Assignment 2: Empirical Workflow

Juan Andrés Rincón[†]

Causal Inference and Research Design
June 2020

1 Gentzkow & Shapiro (2014)

1.1 Summary: Chapters 2-8

- **Chapter 2: Automation**

  Beginning with two main rules of thumb: (A) *Automate everything that can be automated* and (B) *Write a single script that executes the code from beginning to end*, the authors describe the common method we tend to use when managing data sets for investigation, the *interactive mode*. This is mostly intuitive but has two main problems: replicability and efficiency. They describe via an example how the first is desired due to the long and tedious process of getting data-sets, transforming, cleaning and organizing them when working on a paper. The latter when one may need to change the empirical model and not having to turn the process upside-down. So they propose working with a *shell script* that logically organizes the scripts that manage the data and run the models.

- **Chapter 3: Version Control**

  Once again, (A) *Store code and data under version control* and (B) *Run the whole directory before checking it back in.* Based on the disadvantages on still working with a coauthor a 'date and initial' method sounds natural, yet it is tedious when several versions are stored and in is confusing, thus, the solution is *version control.* This method is based on creating a repository on a remote server and every time someone intends to modify something, just *check out*, change and *check back in.* The software keeps date organization and who made the changes. It maintains a single, authoritative version of the directory at all times. Rule (B) is so that every version that is checked back in runs with the shell script.

- **Chapter 4: Directories**

  Rules: (A) *Separate Directories by functions*, (B) *Separate files into inputs and outputs*, and (C) *Make directories portable.* The authors expose a way to organize the scripts in a consistent and logical manner in `/input`, `/code`, `/output` and `/temp` stages so that everything is neatly organized and to know what depends on what.

[†]Economics student at Universidad de los Andes, Colombia. URosario email: juana.rincon@urosario.edu.co. Uniandes email: ja.rincong@uniandes.edu.co.

- **Chapter 5: Keys**

  Rules: (A) *Store cleaned data in tables with unique, non-missing keys* and (B) *Keep data normalized as far into your code pipeline as you can.* The authors introduce the concept of *relational database* that self-explains the data-sets by the information in corresponding levels, this is also called *normalized* data, that is easier to understand and later modify and mutate.

- **Chapter 6: Abstraction**

  Rules: (A) *Abstract to eliminate redundancy*, (B) Abstract to improve clarity, and (C) *Otherwise, don't abstract.* In this case, abstraction is defined as turning the specific instances of something into a general-purpose tool. In the case example, when repeating an algorithm perhaps to create a variable, it is preferable to create a general function to use several times instead of copying and pasting the several code lines, which could lead to typos and mistakes. Although, sometimes less is more, being careful with abstraction is an art on its own.

- **Chapter 7: Documentation**

  Rules: (A) *Don't write documentation you will not maintain* and (B) *Code should be self-documenting.* Regarding the use of comments in code, it is on one hand desirable to consistently check for the comments so that everything works as intended and most of the time logical code is better than extensive comments.

- **Chapter 8: Management**

  In this final chapter, the rules explain it perfectly: (A) *Manage tasks with a task management system* and (B) *E-mail is not a task management system.* Workflow is very important and communication is key for an effective workflow, so a task manager between peers is recommended.

1.2 Importance of these elements to the authors

As explained in the neat summary in the past subsection, each and every principle provides a set of rules of thumb to address problems regarding increasing marginal costs when data-bases grow, when human errors can cause the functionality of the code to break, when organizing scripts is difficult (and so reading them), good practices inside code and in workflow. It is explained how each element solves one of these problems.

1.3 Example of a problems by not following these principles

Regarding the problem addressed on **Chapter 3: Version Control**, if one of the collaborators of the project were to modify the directory without notifying the others and failing to rename the new version accordingly (or without checking that the changes in one script don't affect the functionality of the whole directory), for one, regression results may change and be interpreted incorrectly. For example, if Scott has in mind to run a lin-lin model, but Carolina modifies the script and specifies a log-lin model, and Scott ends up running that model without knowing, his result tables will have log-lin results but be interpreted as lin-lin, affecting drastically the interpretation and possibly damaging the paper. This problem is as well related to chapter 4, 7 and 8.

1.4 Personal incorporation of principles to own work

Actually, reading this guide shed some light on the course I am taking as a research assistant for Professor Philipp Hessel at Uniandes. We are currently working with a very large data-set (ELCA) in order to find the effects of a subsidy on child development through longitudinal data. First, the problems we have encountered are in management, since we assign work through e-mail and WhatsApp and sometimes a bullet point is forgotten or omitted. Second, we share a single R Script that is updated almost instantly to a Cloud Service and refreshed to both out personal computers. The problem here is in version control, that there is only one version and considerable changes are lost forever. I believe we have the directories well organized and our script runs everything. But including these practices in our work should, most certainly, make our lives easier and more efficient.

2 Git

2.1 Basics of Git and GitHub: Differences and Similarities

Based on the information provided by McDermott, n.d., Git is a "distributed version control system" that helps economists and data scientist to be organized with their workflow by keeping track of changes to code whether made by one's self or a colleague. Now, just as one may use Overleaf to compile $\LaTeX$, PyCharm to run Python or RStudio to run R, GitHub is an online hosting platform that makes it easier (and more beautiful) to use Git.

2.2 Benefits of using Git and problems of not using it

Taking into account the lecture by Gentzkow and Shapiro, 2014, when working with RAs or coauthors being able to run a *version control* workflow basically saves lives. So there is a "single, authoritative version of the directory at all times" that everyone could check for changes and don't make a mess with several names for the same file.

A possible problem, as previously mentioned, using a 'date and initial' method to organize versions could completely ruin the directory and the costs of finding the errors overcome the benefits of using that method.

2.3 Challenges on using Git

Imagine the inherent costs of learning a new coding language or a new program. At first getting used to the interface, the commands and the logic behind these is quite difficult, yet, once overcame, every step along the way is even easier than the last one. An economist may think of it as having a decreasing marginal cost function of learning: the cost of learning something new of the program gets lower every time once you accumulate knowledge and proficiency. So that is the main challenge of using Git, that is has a learning curve but surely the benefits of using it overwhelm the costs of learning and as well the costs of messing up using more orthodox methods to organize data. How to overcome the challenge? Practice and having incentives to use it. This class is a great incentive because we are being pushed to understand the usage and logic behind Git, afterwards, most probably I will use it every single time. And, since I prefer using R as my predetermined statistical software, seamless integration with GitHub is already a win.

2.4 Four main Git operations

The for main Git operations are as follow:[1]

1. **Stage**: Tell Git that you want to add changes to the repository history (file edits, additions, deletions, etc.).
2. **Commit**: Tell Git that, yes, you are sure these changes should be part of the repository history.
3. **Pull**: Get any new changes made on the GitHub repository (i.e. the upstream remote), either by your collaborators or you on another machine.
4. **Push**: Push any (commited) local changes to the GitHub repository.

The main differences between these operations become clearer when actually using a software to connect local data to the online repository such as GitTower or GitHub Desktop. And integration is mostly seamless, since you can work and modify the code that is in the repository, push and when accessing the repo (not to be confused by a *repurchase agreement* in finances), all the information on who did the changes and date is stored.

References

Gentzkow, M., & Shapiro, J. (2014). *Code and data for the social sciences: A practicioner's guide.* http://faculty.chicagobooth.edu/matthew.gentzkow/research/CodeAndData.pdf

McDermott, G. (n.d.). *Data science for economists - lecture 2: Version control with git(hub).* https://raw.githack.com/uo-ec607/lectures/master/02-git/02-Git.html#57 (accessed: 09/06/2020)

[1]Since the exposure of McDermott, n.d., is utterly fine, I use the same definition as in his slides.