Note – I'm writing this after already upgrading so there will be a few inconsistencies in the output

- This could just as easily be a `tibble` but I chose `as.data.frame`
- I am deliberately removing base packages from the dataframe by `filter`
- I am eliminating columns I really don't care about with `select`

```
require(tidyverse)

allmypackages <- as.data.frame(installed.packages())

allmypackages <- allmypackages %>%
  filter(Priority != "base" | is.na(Priority)) %>%
  select(-c(Enhances:MD5sum, LinkingTo:Suggests)) %>%
  droplevels()

str(allmypackages)
```

A function to do the hard work

As I mentioned above the stack overflow post was a good start but I wanted more information from the function. Rather than TRUE/FALSE to is it github I would like as much information as possible about where I got the package. The `package~source` function will be applied to the `Package` column for each row of our dataframe. For example `as.character(packageDescription("ggplot2")$Repository)` will get back "CRAN", and `as.character(packageDescription("CHAID")$Repository)` will yield "R-Forge". For GitHub packages the result is `character(0)` which has a `length` of zero. So we'll test with an `if else` clause. If we get an answer like "CRAN" we'll just `return` it. If not, we'll see if there is a GitHub repo listed with `as.character(packageDescription(pkg)$GithubRepo)` as well as a GitHub username `as.character(packageDescription(pkg)$GithubUsername)`. If they exist we'll concatenate and return. If not we'll return "Other". Besides being good defensive programming this may catch the package you have built for yourself as is the case for me.

```
package_source <- function(pkg){
  x <- as.character(packageDescription(pkg)$Repository)
  if (length(x) == 0) {
    y <- as.character(packageDescription(pkg)$GithubRepo)
    z <- as.character(packageDescription(pkg)$GithubUsername)
    if (length(y) == 0) {
      return("Other")
    } else {
      return(str_c("GitHub repo = ",
                   z,
                   "/",
                   y))
    }
  } else {
    return(x)
  }
}

# show the first 60 as an example
head(sapply(allmypackages$Package,
            package_source),
```

```
    60)
```

What's in your libraries?

Now that we have the `package_source` function we can add a column to our data frame and do a little looking.

```
allmypackages$whereat <- sapply(allmypackages$Package,
                                package_source)
str(allmypackages)

table(allmypackages$whereat)
allmypackages %>%
  filter(whereat == "Other") %>%
  select(Package, Version)
```

And just to be on the safe side we'll also write a copy out as a csv file so we have it around in case we ever need to refer back.

```
write.csv(allmypackages, "mypackagelistApril2020.csv")
```

Go ahead and install R 4.0.0

At this point we have what we need, so go ahead and download and install R 4.0.0. At the end of the installation process you'll have a pristine copy with a new (mostly empty) library directory (on my system it's /Library/Frameworks/R.framework/Versions/4.0/). When next you restart R and R Studio you'll see a clean new version. Let's make use of our data frame to automate most of the process of getting nice clean copies of the libraries we want.

We'll start by getting the entire `tidyverse` since we need several parts and because installing it will trigger the installation of quite a few dependencies and bootstrap our work.

```
# post upgrade with output surpessed
install.packages("tidyverse")
library(tidyverse)
```

Now we have R 4.0.0 and some additional packages. Let's see what we can do. First let's create two dataframes, one with our old list and one with what we have right now. Then we can use `anti_join` to make a dataframe that lists the differences `thediff`. We can use `filter` and `pull` to generate a vector of just the the packages that are on CRAN we want to install.

```
oldpackages <- read.csv("mypackagelistApril2020.csv")
allmypackages <- as.data.frame(installed.packages())
allmypackages <- allmypackages %>%
  filter(Priority != "base" | is.na(Priority)) %>%
  select(-c(Enhances:MD5sum, LinkingTo:Suggests))
thediff <- anti_join(oldpackages,
                     allmypackages,
                     by = "Package")

thediff <- droplevels(thediff)
thediff %>%
  filter(whereat == "CRAN") %>%
  pull(Package) %>%
  as.character
```

Just do it!

Now that you have a nice automated list of everything that is a CRAN package you can give it a final look and see if there is anything else you'd like to filter out. Once you are sure the list is right one final pipe will set the process in motion.

```
thediff %>%
  filter(whereat == "CRAN") %>%
  pull(Package) %>%
  as.character %>%
  install.packages
```

Depending on the speed of your network connection and the number of packages you have that will run for a few minutes.

That takes care of our CRAN packages. What about GitHub? Here's another chance to review what you have and whether you still want need these packages. You can automate the process and once again feed the right vector to `devtools::install_github()`.

```
# Manual peek
thediff %>%
  filter(str_detect(whereat, "GitHub repo")) %>%
  select(Package, Version, NeedsCompilation, whereat)

# if you want to automate
thediff %>%
  filter(str_detect(whereat, "GitHub repo")) %>%
  pull(whereat) %>%
  as.character %>%
  str_remove("GitHub repo = ") %>%
  devtools::install_github()
```

Same with the one package I get from R-Forge…

```
allmypackages %>%
  filter(str_detect(whereat, "R-Forge")) %>%
  select(Package, Version, NeedsCompilation, whereat)

install.packages("CHAID", repos="http://R-Forge.R-project.org")
```

At the end of this process you should have a nice clean R install that has all the packages you choose to maintain as well as a detailed listing of what those are.

Done

Hope you enjoyed the post. Comments always welcomed. Especially please let me know if you actually use the tools and find them useful.