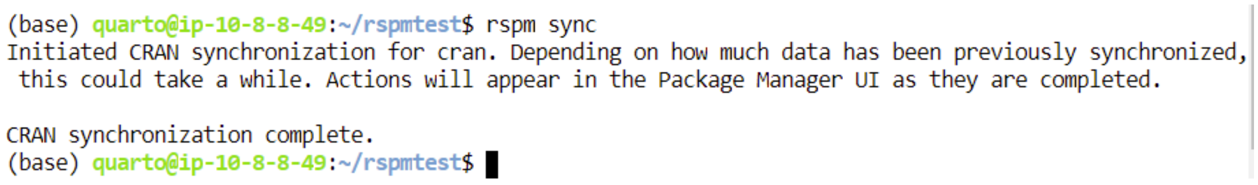
The link below describes at a high-level how to create a curated subset of CRAN that will be the Validated repo:

https://docs.rstudio.com/rspm/admin/appendix/source-details/#curated-cran-source

At this point, RSPM will be installed and now you will want to add the internal Validated repository. At the following link you will find the commands to setup the Validated internal repository:

These steps are outlined in,
https://docs.rstudio.com/rspm/admin/getting-started/configuration/#quickstart-curated-cran as well as a representative example provided below:

Terminal:
# Ensure you have CRAN metadata:
rspm sync --type=cran

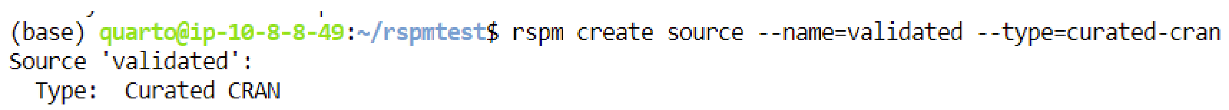


```
(base) quarto@ip-10-8-8-49:~/rspmtest$ rspm sync
Initiated CRAN synchronization for cran. Depending on how much data has been previously synchronized,
 this could take a while. Actions will appear in the Package Manager UI as they are completed.

CRAN synchronization complete.
(base) quarto@ip-10-8-8-49:~/rspmtest$
```

Now we will need to create a "source" which is a collection of packages. This is a "placeholder" / empty bucket for us to put packages in.

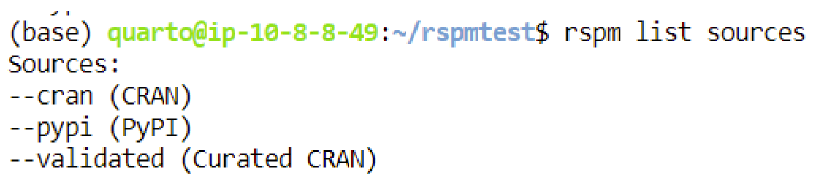
#Create the curated-cran source from a specific starting date:
rspm create source --name=Validated --type=curated-cran --snapshot=2020-07-06



```
(base) quarto@ip-10-8-8-49:~/rspmtest$ rspm create source --name=validated --type=curated-cran
Source 'validated':
  Type:  Curated CRAN
```

You can view the sources with:

rspm list sources



```
(base) quarto@ip-10-8-8-49:~/rspmtest$ rspm list sources
Sources:
--cran (CRAN)
--pypi (PyPI)
--validated (Curated CRAN)
```

To add a few packages, the following command will work:

# Specify the top-level packages you want to add:
rspm add --packages=ggplot2,shiny --source=subset

Curated CRAN source "pulls" packages from the available CRAN source which is why the command requires the --snapshot flag. This capability is provided through the RSPM online sync service to minimize the amount of data that has to be copied to your RSPM instance, though it can also be completely mirrored for air-gapped solutions.

As discussed above, many organizations **fill the Validated repo with the RStudio packages outlined in the RStudio guidance documents** that create the shared baseline.

You can add these packages to the Validated repository "Validated" by adding a large number of packages that are specified in a file. You will find a sample CSV file that contains RStudio packages outlined in the validation documentation detailed above here:

https://github.com/philbowsher/Creating-an-Internal-Repository-of-Validated-R-Packages/blob/main/example.csv

To create your own file, make a csv containing one package name per line. For example, /tmp/packages.csv:

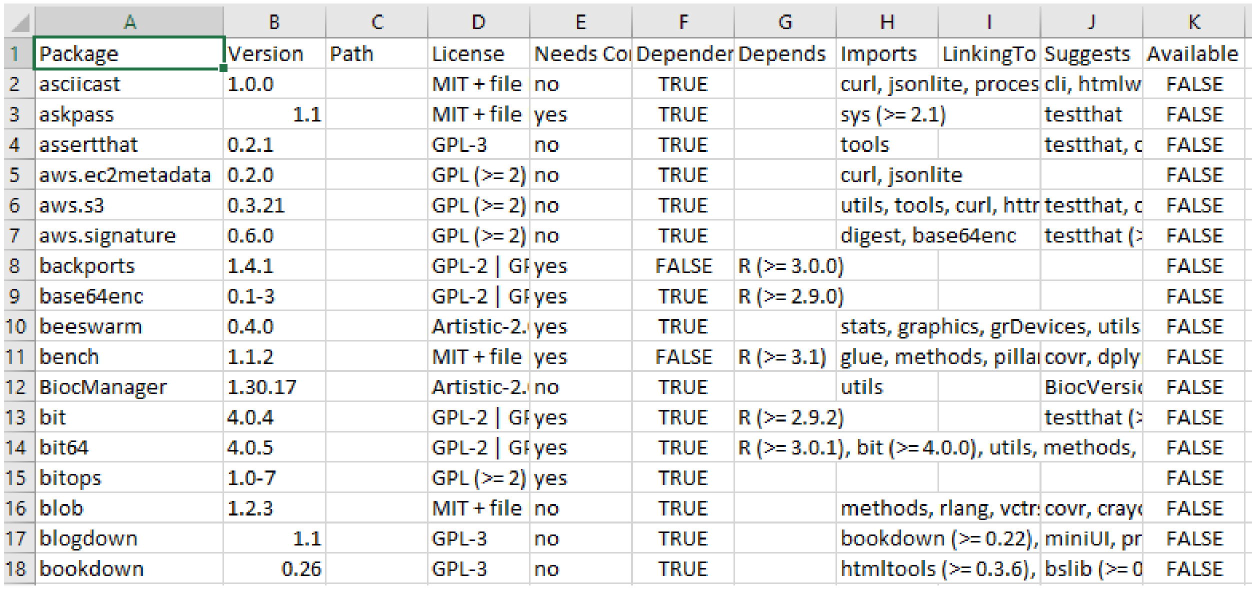
plumber
shiny
ISLR

A sample script to create the csv file is contained here:

https://github.com/philbowsher/Creating-an-Internal-Repository-of-Validated-R-Packages/blob/main/steps.sh

rspm add --source=validated --include-suggests --csv-out example.csv
--packages=tidyverse,tidymodels,gt,shiny,rmarkdown

The file will look like this:



	A	B	C	D	E	F	G	H	I	J	K
1	Package	Version	Path	License	Needs Coı	Dependeı	Depends	Imports	LinkingTo	Suggests	Available
2	asciicast	1.0.0		MIT + file	no	TRUE		curl, jsonlite, proces		cli, htmlw	FALSE
3	askpass	1.1		MIT + file	yes	TRUE		sys (>= 2.1)		testthat	FALSE
4	assertthat	0.2.1		GPL-3	no	TRUE		tools		testthat, c	FALSE
5	aws.ec2metadata	0.2.0		GPL (>= 2)	no	TRUE		curl, jsonlite			FALSE
6	aws.s3	0.3.21		GPL (>= 2)	no	TRUE		utils, tools, curl, httr		testthat, c	FALSE
7	aws.signature	0.6.0		GPL (>= 2)	no	TRUE		digest, base64enc		testthat (>	FALSE
8	backports	1.4.1		GPL-2 | GI	yes	FALSE	R (>= 3.0.0)				FALSE
9	base64enc	0.1-3		GPL-2 | GI	yes	TRUE	R (>= 2.9.0)				FALSE
10	beeswarm	0.4.0		Artistic-2.	yes	TRUE		stats, graphics, grDevices, utils			FALSE
11	bench	1.1.2		MIT + file	yes	FALSE	R (>= 3.1)	glue, methods, pillaı		covr, dply	FALSE
12	BiocManager	1.30.17		Artistic-2.	no	TRUE		utils		BiocVersi	FALSE
13	bit	4.0.4		GPL-2 | GI	yes	TRUE	R (>= 2.9.2)			testthat (>	FALSE
14	bit64	4.0.5		GPL-2 | GI	yes	TRUE	R (>= 3.0.1), bit (>= 4.0.0), utils, methods,				FALSE
15	bitops	1.0-7		GPL (>= 2)	yes	TRUE					FALSE
16	blob	1.2.3		MIT + file	no	TRUE		methods, rlang, vctr:		covr, cray	FALSE
17	blogdown	1.1		GPL-3	no	TRUE		bookdown (>= 0.22),		miniUI, pr	FALSE
18	bookdown	0.26		GPL-3	no	TRUE		htmltools (>= 0.3.6),		bslib (>= 0	FALSE

To complete this operation, execute this command with the --commit and --snapshot=2022-05-18 flags.

rspm add --source=validated --include-suggests --csv-out example.csv --packages=tidyverse,tidymodels,gt,shiny,rmarkdown **--commit --snapshot=2022-05-18**

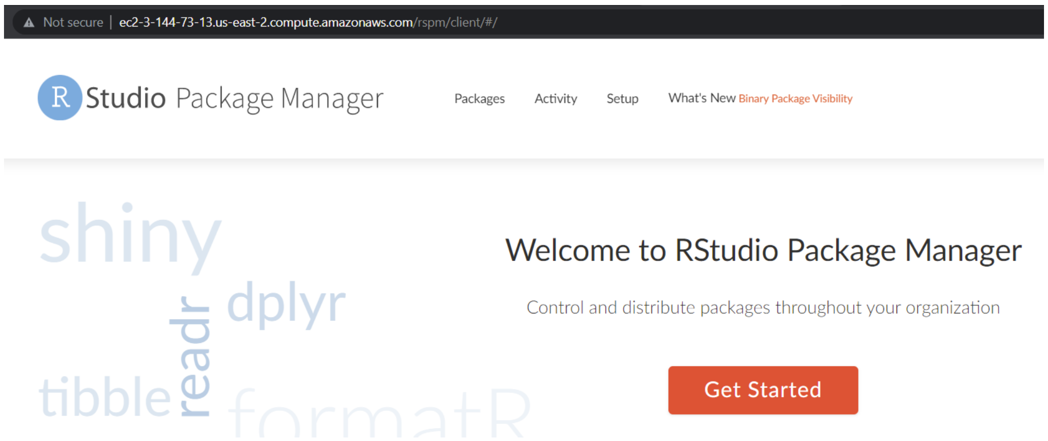
You can add othe approved packages for installation, using a command like this:

rspm add --source=validated --include-suggests --csv-out example.csv --packages=tidyverse,tidymodels,gt,lintr,xml2,pak,usethis,processx,httr,httr2,styler,desc,pillar,rlang,downlit,devtools,rprojroot,roxygen2,here,testthat,cachem,tidyselect,cpp11,cli,vctrs,pkgdown,R6,waldo,ps,remotes,pkgdepends,gh,scales,ymlthis,filelock,callr,withr,gert,fs,crayon,gert,backports,pkgcache,whoami,zip,sessioninfo,pkgbuild,brio,progress,pkgload,bench,gitcreds,rcmdcheck,commonmark,later,lifecycle,covr,gargle,clock,memoise,conflicted,fastmap,meltr,jose,rematch2,lobstr

Step 3: Create Repo

Now we will create the repo with:

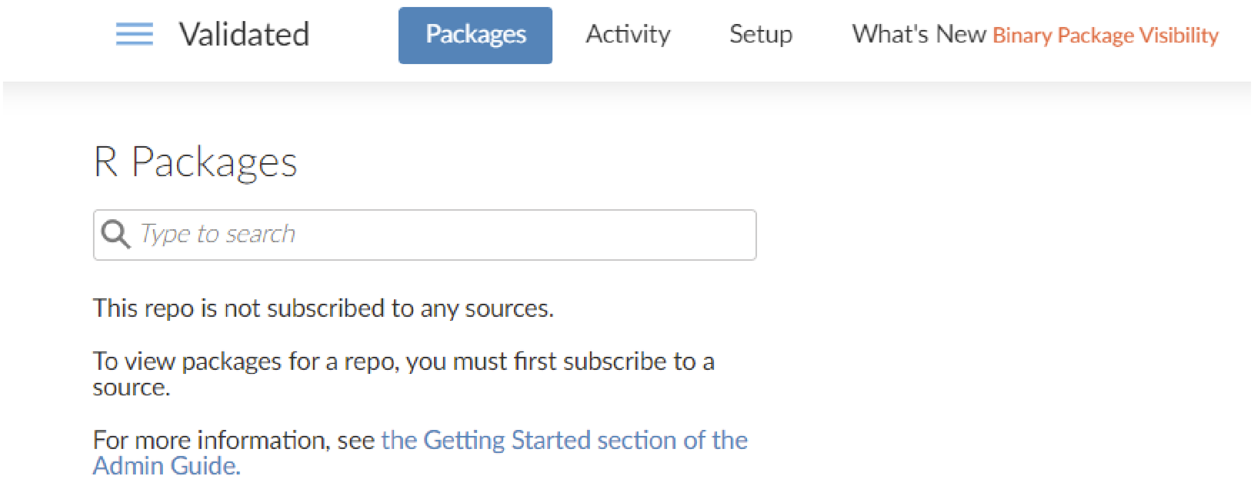
rspm create repo --name=Validated


Not secure | ec2-3-144-73-13.us-east-2.compute.amazonaws.com/rspm/client/#/
RStudio Package Manager
Packages
Activity
Setup
What's New
Binary Package Visibility
shiny
dplyr
readr
tibble
formatR
Get Started

Welcome to RStudio Package Manager

Control and distribute packages throughout your organization

Now we will see that RSPM UI has changed but still has no sources:



R Packages

Type to search

This repo is not subscribed to any sources.

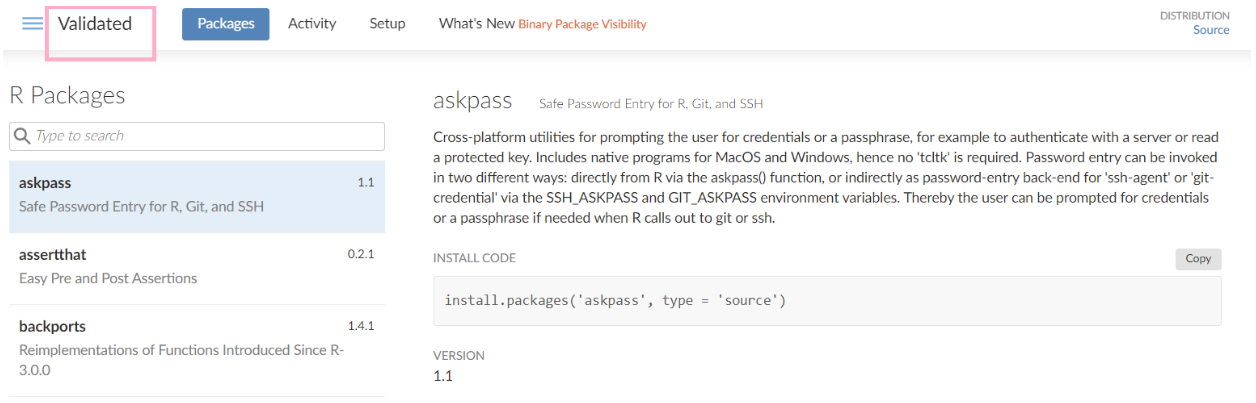
To view packages for a repo, you must first subscribe to a source.

For more information, see the Getting Started section of the Admin Guide.

Now we will run command:

rspm subscribe --repo=Validated --source=validated

At this point, we have added the packages to RSPM and users can now install packages:



askpass Safe Password Entry for R, Git, and SSH	1.1
assertthat Easy Pre and Post Assertions	0.2.1
backports Reimplementations of Functions Introduced Since R-3.0.0	1.4.1

askpass Safe Password Entry for R, Git, and SSH

Cross-platform utilities for prompting the user for credentials or a passphrase, for example to authenticate with a server or read a protected key. Includes native programs for MacOS and Windows, hence no 'tcltk' is required. Password entry can be invoked in two different ways: directly from R via the askpass() function, or indirectly as password-entry back-end for 'ssh-agent' or 'git-credential' via the SSH_ASKPASS and GIT_ASKPASS environment variables. Thereby the user can be prompted for credentials or a passphrase if needed when R calls out to git or ssh.

INSTALL CODE

```
install.packages('askpass', type = 'source')
```

VERSION

1.1

You can read more here:

https://environments.rstudio.com/validated

Step 4 - Set Validated repo as source of install.packages

Admins then often wonder how to lock users into those package sets. Admins can find Setup info in RSPM under the Setup tab at the top. The best way to accomplish that users get the right packages are by:

- Set the right default repository in RStudio Workbench
- Disallow access to other repositories as needed
- Fully disallowing access to public repositories can be accomplished via networking rules. There is no way to disallow RStudio Server users from changing their repositories, but networking rules can prevent those repositories from being accessed.

It is also possible to disallow changing the installation repository in the RStudio GUI by setting the allow-r-cran-repos-edit = 0 in /etc/rstudio/rsession.conf. You can read more here:

https://solutions.rstudio.com/data-science-admin/packages/

There are 2 common approaches for how to actually install the packages into the RStudio sessions for users:

A. Allow users the freedom to install packages from the Validated 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. A user can use the command below to install needed packages from the Validated repo with this command:

```
/opt/R/4.0.2/bin/R -e 'install.packages(c("tidyverse","tidymodels","gt","shiny","rmarkdown"), repos =
"http://ec2-3-144-73-13.us-east-2.compute.amazonaws.com/rspm/Validated/__linux__/bionic/latest")'
```

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.

In the most common scenario, users will install all packages from a single RStudio Package Manager repository that will contain the RStudio and other CRAN packages as well as your internal packages. An admin can discourage users from changing this repository setting. In this scenario, configure RStudio Workbench:

In /etc/rstudio/rsession.conf, set:

```
r-cran-repos=https://<package-manager-server-address>/<repo-name>/latest
```

The exact URL to use is available in the RStudio Package Manager Setup page for the repository.

In RStudio Workbench, to disable the repository option menu and discourage users from changing the repository setting, also include in /etc/rstudio/rsession.conf:

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

You can read more here:

https://docs.rstudio.com/rspm/admin/rstudio-server/#a-single-repository

B. Admins install packages via Validated repo (or a a Frozen Repository within Validated)

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 System 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 i.e. block outbound access to GitHub, CRAN, etc. with a firewall. IT is familiar with these types of restrictions.

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)

Below is the command to install packages into the System location from our Validated repo for all users:

sudo /opt/R/4.0.2/bin/R -e 'install.packages(c("tidyverse","tidymodels","gt","shiny","rmarkdown"), repos = "http://ec2-3-144-73-13.us-east-2.compute.amazonaws.com/rspm/Validated/__linux__/bionic/latest")'

(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
```

User Steps: Accessing the Validated Set

There are three options for accessing the set of validated packages:

1. If you are using a Docker container, add a line to install the available packages from the internal repository:

```
FROM ubuntu
...
RUN R -e 'options(repos = c(CRAN =
"https://r-pkgs.example.com/validated"));
install.packages(available.packages()[,"Package"])'
```

2. If you are creating a shared environment for multiple users, then set the repo option in the Rprofile.site to point at the internal repository, and optionally install the available packages. For more details, refer to the shared baseline strategy, replacing the generic frozen repository with your validated internal repository.

```
# after installing R
# set the repo option in Rprofile.site

export R_HOME=/opt/R/4.0.2
echo 'options("repos" = c("RSPM" =
"https://colorado.rstudio.com/rspm/classroom/__linux__/bionic/5478112"))'
>> $R_HOME/lib/R/etc/Rprofile.site

# optionally, install the packages
sudo R_HOME/bin/R -e 'install.packages(available.packages()[,"Package"])'


cat /opt/R/4.0.2/lib/R/etc/Rprofile.site
```

3. If you are an individual working on a specific project, you can use the renv package to create an isolated project environment and library associated with the validated package set:

```
# from within your project directory
renv::init()

# update the environment to use the validated set from the internal
repository
renv::modify() //TODO: Change this!

# install packages like normal
install.packages(...)
```

You can read more here on validated environments:

https://environments.rstudio.com/validated

Step 5 - Adding New R Versions

It is a best practice not to upgrade R but rather to install multiple versions on the same server. As you add new R versions, you can freeze your Validated CRAN repo to align previous R versions to past versions of packages in the repository.

When Serving the Curated CRAN Subsets as the source of the Validated Repo, in order to ensure consistency, the entire source must be updated at once, not individual packages:

Terminal
#Update packages in a curated-cran source:
rspm update --source=subset --snapshot=2020-10-07

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.

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"))
})
```

One challenge that organizations face when updating a repository is the old package versions are archived during the upgrade. Though tools like renv can help install from the archive, it makes reinstallation of the old combinations unavailable to default tooling such as install.packages. Instead, we propose either the time-travel functionality of RSPM be used within a repository or to create multiple, unique “validated” repos. Namely, instead of calling it just "validated", a repository could be named "validated-2022-05-01" or "validated-R40". When it comes time to create a new set of packages that would correspond to some event (eg. when upgrading R version) one would make a new repo following the exact same procedure as before. At that point users have both validated repositories accessible. This also gives some great other benefits besides just making it always accessible/installable even later in time. For example, you can much more easily build tooling to inspect those repos for IT documentation such as generating tables saying what the old/new versions are. This is about the same level of effort, and actually shortens the overall procedure conditions since no "upgrade" style procedures are needed. The other reason a multi-repo strategy over time can be a good foundation is when organizations are positioned to support multiple teams that want to construct different subsets of cran, the repository management procedures continue to stay consistent and users are already accustomed to interacting with different named repositories.

PICK SNAPSHOT FOR NEWLY INSTALLED RSPM FOR LATEST DATE FOR VALIDATED REPO

To configure R, set the repos option in R to include the repository URL at the top of the page.

Example repository setup code:

options(repos = c(REPO_NAME = "https://colorado.rstudio.com/rspm/validated/latest"))

We recommend adding this to your R startup file (Rprofile.site or .Rprofile) to maintain the configuration across R sessions. More information about managing R startup files is available here:

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

STEP 6 - FURTHER INFORMATION

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

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')

renv

renv is a R package that helps to create reproducible environments for your R projects including shiny apps. This ensures reproducibility by having project-local dependencies stored in a project library for your work.

https://rstudio.github.io/renv/

1. **renv::init()** - initialize a new project-local environment with a private R library
2. Work in the project as normal, installing and removing new R packages

3. **renv::snapshot()** - save the state of the project library to the lockfile (called renv.lock)
4. Work in the project as normal, installing and removing new R packages
5. **renv::restore()** - revert to the previous state as encoded in the lockfile

If for some reason, a R programmer needs to do development in a new environment, or if a colleague needs to replicate the environment that was used previously, this can be accomplished with: **renv::restore()**

Installing R
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

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 advice from above applies: Install packages from a frozen repository that aligns to the version of R used. You can read more here:

https://environments.rstudio.com/docker

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 practice 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.

Production for Clinical Artifacts
It is recommended that statistical programmers write the code; then run the results in a way that can demonstrate the reproducibility of the script. 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 controlled 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 with an artifact-specific manifest. RStudio Connect can also be used as a run-log to capture the run results in a traceable, shareable way.

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.

Contact & Summary

The information above highlights the exciting space of software development, and how well established tools in that ecosystem can help modernize clinical processes for reporting via Shiny applications.

Phil Bowsher

phil@rstudio.com

https://github.com/philbowsher