Project Management*

Project management for reproducible research

This exercise is about practical stuff – managing scientific projects and workflows efficiently to avoid mistakes and keep the research moving forward.

Learning outcomes

1. Keeping file structure
2. Version control
3. Create a data analysis workflow, reporting, and the writing of scientific manuscripts

R Projects

At this point, one might wonder how to keep track of all these files, datasets, and figures. The answer: R projects. An R project consists of a file in a project directory (with .Rproj extension) and the associated data, scripts, and other files. When an R project is opened the following occurs (from: http://rstudio.com):

- A new R session (process) is started
- The .Rprofile file in the project's main directory (if any) is sourced by R
- The .RData file in the project's main directory is loaded (if project options indicate that it should be loaded).
- The .Rhistory file in the project's main directory is loaded into the RStudio History pane (and used for Console Up/Down arrow command history).
- The current working directory is set to the project directory.
- Previously edited source documents are restored into editor tabs
- Other RStudio settings (e.g. active tabs, splitter positions, etc.) are restored to where they were the last time the project was closed.

Version control, Git, and Github

R projects are especially useful for *version control*. Version control is software for managing changes to code, data, and the other files associated with a project. Version control facilitates undoing revisions to an arbitrary point in the history of a file and the clean synchronization of changes contributed by multiple users. *Git* is a popular version control system that works well with R/RStudio through the Github repository service.

Create a Github account:

1. Go to the Github website http://github.com and click on "sign up".
2. Create a profile (username, email, password).
3. Click "start a project".
4. Verify email address.
5. Create a project using the provided form.

Link Github account to R/RStudio:

(There are lots of ways to link your installation of R with Github. These instructions are for use with R/Rstudio installed on a Windows computer such as in the teaching lab. Note that to use Github you also have to have Git installed on your computer.)

*Contributions to lectures and practicals by Andrew W. Park, John M. Drake and Ana I. Bento

1. Open RStudio.
2. Select File -> New Project -> Version Control -> Git.
3. Go to the website for the project you created and click the green "Clone or download" button; copy the URL.
4. Paste the copied URL in the RStudio dialog box and navigate to the directory where you would like your project to be cloned. This will create a local copy of all the files associated with your project.

You should now have a "project" open. In the file explorer you should see the file `README.md` (if you created one), an `.Rproj` file, and another file `.gitignore` (which, conveniently, you can ignore). To include files (e.g. data, scripts) in the project, simply create or copy them into the directory. For instance:

Exercise. Copy the MERS data file `cases.csv` and paste it into your working directory.

Exercise. Create a new script following the prototype we introduced. Your script should load the MERS data and make a plot.

Now you have two files in your local directory that you don't have on the Github repository. To synchronize your local copy and the respository, you need to *Commit* the changes and then *Push* the changes to Github.

Exercise. To *commit* a change, navigate to the "Git" tab in the top right explorer window. You will see a list of files in your work directory. Select the files that need to be pushed to Github and click on "Commit". A dialog box will open. In the top right there is an editing window where you can register a comment describing the nature of the commit.

Exercise. Once you have committed one or more changes (and documented with comments), you need to *push* the changes to the archived version. To do this, click on "Push". You will need to enter your Github credentials in the dialog box that pops up. Now refresh the website for your project. The latest versions of your files should appear.

Github is very useful for collaboration. Multiple users can contribute code to the same project. Projects can be "branched" and "merged" and Github provides tools to identify and resolve conflicts when multiple programmers are working on the same project. Although it's fine to use the Github repository as the definitive version and create a new local copy (a new project) every time, you don't have to. Instead, you can leave your local working directory as it is and simply *pull* your collaborators contributions every time you start a new session.

Statistically literate programming with R Markdown

Literate programming is a programming paradigm due to Linux founder Donald Knuth in which natural langauge explanations of a program's logic are interspersed with the code *snippets* that actually perform the computation. *Statistically literate programming* applies this paradigm to data analysis. Statistically literate programming is the idea that the thought process of the data analyst can be captured in a report that contains explanation and interpretation, the code used to perform an analysis, and the products of that analysis such as tables of data, quantities (e.g. p-values) and graphs.

There are a handful of ways that one can do statistically literate programming with R/RStudio. In this exercise, we will use R/Markdown and `knitr`. Markdown is a lightweight markup language that allows documents to be rendered in html and other formats (e.g. pdf) with a minimum of special formatting. Knitr is a system for dynamic report generation in R. Both should already be installed on your computer.

In general, a project developed with R/Markdown will consist of a markdown document (with extension `.Rmd`) and the compiled report. You should learn a little more about the functions of R/Markdown and knitr, but first we will engage in a little learning-by-doing.

Exercise. Create a basic R Markdown document. Go to File -> New File -> R Markdown. Enter a title and author into the dialog box. Select the desired default output format. Save the resulting, automatically generated file. To compile the document, click on "Knit" in the R Studio editor. Study the code and the resulting report.

You now know enough R programming that you should be able to discern what is commentary and what is code in the `.Rmd` document. First, notice that the `.Rmd` document begins with a header and how it is set off from the rest of the document with three hyphens at the start and end of the header. These are editable properties. Indeed, there are many more options that we could encode in the header, but do not need to bother with now.

Next, observe that *R chunks* begin with three back ticks and curly braces (with the argument "r"). Everything within this section must be properly formatted R code. R code chunks are indicated in the editor window with a different background color. Notice that the third R code chunk includes code to generate a plot. The compiled document shows this code, but also the figure it produces! This is the beauty of statistically literate programming: it keeps the explanation, code, and results all together in a way that can be inspected and used both by the data analyst and by others who come later. Moreover, if the data change (they get updated, there is a correction, the analyst decides to look at a particular subset of the data), all that is required to generate a new report is to recompile the document.

Finally, notice that natural language explanation is interspersed with the code chunks. These chunks don't require any special designation (like the hyphens for the header or the back ticks for the R code), but are basically just "everything else". This natural language commentary can nonetheless be "marked up" with special symbols that cause the formatted document to display sections titles, bold face or italics, clickable URLs, etc. In fact, the Markdown markup language enables quite a lot of reasonably sophisticated typesetting.

In closing, we note that we have only scratched the surface of what can be done with R/Markdown and knitr.

Exercise. Visit the R Markdown website and look around. Especially, read through the Authoring Basics.

Exercise. Borrowing from your earlier exercises, prepare an analysis of the WNV or MERS data as a reproducible document using R/Markdown. Compile to a pdf (if your desktop in class is not compiling in PDF compile in HTML.

Exercise. Add an interactive plotly graph to your reproducible document. Compile to HTML.

This document – and all the others used for this workshop – were written in R/Markdown.