Project-oriented workflow

Photo by secumem

2017/12/12

Jenny Bryan

I was honored to speak this week at the IASC-ARS/NZSA Conference, hosted by the Stats Department at The University of Auckland. One of the conference themes is to celebrate the accomplishments of Ross Ihaka, who got R started back in 1992, along with Robert Gentleman. My talk included advice on setting up your R life to maximize effectiveness and reduce frustration.

Two specific slides generated much discussion and consternation in #rstats Twitter:

> *If the first line of your R script is*
>
> `setwd("C:\Users\jenny\path\that\only\I\have")`
>
> *I will come into your office and SET YOUR COMPUTER ON FIRE* 🔥.

> *If the first line of your R script is*

> *I will come into your office and SET YOUR COMPUTER ON FIRE 🔥.*

I stand by these strong opinions, but on their own, threats to commit arson aren't terribly helpful! Here I explain *why* these habits can be harmful and may be indicative of an awkward workflow. Feel free to discuss more on community.rstudio.com.

Caveat: only you can decide how much you care about this. The importance of these practices has a lot to do with whether your code will be run by other people, on other machines, and in the future. If your current practices serve your purposes, then go forth and be happy.

Workflow versus Product

Let's make a distinction between things you do because of personal taste and habits ("workflow") versus the logic and output that is the essence of your project ("product"). These are part of your workflow:

- The editor you use to write your R code.
- The name of your home directory.
- The R code you ran before lunch.

I consider these to be clearly product:

- The raw data.
- The R code someone needs to run on your raw data to get your results, including the explicit `library()` calls to load necessary packages.

Ideally, you don't hardwire anything about your workflow into your product. Workflow-related operations should be executed

Self-contained projects

I suggest organizing each data analysis into a *project*: a folder on your computer that holds all the files relevant to that particular piece of work. I'm **not** assuming this is an RStudio Project, though this is a nice implementation discussed below.

Any resident R script is written assuming that it will be run from a fresh R process with working directory set to the project directory. It creates everything it needs, in its own workspace or folder, and it touches nothing it did not create. For example, it does not install additional packages (another pet peeve of mine).

This convention guarantees that the project can be moved around on your computer or onto other computers and will still "just work". I argue that this is the only practical convention that creates reliable, polite behavior across different computers or users and over time. This convention is neither new, nor unique to R.

It's like agreeing that we will all drive on the left or the right. A hallmark of civilization is following conventions that constrain your behavior a little, in the name of public safety.

Use of a development environment

You will notice that the workflow recommendations given here are easier to implement if you use an IDE (integrated development environment). RStudio is a great example (what I

Visual Studio + RTVS.

Direction of causality: long-time coders don't organize their work into self-contained projects and use relative paths *because* they use an IDE. They use an IDE *because* it makes it easier to follow standard practices, such as these.

What's wrong with `setwd()`?

I run a lot of student code in STAT 545 and, at the start, I see a lot of R scripts that look like this:

```
library(ggplot2)
setwd("/Users/jenny/cuddly_broccoli/verbose_funicular/foofy/data")

df <- read.delim("raw_foofy_data.csv")
p <- ggplot(df, aes(x, y)) + geom_point()
ggsave("../figs/foofy_scatterplot.png")
```

The chance of the `setwd()` command having the desired effect – making the file paths work – for anyone besides its author is 0%. It's also unlikely to work for the author one or two years or computers from now. The project is not self-contained and portable. To recreate and perhaps extend this plot, the lucky recipient will need to hand edit one or more paths to reflect where the project has landed on their machine. When you do this for the 73rd time in 2 days, while marking an assignment, you start to fantasize about lighting the perpetrator's computer on fire.

This use of `setwd()` is also highly suggestive that the useR does all of their work in one R process and manually switches gears when they shift from one project to another. That sort of workflow makes it unpleasant to work on more than one (e.g., objects, loaded packages, session options).

Use projects and the here package

How can you avoid `setwd()` at the top of every script?

- Organize each logical project into a folder on your computer.
- Make sure the top-level folder advertises itself as such. This can be as simple as having an empty file named `.here`. Or, if you use RStudio and/or Git, those both leave characteristic files behind that will get the job done.
- Use the `here()` function from the here package to build the path when you read or write a file. Create paths relative to the top-level directory.
- Whenever you work on this project, launch the R process from the project's top-level directory. If you launch R from the shell, `cd` to the correct folder first.

To continue our example, start R in the `foofy` directory, wherever that may be. Now the code looks like so:

```
library(ggplot2)
library(here)

df <- read.delim(here("data", "raw_foofy_data.csv"))
p <- ggplot(df, aes(x, y)) + geom_point()
ggsave(here("figs", "foofy_scatterplot.png"))
```

This will run, with no edits, for anyone who follows the convention about launching R in the project folder. In fact, it will even work if R's working directory is anywhere inside the project, i.e. it will work from sub-folders. This plays well with

Read up on the here package to learn about more features, such as additional ways to mark the top directory and troubleshooting with `dr_here()`. I have also written a more detailed paean to this package before.

RStudio Projects

This work style is so crucial that RStudio has an official notion of a Project (with a capital "P"). You can designate a new or existing folder as a Project. All this means is that RStudio leaves a file, e.g., `foofy.Rproj`, in the folder, which is used to store settings specific to that project.

Double-click on a `.Rproj` file to open a fresh instance of RStudio, with the working directory and file browser pointed at the project folder. The here package is aware of this and the presence of an `.Rproj` is one of the ways it recognizes the top-level folder for a project.

RStudio fully supports Project-based workflows, making it easy to switch from one to another, have many projects open at once, re-launch recently used Projects, etc.

What's wrong with `rm(list = ls())`?

It's also fairly common to see data analysis scripts that begin with this object-nuking command:

```
rm(list = ls())
```

switches gears when they shift from one project to another. That, in turn, suggests that development frequently happens in a long-running R process that has been used vs. fresh and clean.

The problem is that `rm(list = ls())` does NOT, in fact, create a fresh R process. All it does is delete user-created objects from the global workspace.

Many other changes to the R landscape persist invisibly and can have profound effects on subsequent development. Any packages that have been loaded are still available. Any options that have been set to non-default values remain that way. Working directory is not affected (which is, of course, why we see `setwd()` so often here too!).

Why does this matter? It makes your script vulnerable to hidden dependencies on things you ran in this R process before you executed `rm(list = ls())`.

- You might use functions from a package without including the necessary `library()` call. Your collaborator won't be able to run this script.
- You might code up an analysis assuming that `stringsAsFactors = FALSE` but next week, when you have restarted R, everything will inexplicably be broken.
- You might write paths relative to some random working directory, then be puzzled next month when nothing can be found or results don't appear where you expect.

The solution is to write every script assuming it will be run in a fresh R process. How do you adopt this style? Key steps:

- User-level setup: Do not save `.RData` when you quit R and don't load `.RData` when you fire up R.

- If you run R from the shell, put something like this in your `.bash_profile`:

  ```
  alias R='R --no-save --no-restore-
  data'
  ```

  .
- Don't do things in your `.Rprofile` that affect how R code runs, such as loading a package like dplyr or ggplot or setting an option such as `stringsAsFactors = FALSE`.
- Daily work habit: Restart R very often and re-run your under-development script from the top.
  - If you use RStudio, use the menu item *Session > Restart R* or the associated keyboard shortcut Ctrl+Shift+F10 (Windows and Linux) or Command+Shift+F10 (Mac OS). You can re-run all code up to the current line with Ctrl+Alt+B (Windows and Linux) or Command+Option+B (Mac OS).
  - If you run R from the shell, use Ctrl+D to quit, then `R` to restart.

This requires that you fully embrace the idea that **source is real**:

> *The source code is real. The objects are realizations of the source code. Source for EVERY user modified object is placed in a particular directory or directories, for later editing and retrieval. – from the ESS manual*

This doesn't mean that your scripts need to be perfectly polished and ready to run unattended on a remote server. Scripts can be messy, anticipating interactive execution, but still be *complete*. Clean them up when and if you need to.

`saveRDS(my_precious, here("results", "my_precious.rds"))`. Now you can develop scripts to do downstream work that reload the precious object via `my_precious <- readRDS(here("results", "my_precious.rds"))`. It is a good idea to break data analysis into logical, isolated pieces anyway.

Lastly, `rm(list = ls())` is hostile to anyone that you ask to help you with your R problems. If they take a short break from their own work to help debug your code, their generosity is rewarded by losing all of their previous work. Now granted, if your helper has bought into all the practices recommended here, this is easy to recover from, but it's still irritating. When this happens for the 100th time in a semester, it rekindles the computer arson fantasies triggered by last week's fiascos with `setwd()`.