Setting Up a Dofile for Team Research[†]

Ryan McWay[1], James Allen IV[1,2,3], and Hang Yu[4]

[1]Population Studies Center, University of Michigan
[2]Economics Department, University of Michigan
[3]Ford School of Public Policy, University of Michigan
[4]National School of Development, Peking University

Version: October 4, 2021
Latest Version: **Click Here**

Abstract

Co-authoring Dofiles can be challenging as most Stata users have idiosyncratic preferences and methods for organizing and writing Dofiles. Which standards and practices can research teams adopt to improve the cohesion of this group work? This article proposes some best practices to overcome team research coordination issues adapting methods from software engineering and data science along with personal experience with group research. We prioritize improvements that increase efficiency of the team workflow by establishing global parameters and directories, standardizing communication between team members, and enabling reproducibility of results.

JEL Classification: C89, Y20, Y80

Keywords: Team Research, Workflow, Reproducibility, Dofile, Stata

[†]Corresponding Author: McWay, rmcway@umich.edu. We appreciate feedback from Rebecca Reilly, Wallace Pupadopoulos, James Zumel Dumlao, and Jared Stolove. And we would like to especially acknowledge Dean Yang for his support. All remaining errors remain our own.

While the Stata User Guide lays out a considerable taxonomy and documentation for coding in Stata and Mata languages (StataCorp, 2021), the idiosyncratic preference and habits of individual Stata users often do not meld well when placed in a group environment. As academic work has become increasingly group-oriented in the social sciences (Parish et al., 2018; Rath and Wohlrabe, 2015), a lack of consensus on best practices in group empirical work has become a predominate friction in research productivity. Pulling from personal experience, training, and current practices established in the programming and data science fields, this article presents practices useful for improving group research via Stata.

Previous work has been written on improving coding skills in social science in general (Gentzkow and Shapiro, 2014) or specifically in Stata (Cox, 2001, 2014; Long, 2009; Acock, 2009; Baum et al., 2010)[1], as well as the integration of Stata into platforms designed for collaboration (Hicks, 2014; Haghish, 2020). Building on these tips and tricks, we note that standardization of best practices for coding in teamwork remains a warranted addition in this catalogue.

In what follows, we discuss methods within a Dofile to increase efficiency of the team workflow by establishing global parameters and directories, standardizing communication between team members, and enabling reproducibility of results. While some may be worth adopting as is, it is not hard to imagine adaptations of the following examples which may best suit the needs of a particular team's work. What we hope to impress is the desire to implement similar standards within your team's group work.

1 Global Parameters and Directories

Global parameters and directories are set in the preamble as the start of a Dofile. Listing 1 offers an example of how to start a preamble, by establishing a common version, the maximum settings and memory, along with a seed for any further randomization which helps to standardize the script before anything has begun.

Listing 1: Setting Parameters

```
// Set Stata Version
version 15
// Clear everything in memory
clear all
// Set limits
    set maxvar 30000
    set matsize 11000
    set more off
```

[1]Eminently Nick Cox's 'Speaking Stata' series and J. Scott Long's 'The Workflow of Data Analysis Using Stata'.

Once the parameters of a script are set, the preamble can set up automating changes in the root directory. This produces an efficient way for multiple members to quickly jump into the code. Through a basic *if* statement, team members can use their Stata *username* (derived from the local's user name) to efficiently switch between root directories by user as shown in Listing 2.

Listing 2: Automate Switch Between Root Directories

```
// Team Member Directories
    // 1st Member's Directory
    if "`c(username)'" == "member1" ///
        global dir "C:/Users/member1/project_folder"
    // 2nd Member's Home Directory
    if "`c(username)'" == "member2home" ///
        global dir "C:/Users/member2home/project_folder"
    // 2nd Member's Work Directory
    if "`c(username)'" == "member2work/project_folder" ///
        global dir "C:/Users/member2work/project_folder"
```

Once the root directory is established, the preamble can establish *globals* or *locals* that provide a succinct reference for the extension of directory paths used throughout the Dofile. Complied together in the preamble, this provides clear navigation of the inputs and outputs from a Stata script in commonly structured or shared folders. Listing 3 shows an example of shortening folder extensions into a simple macro given a shared or similarly structured directory.

Listing 3: Establishing Folder Paths

```
// Script Paths
    global dir_scripts  "$dir/scripts"
// Data Paths
    global dir_df_raw  "$dir/data/raw"
    global dir_df_clean  "$dir/data/clean"
// Output Paths
    global dir_tables  "$dir/output/tables"
    global dir_figures  "$dir/output/figures"
```

Lastly, automating version history is important when many members are adapting and modifying the same script (often without consulting each other). Copying a timestamped version of the script is an easy way to track and manage changes. Listing 4 shows a means of capturing the current date, and creating a copy of the script prior to running the analysis (in this case stored in a sub-folder named 'dated'). When placed in the preamble or at the end of a Dofile, this script will update the timestamped version whenever the Dofile is saved and executed and capture the last update at the end of the day, though the script can be easily adapted to use a more or less frequent timestamp. This accompanies version control measures alongside Dropbox, Github, and Git.

Listing 4: Dated Version of a File

```
    // Establish Current Date as (YY/MM/DD)
        global date = strofreal(date(c(current_date),"DMY"), "%tdYYNNDD")
    // Example of Creating Dated Version
        copy "${dir_scripts}/dofile_name.do" ///
            "${dir_scripts}/dated/dofile_name_${date}.do", replace
```

It is worth noting that Listings above fit the best into standalone Dofiles. When adopting them to a more complicated scripts system with one master Dofile executing multiple sub-Dofiles, global parameters should typically only be set in the master Dofile to avoid inconsistencies.

2 Consistency & Communication

Naming and Labeling

As collaborators are added to a project, each contribute their own expertise expressed through uniquely formatted syntax. This is nowhere more evident than in the naming conventions they choose and the labels they provide. Instating consistency improves functionality. To this end, it is best practice that a project remain consistent in both naming conventions as well as labeling. Deciding upon Snake Case or Camel Case[2] for variable names (as promoted in the programming world (Oracle Corporation, 2021; Python Software Foundation, 2021; Baath, 2012; Sharif and Maletic, 2010; Suzuki et al., 2014)) allows partners to decipher the purpose and meaning of a variable quickly within a script. Generalizing the suffix and prefix of variables creates a categorization of variables into groupings, which allows for the quick filter through variables as well as provides a logical cut off to apply wildcard operators when calling a group of variables (e.g., *?* and *** in Stata).

When adding labels or notes to a variable, following an agreed upon system for truncating information creates a tautological convention for disseminating the information needed by multiple members of a team. For example, a variable holding the geographic identification of a unique observation may have the label "ID: Latitude" with similar variables in this family following a similar pattern, such as "ID: Longitude." By agreeing upon a system of labeling, partners can add new variables with labels providing information in a manner consistent across its group allowing for a quick understanding of its utility.

[2]Snake Case strings a variable name along with '_' as a delimiter, e.g., 'snake_case_name'. While Camel Case does not separate the parts of a variable name and instead relies on capitalization for punctuation, e.g., 'CamelCaseName'.

Comments

Quality communication is an underrated attribute of any research project. This extends to the communication left within the Dofile to one's own future self as well as collaborators. The mantra of any good scripter or programmer is to 'Comment Often.' That is to say that your Dofile should be littered with comments on the purpose of a chunk of code, as well as questions or concerns to look out for within the script. When performed with intention, comments can provide a narrative within a script to allow even the most ancillary team member to securely understand each functionality throughout a Dofile.

Beyond a typical comment to mark the function of a series of lines of code, there are two comments with service to further communicate important information in a research team setting. The first is a 'TODO' comment. Adopted from programming best practices (Storey et al., 2008; Nie et al., 2018; Sridhara, 2016), these are comments that start with 'TODO' and function as a reminder or suggestion for pseudo-code to come. In place of imparting functional code in that moment, leaving a TODO comment provides a signal of what needs to be added or addressed in a particular part of the script. TODO comments can also be directed at individual team members by writing 'TODO @NAME', which provides an easy way for team members to search for the Dofile for specific TODO comments assigned to them.

The second is a 'CHECK' comment. The purpose of 'CHECK' comments is to draw attention to and explain code written to test that changes made by contributors in other sections of the script do not unwittingly alter assumed properties of the data in a later section of code. These checks can be forced, as when Stata's *assert* command or *merge*'s *assert* option can prevent execution of the code beyond a certain point if a key condition is not met; for these, a 'CHECK' comment should justify why the *assert* should hold. For less critical matters, a 'CHECK' comment can also be a chunk of code purposely commented out to be run with discretion to verify certain properties of the data in memory or the accuracy of the scripts operations. Such 'CHECK' comments can help all team members be vigilant for and prevent against dramatic alterations to the output of a script when making contributions in earlier sections of the Dofile.

Sectioning

Beyond increasing the frequency and intention of comments for communication, the division of a Stata Dofile into sections allows for natural breaks in dividing the work via its natural structure. These sections can be anticipated ex-ante, and can communicate the responsibility of certain additions by different members of a team. In this way, team members can work on different sections concurrently without breaking the code when running an entire Dofile (E.g., sections 'in progress' or 'under construction' may lead to such a break). The use of macros with an 'on'/'off' switch in the preamble of a script using *if* statements encompassing a section allows a script to be parsed into the portions intended to be run. Listing 5 provides an example of toggling sections in this manner within a Dofile.

Listing 5: Sectioning a Dofile

```
// Declare Sections
    global sect_raw             "on"
    global sect_clean           "off"
    global sect_summstats       "off"

// Example of Toggling a Section On
if "$sect_raw" == "on" {
    display "This section only runs if $sect_raw is toggled"
}
```

3 Reproducibility

The reproducibility of work has not only become a major concern considering the reproducibility crisis (Baker, 2016; Open Science Collaboration, 2015; Camerer et al., 2016), but remains an important consideration when working amongst others who may monitor or update shared works. Within a Dofile, there are a few best practices that can be taken to improve the reproduction of a script by any collaborator.

Log files are a useful way of documenting the printed output from the commands run in a Dofile. This is a documentation of the output as 'you see it', as well as a reference of the work that was done for the future which may be cut from the final manuscript. Capturing the content in a Dofile and translating a log file into PDF format provides an informal documentation of the analysis run which can be referenced or shared as needed. Listing 6 displays an example of how to implement this automatically within a script.

Listing 6: Translating a Log File

```
// Start to Capture Log File as a SMCL Before You Begin the Analysis in
    the Script
    cap log using "$dir_log\dofile_name_${date}.smcl", smcl replace

// ... Some Analaysis Here ...

// Close Log File and Translate SMCL into a PDF at the End of the Script
    cap log close
    translate "$dir_log/dofile_name_${date}.smcl" ///
        "$dir_log/dofile_name_${date}.pdf", replace
```

As the demands of a project necessitate tailored user submitted Adofiles, it is important to ensure that the script enforces those dependencies are installed. To prevent the script from breaking midway because a member has not yet downloaded a needed package to call a command, declaring community contributed packages in a Dofile ensures a successful run. Placed in the preamble, Listing 7 demonstrates a means of flagging the packages needed to run a Dofile before starting the analysis.

Listing 7: Declaring Dependencies to Run Dofile

```
// These are the packages you will need to run this script
local package_list "names of community contributed commands (e.g., gsort, outreg2, etc.)"

// Flag at the beginning to download Ado files needed to run script
foreach package of local package_list {
    cap which ‘package’
    if _rc di "This script needs -‘package’-, please install first (try -ssc install ‘package’-)"
    }
```

A large project usually involves Dofiles that use outputs of other Dofiles. To ensure that each team member uses consistent, up-to-date input datasets, it is useful to demonstrate in the preamble the position of the current Dofile in the input-output Dofile-network. Listing 8 is an example statement of data inter-dependency:

Listing 8: Inter-dependency of Dofile Network

```
/*
This dofile uses the following datasets as inputs:
    "$dir_df_raw/household_raw.dta"
    "$dir_df_clean/individual_clean.dta"

Execute the following do-files to update the input datasets when needed:
    // Generate "$dir_df_clean/individual_clean.dta"
    do "$dir_scripts/gen_ind_clean.do"

This dofile generates the following outputs:
    "$dir_tables/main_regression.xlm"
    "$dir_figures/main_trend.png"
*/
```

Different researchers may prefer different forms of output for their results, and may differ at different stages of research. Manuscript drafts are being created ever more commonly in LaTeX, but seeing the results as an Excel output may be simpler or quicker at certain stages of the research. Incorporating flexibility into the Dofile so that the results can quickly and easily transition from one form of output to another allows the coder to more easily respond to requests from the research team.

4 Beyond the Dofile

That mentioned above is best accompanied by established standards and practices outside of the Dofile script itself. These include a 'Master' Dofile calling in sequence all of the Dofiles needed to complete a project from start to finish, a form of version control, an intuitive file folder structure, a consistent file naming convention, and coordination on editing Dofiles via Github or an Integrated Development Environment (IDE) that provide management of the merger of different branches into the master file.

While contextualized for team research in Stata, this commentary on best practices is not restricted to academia or Stata users. Much of the content mentioned extends to other forms of collaboration or other statistical softwares insofar as it can easily be adapted to serve similar functions. These best practices are intended to increase the efficiency and productivity of the use of Stata Dofiles. Implement them at your discretion to advance your own work.

References

Acock, A. C. (2009). Review of The Workflow of Data Analysis Using Stata, by J. Scott Long. *The Stata Journal 9*(1), 158–160.

Baath, R. (2012). The State of Naming Conventions in R. *The R Journal 4*(2), 74–75.

Baker, M. (2016). 1,500 Scientists Lift the Lid on Reproducibility. *Nature 533*(7604), 452–454.

Baum, C. F., M. E. Schaffer, and S. Stillman (2010). Using Stata for Applied Research: Reviewing its Capabilities. *Boston College Working Papers in Economics* (764).

Camerer, C. F., A. Dreber, E. Forsell, T.-H. Ho, J. Huber, M. Johannesson, M. Kirchler, J. Almenberg, A. Almejd, T. Chan, E. Heikensten, F. Holzmeister, T. Imai, S. Isaksson, G. Nave, T. Pfeiffer, M. Razen, and H. Wu (2016). Evaluating Replicability of Laboratory Experiments in Economics. *Science 351*(6280), 1433–1436.

Cox, N. J. (2001). Speaking Stata: How to Repeat Yourself Without Going Mad. *The Stata Journal 1*(1), 86–97.

Cox, N. J. (2014). Speaking Stata: Self and Others. *The Stata Journal 14*(2), 432–444.

Gentzkow, M. and J. M. Shapiro (2014). Code and Data for the Social Sciences: A Practitioner's Guide.

Haghish, E. F. (2020). Developing, Maintaining and Hosting Stata Stastical Software on GitHub. *The Stata Journal 20*(4), 931–951.

Hicks, R. (2014). Stata and Dropbox. *The Stata Journal 14*(3), 693–696.

Long, J. S. (2009). *The Workflow of Data Analysis Using Stata*. StataCorp LLC.

Nie, P., J. J. Li, S. Khurshid, R. Mooney, and M. Gligoric (2018). Natural Language Processing and Program Analysis for Supporting ToDo Comments as Software Evolves. *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Open Science Collaboration (2015). Estimating the Reproducibility of Psychological Science. *Science 349*(6251).

Oracle Corporation (2021). The Java Tutorials.

Parish, A. J., K. W. Boyack, and J. P. A. Ioannidis (2018). Dynamics of Co-Authorship and Productivity Across Different Fields of Scientific Research. *PLoS One 13*(1).

Python Software Foundation (2021). PEP 8 – Style Guide for Python Code.

Rath, K. and K. Wohlrabe (2015). Recent Trends in Co-Authorship In Economics: Evidence From RePEc. *Applied Economics Letters 23*(12), 897–902.

Sharif, B. and J. I. Maletic (2010). An Eye Tracking Study on camelCase and under_score Identifier Styles. *Proceedings of the IEEE 18th International Conference on Program Comprehension*.

Sridhara, G. (2016). Automatically Detecting the Up-To-Date Status of ToDo Comments in Java Programs. *ISEC '16 Proceedings of the 9th India Software Engineering Conference*, 16–25.

StataCorp (2021). *Stata User's Guide: Release 17*. StataCorp LLC.

Storey, M.-A., J. Ryall, R. I. Bull, D. Myers, and J. Singer (2008). TODO or to Bug. *2008 ACM/IEEE 30th International Conference on Software Engineering*.

Suzuki, T., K. Sakamoto, F. Ishikawa, and S. Honiden (2014). An Approach for Evaluating and Suggesting Method Names Using N-Gram Models. *Proceedings of the 22nd International Conference on Program Comprehension*, 271–274.