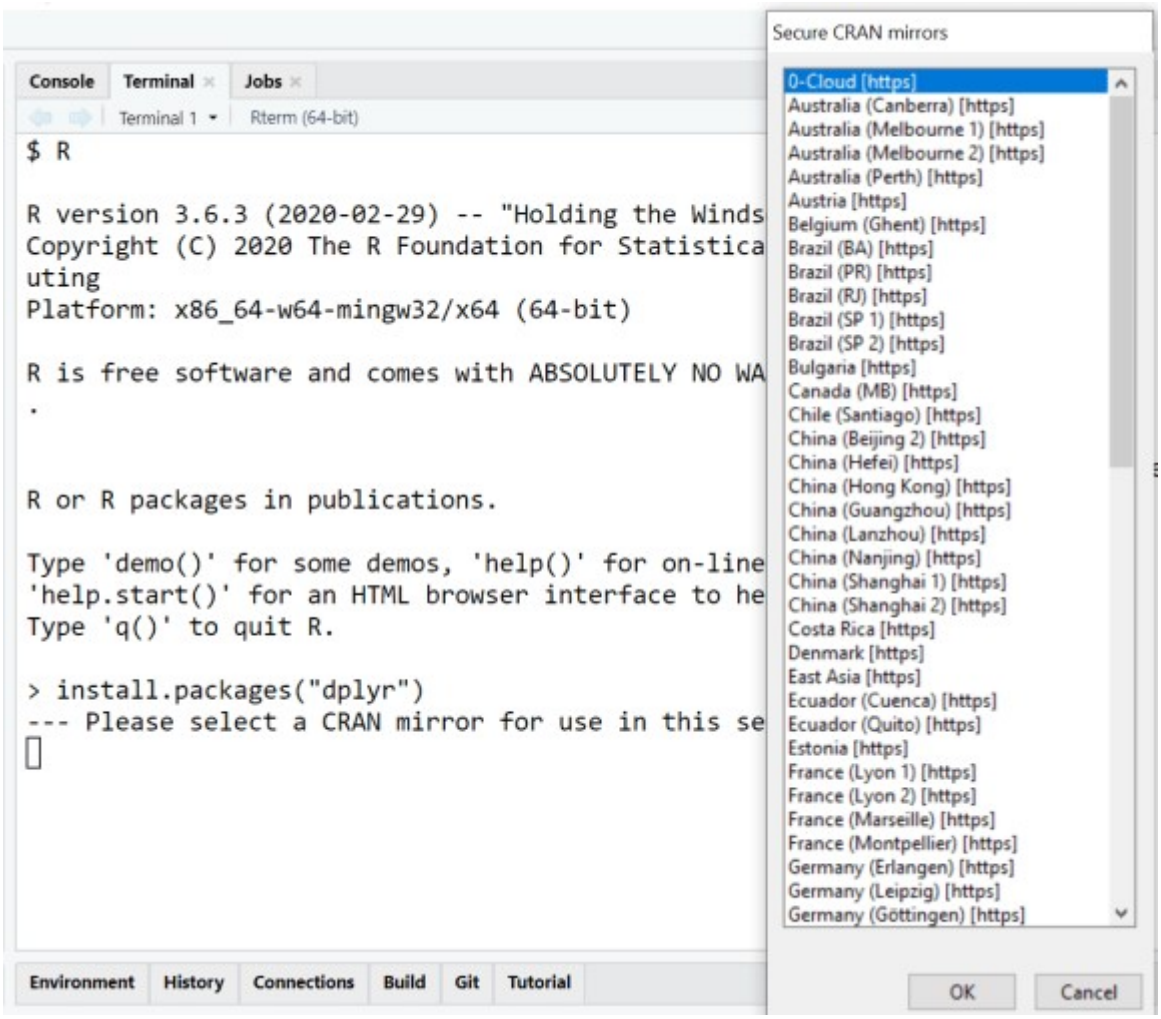
Installing from CRAN

Installing packages from the Comprehensive R Archive Network (CRAN) couldn't be easier! Simply type `install.packages()` with the name of your desired package in quotes as first argument. In case you want to install the {dplyr} package you would need to type the following.

```
install.packages("dplyr")
```

This will install `{dplyr}` along with all its dependencies, i.e. other packages that `{dplyr}` uses internally.

When you execute `install.packages()` outside of RStudio, e.g. in R's built-in GUI or in the terminal, you will be asked to select a CRAN mirror.



CRAN is hosted on over 50 different servers spread across the world. If you are lazy you can just select the first entry (0-Cloud) which is the mirror hosted by RStudio. Otherwise choose the mirror nearest to your current location to maximize the download speed.

Installing from Bioconductor

In order to install R packages from Bioconductor—a repository specifically designed for bioinformatics packages—you first need to install the {BiocManager} package from CRAN.

```
install.packages("BiocManager")
```

Once {BiocManager} has been successfully installed, you can install any package from Bioconductor using the `BiocManager::install()` function, e.g.

```
BiocManager::install("ArrayTools")
```

Installing from GitHub

While CRAN is still by far the most popular repository for R packages, you will find quite a lot of packages that are only available from GitHub. Furthermore, if you'd liked to try out the latest development versions of popular packages such as {ggplot2} and {tidyr} you will have to install them from GitHub.

Before you can install any package from GitHub, you need to install the {remotes} package from CRAN.

```
install.packages("remotes")
```

Now you can install any package from GitHub by providing `"username/repository"` as argument to `remotes::install_github()`. For example, to install the latest development version of {ggplot2} from GitHub run this command.

```
remotes::install_github("tidyverse/ggplot2")
```

Please note that installing a package from GitHub means that you have to install it from source. This requires a proper development environment including (at the minimum) a C and FORTRAN compiler. On Linux and macOS that's a given but on Windows it's not. More on that below where I discuss the difference between installing binaries vs. installing from source.

Installing from Other Sources

The {remotes} package can install R packages from many other sources including Gitlab, SVN and Bitbucket as well as your local PC. Here's a list of all `install_*()` functions the package contains.

```
grep(
  pattern = "^install_",
  x = getNamespaceExports("remotes"),
  value = TRUE
)
##  [1] "install_gitlab"    "install_url"       "install_github"
##  [4] "install_dev"       "install_git"       "install_version"
##  [7] "install_bioc"      "install_deps"      "install_bitbucket"
## [10] "install_cran"      "install_local"     "install_svn"
```

Installing Specifc Package Versions

By default, `install.packages()` and co. install the latest version of a package. In general that's a good thing. The most recent version likely incorporates bug fixes for issues from earlier version plus new features. However, there are situations in which you don't want to install the latest version of a package because it could break your existing code.

The first way to install a specific version of a package is using `remotes::install_version()`. In addition to the name of the package you'd like to install you pass it a version string as second argument. The function will go though the CRAN archives

and install that specific version—if it exists.

```
remotes::install_version("dplyr", "0.8.5")
```

Another approach to installing a specific package version—or rather the version from a specific point in time—is to make use to use of MRAN's (Microsoft R Archive Network) daily CRAN snapshots. Every day new packages and new versions of existing packages are uploaded to CRAN. Thus, running `install.packages("dplyr")` next week might install a different version than running the command today. By using a MRAN snapshot you make sure that whenever you execute the installation command you always get the packages from the date of the snapshot.

MRAN snapshots are URLs with the canonical form https://cran.microsoft.com/snapshot/yyyy-mm-dd/, e.g. https://cran.microsoft.com/snapshot/2020-01-20/. To use such a snapshot you have two options. The first one is to explicitly provide the URL as argument to the `repos` parameter of `install.packages()`.

```
install.packages("dplyr", repos = "https://cran.microsoft.com/snapshot/2017-03-15/")
```

Alternatively, you can set the `repos` option globally.

```
options(repos = "https://cran.microsoft.com/snapshot/2017-03-15/")
```

Any subsequent call to `install.packages()` will make use of this globally set option unless you overwrite it by passing another value to the `repos` parameter.

Binary vs. Source Installation

The `install.packages()` function has a `type` parameter that controls whether a package is installed from pre-complied binaries or from source. The former is the default on Windows and some macOS versions while on Linux it's the latter. So what's the difference?

When `type = "binary"`, `install.packages()` will attempt to download a pre-compiled version of the requested package for the operating system (OS) you are working on. Pre-complied means that the package has been complied on another machine with the same OS as yours and subsequently uploaded to CRAN. This is very convenient because not having to compile packages yourself has a huge advantage: it's (much) faster. If you've ever tried to install the {tidyverse} from source you will know what I mean. It's no overstatement when I say that you should take a long coffee break after entering `install.packages("tidyverse", type = "source")`.

Are there advantages to installing packages from source? In theory yes. You could set certain compiler flags to optimize the code and link to more performant libraries, e.g. multi-threaded BLAS/LAPACK linear algebra libraries. However, that is provided you know what you are doing. I assume that 99.9% of my readers don't and I include myself to that list. So in short, install from pre-compiled binaries unless they are not available for your OS.

Where Are Packages Installed To?

R has what I like to call the package search path. This is a list of directories in which it looks for R packages when you use `library()`. You can view these directories using the `.libPaths()` function.

```
.libPaths()
## [1] "C:/Users/neitmant/AppData/Roaming/R-3.6.3/library"
```

As you can see on my machine the package search path is just a single directory. That is typically the case on Windows if you install R only for a specific user. If you install it for all users on the machine you usually have two entries: a global one containing all the default packages that ship with R and a user specific one where all additional packages are installed to.

You can add any directory you like to the package search path yourself. Importantly, R will install a new package to the very first entry of the package search path, i.e. `.libPaths()[1L]`.

```
if (!dir.exists("./library")) {
  dir.create("./library")
}
old_libraries <- .libPaths()
.libPaths(c("./library", old_libraries))
.libPaths()
## [1] "C:/Users/neitmant/Documents/blog2/library"
## [2] "C:/Users/neitmant/AppData/Roaming/R-3.6.3/library"
```

From now on any time I call `install.packages()` the package will be installed in the C:/Users/neitmant/Documents/blog2/library directory rather then the standard C:/Users/neitmant /AppData/Roaming/R-3.6.3/library directory.

Troubleshooting Common Installation Errors

Package ‘abc’ Is Not Available (for R Version x.y.z)

A rather common issue that pops up when trying to install a package from CRAN is `package ‘abc’ is not available (for R version x.y.z)`. On first sight this looks as though the package you requested is not available for the R version you are using (but potentially other ones). In reality the most common reason for this warning is that you misspelled a package name, e.g. you tried to install `{gplot2}` rather than `{ggplot2}`.

```
install.packages("gplot2")
## Installing package into 'C:/Users/neitmant/Documents/blog2/library'
## (as 'lib' is unspecified)

## Warning: package 'gplot2' is not available (for R version 3.6.3)
```

Another reason for this warning could be that the package is not on CRAN (yet). This is often the case with experimental new packages that you may have read about on #RStats twitter. The package will most likely be available on GitHub and can be installed using `remotes::install_github()`.

Yet another potential source for this issue is that you attempted to install a Bioconductor package with `install.packages()`. For that you have to use `BiocManager::install()` rather then `install.packages()`.

Updating Loaded Packages

This is a very annoying error. The error message itself is rather self-explanatory. You have already called `library()` with one of the packages you are attempting to install. Often, though, it’s not obvious which package is causing the problem. You may have called `library(dplyr)`

at the start of your session and are now trying to install another package that depends on `{dplyr}`. This package likely requires a specific version of `{dplyr}`. You might have version `0.8.1` installed but `1.0.0` is required. In such cases R will automatically try to install the most recent version of `{dplyr}`. But if that package is already loaded you'll get the `Updating loaded packages` error.

When you encounter this error the first thing you should do is restart your R session and try to install again without loading any packages. If you still get the same error you likely have a `library(pkg)` call in your `.Rprofile` file or you load saved objects into your workspace when starting R. I'd highly recommend to do neither of those. As a next step I'd recommend to close any currently active R session. Then go to your terminal—or CMD on Windows—and type `R --vanilla`. This will start an R session that will ignore your `.Rprofile` and doesn't restore any saved objects. You are starting from a clean state so to say. Installing the package should now work.

There Are Binary Versions Available but the Source Versions Are Later

This is a common error when using R on Windows. It is typically followed by the seemingly innocent question: `Do you want to install from sources the package which needs compilation?`. Unless you have a proper development environment on your Windows machine, i.e. you installed the Rtools software, you should say `no`. Attempting to install from source when you don't have a proper development environment is guaranteed to lead to failing package installations.

When you do answer `no` R will install the previous version of the package for which pre-complied binaries are available rather than the latest version which is only available from source. This is typically the case in the first few days after a new package version has been published on CRAN. Should that still not work try to explicitly install the previous version of the package using `remotes::install_version()` as described above.

Failed to Create Lock Directory

This error occurs when your last package installation attempt has interrupted abnormally, e.g. when you hit Ctrl-C to terminate it.

```
install.packages("Rcpp")
Installing package(s) into 'C:/Users/neitmant/AppData/Roaming/R-
3.6.3/library' (as 'lib' is unspecified)
trying URL 'http://cran.us.r-project.org/src/contrib/Rcpp_0.10.2.tar.gz'
Content type 'application/x-gzip' length 2380089 bytes (2.3 Mb)
...
Warning in dir.create(lockdir, recursive = TRUE) :
  cannot create dir '/home', reason 'Permission denied'
ERROR: failed to create lock directory 'C:/Users/neitmant/AppData/
Roaming/R-3.6.3/library/00LOCK-Rcpp'
...
```

According to the `install.packages()` documentation "[the lock directory] has two purposes: it prevents any other process installing into that library concurrently, and is used to store any previous version of the package to restore on error".

To mitigate this error you can either manually delete the left-over `00LOCK` directory or you tell R to not create one during installation. To delete the `00LOCK` directory you can either use the file

explorer of your OS or you do it directly from within R using the following command.

```
unlink("C:/Users/neitmant/AppData/Roaming/R-3.6.3/library/00LOCK-Rcpp",
recursive = TRUE)
```

Make sure to pass the exact path you received in the error message as first argument to `unlink()`.

To make `install.packages()` not create a `00LOCK` directory, pass the `"--no-lock"` flag to its `INSTALL_opts` parameter.

```
install.packages("Rcpp", INSTALL_opts = "--no-lock")
```

Wrap Up

Packages are a primary reason for R's popularity. They can be installed from various sources including—but not limited to—CRAN, Bioconductor and GitHub. While installing packages is most of the time straightforward, errors do occur. This article discussed some of the most common installation errors and provided solutions on how to mitigate them. Now it's up to you to install your favorite packages and get programming!