Using and Customizing R Packages

Creating a Custom R Library

R packages can be finicky. See **Switching Between Custom Libraries** and **Common Problems** below to help with frequent user issues.

Creating Your First Library

1. Make a local directory to store your packages:

```
$ mkdir -p ~/R/library
```

2. Tell R where the directory is by creating an environment file:

```
$ echo 'R_LIBS=~/R/library/' >> ~/.Renviron
```

3. That's it! Now you can install packages normally. For example, to install and load the package "ggplot2":

```
$ module load R
$ R
> install.packages("ggplot2")
> library(ggplot2)
```

Switching Between Custom Libraries

If you're using different versions of R, we recommend you use different libraries. See **Common Problems** below for more information. When creating a library, consider including pertinent information in the name such as R version. For example:

If you start by using R version 4.0, following the instructions provided above:

```
$ mkdir -p ~/R/library_R_v4.0
$ echo 'R_LIBS=~/R/library_R_v4.0/' >> ~/.Renviron
```

If you later decide to switch to R version 4.1, instead of using your existing library, create a new one:

```
$ mkdir -p ~/R/library_R_v4.1
```

To use your new library, edit your .Renviron file:

```
$ nano ~/.Renviron                 # opens the R environment file for
editing
> R_LIBS=~/R/library_R_v4.1        # delete ~/R/library_R_v4.0 and enter
the new directory
> CONTROL + X                      # exits (remember to save at the prompt)
```

Now you can go about your business and install as you normally would.

Common Problems

General Debugging

Working on a cluster without root privileges can lead to complications. Some are listed below with suggested solutions:

1. **General Installation Questions**

   Solution: Check out http://www.r-bloggers.com/installing-r-packages/ for more information.

2. **A Corrupted Environment:** One of the most common reasons R packages won't install is an altered environment. Most frequently this is caused by the presence anaconda (or miniconda) installed locally.

   Solution: If Anaconda is present, follow the instructions in the **Resolving Anaconda Issues** sec tion below. Otherwise:

   Look for any of the file types listed below on your account. If you find them, remove them (make a backup somewhere if you need them) and try the installation again.
   a. **Saved R sessions.** If this is the case, after starting a session, you will get the message "[Previously saved workspace restored]". Old sessions are saved as a hidden file .RData in your home directory.
   b. **Gnu compilers**
   c. **Windows files**

3. **Library Issues**

   Solution: Double-check that you have an .Renviron file. This is a hidden file located in your home directory and should set the path to your custom R library. If you do not have a custom library name set up, R will create one for you saved as something like:

   ```
   ~/R/x86_64-pc-linux-gnu-library
   ```

   This directory can lead to unwanted behavior. For example, if you're trying to use a new custom library (such as when switching R version), R will still search x86_64-pc-linux-gnu-library for package dependencies and may cause installs to fail. To fix this, rename these types of folders something unique and descriptive.

   To set up/switch custom libraries, follow the instructions in the **Creating a Custom R Library** se ction above.

4. **Mixing R Versions:** Because HPC is a cluster where multiple versions of R are available, users should take care to avoid mixing and matching. Because packages often depend on one another, libraries using different versions of R can turn into a tangled mess. Common errors that can crop up include: "Error: package or namespace load failed."

   Solution: If you're switching R versions, we recommend creating a new library.

5. **OOD RStudio Issues:** OOD RStudio is a great tool! Sometimes though, because it's a different environment than working directly from the terminal, you may run into problems. Specifically, these typically arise for installs or when using packages that rely on software modules.

   Solution:

   Package Installations: If you're trying to install a package in an OOD RStudio session and you've tried all the troubleshooting advice above without luck, try starting R in the terminal and give the installation another try. You can start an R session in the terminal using:

```
$ module load R/<version>
$ R
> install.packages("package_name")
```

Remember that when you log into HPC, you're on a login node so you'll want to start up an interactive session to access R and for the installation.

Accessing Modules: RStudio does not have access to **module load** commands. This means that if you have a package that relies on a system module, the easiest option is to work through an interactive terminal session.

The alternative is to to modify your RStudio environment. For example, the library hdf5r relies on the hdf5 software module. If you try to load hdf5r, you will get an error complaining about a shared object file. To get around this, you will need to manually add that shared object to your environment using dyn.load(). For example:

```
> library("hdf5r") # without using dyn.load()
Error: package or namespace load failed for ‘hdf5r’ in dyn.load
(file, DLLpath = DLLpath, ...):
 unable to load shared object '/home/u21/sarawillis/R/lib_4.0/hdf5r
/libs/hdf5r.so':
  libhdf5_hl.so.100: cannot open shared object file: No such file or
directory
> dyn.load("/opt/ohpc/pub/libs/gnu8/hdf5/1.10.5/lib/libhdf5_hl.so.
100")
> library("hdf5r") # success!
>
```

This requires that you know the location of the relevant file(s). These can usually be tracked down by looking at your system path variables (e.g. LD_LIBRARY_PATH) after loading the relevant module in a terminal. It should be noted that modifying your system paths from RStudio will not help since RStudio has its own configuration file that overrides these.

6. **Software Installed in Non-Standard Locations:** When packages are dependent on 3rd party software, particularly when the software is installed locally, R can have trouble finding it. This can usually be fixed by changing your environment paths but can sometimes be challenging.

   Solution:

   a. Packages that require 3rd party software should be installed in a terminal session and not through an OOD RStudio session.
   b. Check whether the software you need is installed as a module using the **module avail** command.
   c. If you know which paths need to be changed, point them to the correct location.
   d. Search online help forums such as R-Bloggers, Stack Exchange, Stack Overflow, etc. for your specific error. It's likely others have experienced the same problem you're encountering and know where the trouble spots are.
   e. If you're in too deep, reach out to the consultants with a support ticket.
   f. Sometimes it can't be helped and you need the software installed globally. Submit a software installation request to get things up and running. Note: there is no expected timeframe for software requests.

Resolving Anaconda Issues

When Anaconda is initialized, your .bashrc file is edited so that it becomes the first thing in your PATH variable. This can cause all sorts of mayhem. To get around this, you can either remove anaconda from your PATH and deactivate your environment, or comment out/delete the initialization in your ~/.bashrc if you want the change to be permanent.

Turn off Auto-activation

Anaconda's initialization will tell it to automatically activate itself when you log in (when anaconda is active, you will see a "(conda)" preceding your command prompt). To disable this behavior, run the following from the command line in an interactive terminal session:

```
conda config --set auto_activate_base false
```

This will suppress anaconda's activation until you explicitly call `conda activate` and is a handy way to have more control over your environment. Once you run this, you will either need to log out and log back in again to make the changes live, or you can follow the instructions in the section below.

Temporary Removal

You can either use the command conda deactivate and then manually edit your PATH variable to remove all instances of anaconda/miniconda or copy the following and run it in your terminal:

```
conda deactivate > /dev/null 2>&1
IFS=':' read -ra PATHAR <<< "$PATH"
for i in "${PATHAR[@]}"
    do if [[ $i == *"conda"* ]]
        then echo "removing $i from PATH"
    else NEWPATH=$i:$NEWPATH
    fi
done
export PATH=$NEWPATH
module unload gnu8 && module load gnu8
unset NEWPATH
echo "Successfully removed conda"
```

Permanent Removal

Your .bashrc file configures your environment each time you start a new session. You may consider making a backup before editing in case of unwanted changes.

Note: this change will remove anaconda from all future terminal sessions but will not make the changes live right away. To make the changes live, either follow the instructions above for removing anaconda from your PATH, or log out and back in again.

```
$ nano ~/.bashrc # opens your bashrc file to edit
```

Then comment out or delete the following lines and the text in between:

```
# >>> conda initialize >>>
...
# <<< conda initialize <<<
```

To exit and save, use control+x and follow the prompts.

Using RStudio

Open OnDemand Application

We provide access to the popular development environment RStudio through our Open OnDemand web interface. This is a very handy tool, though it should be noted that it is a less flexible environment than using R from the command line. This is because RStudio sets its own environment which prevents easy access to third party software installed as system modules. These issues can sometimes worked around by following the guide in the debugging section above.

Using Singularity

In some circumstances, you may want to run RStudio using your own Singularity image. For example, this allows access to different versions of R not provided when using our OOD application. We have some instructions on how to do this below.

First, log into HPC using an Open OnDemand Desktop session and open a terminal. A Desktop session is the easiest solution to access RStudio since it eliminates the need for port forwarding.

In the terminal, make an RStudio directory where all of the necessary files will be stored. In this example, we'll be working in our home directory and will pull an RStudio image from Dockerhub to use as a test. If you're interested, you can find different RStudio images under rocker in Dockerhub.

Make an RStudio directory and pull your image

```
mkdir $HOME/RStudio
cd $HOME/RStudio
singularity pull ./geospatial.sif docker://rocker/geospatial.sif
```

Next, create the necessary directories RStudio will use to generate temporary files. You will also generate a secure cookie key.

Make RStudio temp directories and generate secure cookie key

```
TMPDIR=$HOME/RStudio/rstudio-tmp
mkdir -p $TMPDIR/tmp/rstudio-server
uuidgen > $TMPDIR/tmp/rstudio-server/secure-cookie-key
chmod 600 $TMPDIR/tmp/rstudio-server/secure-cookie-key
mkdir -p $TMPDIR/var/{lib,run}
```

Next, create a file in your RStudio directory called rserver.sh and make it an executable:

Create rserver.sh and modify permissions

```
touch rserver.sh
chmod u+x rserver.sh
```

Open the file in your favorite editor and enter the content below. Modify the variables under USER OPTIONS to match your account if necessary. You can change PASSWORD to any password you'd like to use. Once you've entered the contents, save and exit:

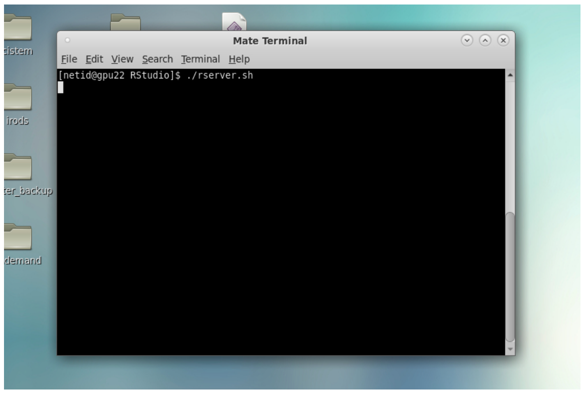
rserver.sh

```
#!/bin/bash

# --- USER OPTIONS --- #
WD=$HOME/RStudio
SIFNAME=geospatial.sif
PASSWORD="PASSWORD"

# --- SERVER STARTUP EXECUTED BELOW --- #
NETID=$(whoami)
TMPDIR=$WD/rstudio-tmp
SIF=$WD/$SIFNAME
PASSWORD=$PASSWORD singularity exec -B $TMPDIR/var/lib:/var/lib/rstudio-
server -B $TMPDIR/var/run:/var/run/rstudio-server  -B $TMPDIR/tmp:/tmp
$SIF rserver --auth-none=0 --auth-pam-helper-path=pam-helper --server-
user=$NETID --www-address=127.0.0.1
```

Now, in your desktop session's terminal, execute the rserver.sh script using:

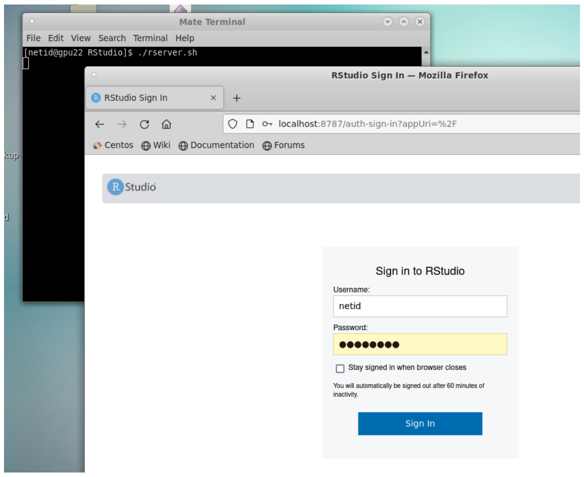
Execute rserver.sh

```
./rserver.sh
```



```
[netid@gpu22 RStudio]$ ./rserver.sh
```

Next, open a Firefox window and enter "localhost:8787" for the URL. In your browser, you will be prompted to log into your RStudio server. Enter your NetID under Username. Under Password, enter the password you defined in the script server.sh.



```
[netid@gpu22 RStudio]$ ./rserver.sh
```

localhost:8787/auth-sign-in?appUri=%2F

Sign in to RStudio

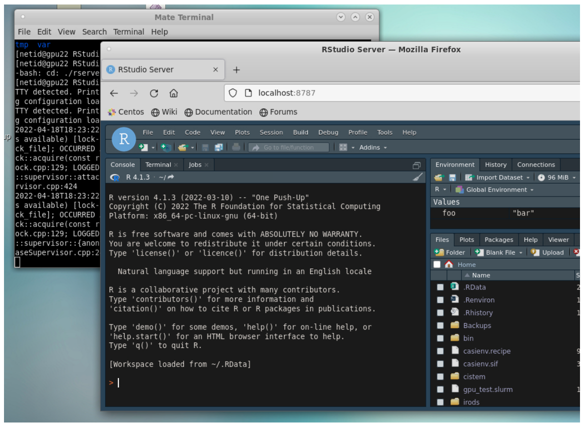
Username: netid

Password: ●●●●●●●●

Stay signed in when browser closes

You will automatically be signed out after 60 minutes of inactivity.

This will open your RStudio session:



```
R version 4.1.3 (2022-03-10) -- "One Push-Up"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Example R Scripts

We have examples for running R workflows in batch on our Github Examples page. Have a question or suggestion? Let us know!