Tips to produce a clear and organized do-file

(updated from https://thedatahall.com/stata-do-file/.
Also you can watch this tutorial: https://www.youtube.com/watch?v=T5G9fT64DRw)

Stata users should develop a habit of working in Stata do file rather than the command box since this allows you to continuously save your code without the danger of losing hours of hard work. Remember that you can create commands using the drop-down menu, and stata will keep those commands in the review window.
Some important tips help us enhance our experience when using do files, some of which I summarize below.

1-The do file should always start with the version command

This lets Stata know which version it should base its interpretation of your command on. If you wish for this to be version 17 of Stata, simply type:

version 17

Some commands (or their syntax) are specific to older versions of Stata and are not supported by newer versions. If you want to use a command from an older version, the version command will let you use it in a newer version. Your commands will then be executed as they would have been in the older version of Stata. This is especially a good idea in hindsight because you may want to execute an older do file months (or years) later with a new version of Stata, but you may not remember which version of Stata you wrote that command in. A version command written at the start of the do file will make the execution of any code smooth, regardless of which new Stata version you are running it on.

2- Add header to do file

Always add a header to your do file to indicate what the do file's purpose is. Specify a project title, its description and all the datasets used in the file. Also add a date and author name if it is to be used by multiple people.

```
*********************************************
* Project Title:                            *
* Description:                              *
* Datasets used:                            *
* Author:                                   *
*********************************************
```

3-Use Relative Paths in Stata Do file

To understand relative paths, let's first understand absolute paths. Absolute paths are those paths that give the exact location of a file. For example:

use "E:\\Stata\Project1\data1.dta", clear

merge 1:1 symbol using "E:\\Stata\Project1\example\data2.dta"

The above code specifies the entire directory where datasets 'data1.dta' and 'data2.dta' are stored. If you share this do file with another co-author, or you decide to run it yourself on a different computer system, chances are these datasets will not be saved in the exact path you specified initially. For example, your co-author may not have the datasets saved in a folder called 'Project1' like you. The above commands will not run on their computer.

To get around this issue, we use relative paths that allow such command to run on any computer. There are two ways to use relative paths:

Changing the working directory of the Stata file we are working on.

- Using global variables
- Changing the Working Directory

To change the working directory, we first specify the cd command before importing any dataset:

cd "E:\\Stata\Project1\"

This command changes our working directory to the folder 'Project1'. We don't have to specify this entire path when we import or merge data sets anymore. 'data1.dta' can simply be loaded by:

use "data1.dta", clear

In case of datasets like 'data2.dta' which reside in a subfolder called 'example' within 'Project1', we only specify the subfolder's name. The rest of the path has already been taken care of by the cd command. We can merge 'data2.dta' using:

merge 1:1 symbol using "example\data2.dta"

Using this method, any author of the do file will only need to change the cd command according to whichever directory they have stored the relevant datasets in on their own respective computers.

Using Global Variables

Global variables allow us to define variables to store our directory path. For example, if we define a global variable called 'path' and store the directory in it, our syntax will look like:

global path "E:\\Stata\Project1\"

Once we store the directory path in a global, we only need to refer to its name, in this case, 'path', preceded by $ sign, when loading or merging datasets.

use "$path\data1.dta", clear

merge 1:1 symbol using "$path\example\data2.dta"

If someone wishes to change the directory, they only need to change the path specified in the global variable.

4-Use indentations or spaces

Stata code is not affected by spaces or indentations in the code. One should take advantage of this fact and use indentations to make code easier to read. For example, it is easier to identify which commands come within a loop in the following code because of good use of indentations:

```
forvalues i = 1/10 {
        display `i'
}
```

5-Set More Off

Oftentimes our output data cannot fit in Stata's Result window because of how long it is. Stata, therefore, pauses the execution of our commands and displays a "-more-" icon that needs to be clicked if we want to see the rest of the output. To tell Stata not to pause when displaying output i.e. not show a "-more-" option, we write the following command at the start of our do file:

set more off

This makes Stata display all the output at once without the user having to press anything.

To ensure that this setting is remembered next time you open Stata, add a permanently option to the command above:

set more off, permanently

Finally, if you do want Stata to show a "-more-" option when displaying long outputs in the Result window, simply use the command:

set more on

6-Log Files

Log files store anything that appears in Stata's Result window. This includes commands and their outputs. If you were to run a regression, both the regression command and the output table will appear in the log file. To start recording a 'log' of your work, type in:

```
log using example.log
```

This command opens a log file and records the commands we execute and their outputs. The command above writes the log file in text format. By default, Stata uses the extension SMCL (Stata Markup and Control Language) format.

7-Stand Alone Do Files

When creating a do-file, think of creating a stand-alone do file. This file will be able to execute entirely without any human intervention. This means that a do file can load a dataset, execute a set of commands, and then save the output; all within a single execution.

8-Grouping Data Management Commands

Commands used for data management and of a similar nature should be grouped together. This improves the readability of the do file. For example, when generating a number of variables, one should group these commands together. Similarly, commands that rename variables should also be grouped together.

```
*Grouping commands
generate abc=       1
generate abcdef=    2


rename abc          newabc
rename abcdef       newabcdef
```

9-Don't Abbreviate Too Much

While abbreviating commands makes our code concise and quick to write, we shouldn't go overboard with abbreviations. The command to create new variables called generate can be abbreviated as gen or g. Very short abbreviations like g often make it hard for others to understand what the function of a code is; while gen would be more comprehensible and concise at the same time.

```
*Dont abbreviate too much
generate abc=1
gen abc=1
g   abc=1
```

10-Different File Names for Input and Output Data Files

Input and output data files should be saved with different names. This is crucial in ensuring that our initial data file remains present in its original form. Any changes made to it after data cleaning or formatting should be saved as a new, differently named data file. It is very likely that you (or a coauthor) would need the original data file again, in which case it would come in handy to have it saved without any changes made to it.

11- Closing the Log File

When we execute a do file, they keep running until they encounter an error in executing the code. When this occurs, the do file stops running and any code beyond the error point remains unexecuted. One such error occurs when we are closing a log file.

We close/end a log file through the command:

log close

However, this results in an error when our log file is not open or we have mistakenly closed it. To get around this error, we precede the above command with capture:

capture: log close

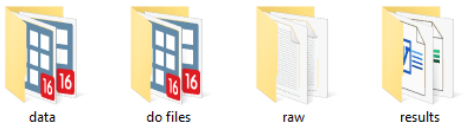
The capture prefix suppresses any errors that may occur during the execution of our do file, and will allow it to continue executing despite any errors. If there are other commands where we might require the use of capture, we can encapsulate it in the capture command like:

capture {

}

12- Organising Your Files

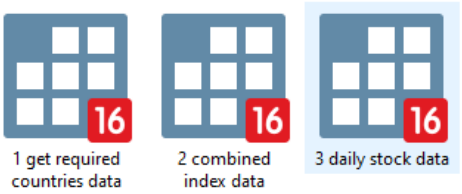
It is strongly recommended that you save your data files and do files in their respective subfolders that are stored in one main folder. You may also make a separate sub folder for your outputs, graphs and log files.



13- Numbering the Do Files

If your project involves working with multiple do files, it is a good idea to add a number in their names to reflect the sequence in which they need to be executed. For example, if a do file generates new data files, which in turn need to be loaded into a second do file, it would help to add a '1' to the first file, and a '2' to the name of the second do file to indicate the order of their execution.

Another takeaway from this tip is the advantage of having multiple do files for one project. If there are several sections of related but separate codes in your project, divide them into different do files. It brings ease into the process of debugging and reading code.



14-Adding Comments to Your Do File

The importance of commenting on each part of your code cannot be stressed enough. *It is not an exaggeration to suggest that your do file should have more comments than code*. Commenting adds value to your do file by saving you (and your coauthors) the time that it would otherwise take to make sense of the sequence and purpose of any piece of code written in it.

There are three ways to add comments in Stata:

1. Double forward slash – //
2. Asterisk – *
3. A forward slash and an asterisk – /* comment */

Double Forward Slash (//)

Anything written right after the double forward slash gets treated as a comment if it is in the same line as the forward slashes.

This can also be used to comment out actual Stata commands. For example, in the following example, the clear option is commented out and will not be executed.

use nocardia //, clear

Always add a space before the double forward slash, otherwise Stata treats them as part of the command and attempts to (unsuccessfully) execute them. The following is the wrong way to use the double forward slash:

Asterisk (*)

Asterisks provide limited commenting capability in that they only work if added at the very beginning of a line. You cannot add them in front of a command like we used the double forward slash.

use nocardia, clear *This is an incorrectly written comment and will return an error.

Asterisk comments can be used interactively, which means they can be written directly in Stata's command box and displayed in the Result window. Comments with double forward slash cannot be executed from the command box.

****This comment can be entered through Stata's command box or the do file.***

A Forward/Backward Slash and an Asterisk (/* */)

Any text encased between a forward slash and an asterisk will be treated as a comment. Not only that, this style of commenting also allows you to take your comment to a new line. You can therefore comment out multiple lines of text or code as one comment. To end the comment(s), you need to use an asterisk followed by a backslash. This commenting style can be used at the beginning, middle or end of a command or line.

```
/*This is the first line of this comment.
This is the second line of this comment.
use nocardia, clear
The command above is part of the comment and will not be executed.
This is the last line of the comment.
*/
```

15-Breaking very long codes

Oftentimes, our line of code gets very long and we have to resort to scrolling left and right to read it entirely. This is an inconvenience that interrupts the smooth reading of a do file.

We cannot use the enter key to move part of our command to a new line since Stata considers the new line as a new command.

Since version 13 of Stata, a line break in a long command can be created using three forward slashes:

reg price mpg rep78 headroom trunk weight lengt ///
turn displacement gear_ratio

The above command is treated as one line of code because the three forward slashes indicate that anything written in the next line is still part of the first line.

Since version 16 of Stata, the lines are wrapped in a new line if the current window size is too small for its entire length. This feature can be switched off from the 'View' drop down menu and unchecking the option called "Wrap lines"

There is also another way of taking our long line of code onto the next line. For this, we use the #delimit command and specify a character that will identify the end of a command. Most people use the semicolon (;) as a delimiter

#delimit ;

The above command tells Stata that anytime a semicolon (;) appears at the end of a line, it should end the command.

delimit ;
reg price mpg rep78 headroom trunk weight length
turn displacement gear_ratio;

In the code above, the two lines are considered as one command. The semicolon at the end of the second line signifies the point where the reg command ends.

To revert back to Stata's default settings, just enter the following command:

#delimit cr.

#delimit cannot be used interactively.

16-Consistency in Naming Variables and Labels

Always use a consistent style when naming variables and their labels. For example, all log variables should start with an 'ln_' prefix, and all standardized variables could start with an 'std_' prefix.

You do not have to label every single variable at the start of your project. This is only required for variables you will be outputting summary statistics for or those that you will be using in your regressions or other analysis.

```
*Stata version
version 17


*Add a header to your do file

*********************************************
* Project Title:                             *
* Description:                               *
* Datasets used:                             *
* Author:                                    *
*********************************************


*Use relative paths

*Absolute paths
use "C:\\vhm8120\nocardia.dta",clear

*Relative paths
*Method 1: Use change directory command
cd "C:\\vhm8120\"

use "nocardia.dta",clear

*Method 2: Use global variables
global path "C:\\vhm8120\"

use "$path\nocardia.dta",clear

*Stata don't mind spaces and indentation

//Wrong way
forvalues i=1/10 {
display `i'
}

//Right way

forvalues i=1/10 {
        display `i'
}


*More option
set more off
```

```
set more off, permanently

*Log files

log using "example.log"

*Grouping commands
generate abc=           1
generate abcdef=        2

rename abc                      newabc
rename abcdef           newabcdef


*Dont abbreviate too much
generate abc=1
gen abc=1
g       abc=1


*Different file names for input and output


*Closing log file
capture: log close
capture {
}


*Organizing the folder


*Numbering the do files

**********************
* Stata Comments   *
**********************

//Three ways to write comments in stata

/*
1. Double Forward Slash
2. Asterisk
3. Forward Slash and Asterisk
*/
```

//1. Double Forward Slash

//can be used at the beginning of a line or at the end of the line
//anything appearing after double forward slashes will not be executed as command
sysuse auto,clear //This command will load auto dataset
sysuse auto //,clear //Now clear option will not be executed

//Must have a space between command and forward slashes
sysuse auto,clear//Wrong way of doing it.

*2. Asterisk

*A single asterisk at the beginning of a line will turn the line to a comments
*can only be used at the beginning of line not at the end of the middle
*can be used interactively
*sysuse auto,clear
sysuse auto,clear *this asterisk sign won't work

/*3. Forward slash and asterisk

Any thing between "/" and "*/" will be considered comments

sysuse auto,clear

The above command will not be executed
*/

sysuse auto /*,clear
like double forward slashes in can be used at begining, middle or end of line.
Usually they are used to comment out long line of code or comments
*/

sysuse auto /*middle*/,clear

****
*Line Breaks *
****

//Three forward slashes
//only in do file not intereactively

```
regress price mpg rep78 headroom trunk weight lengt turn displacement gear_ratio

//Using delimit instead of forward slashes
#delimit ;

regress price mpg rep78 headroom trunk weight lengt
turn displacement gear_ratio;

#delimit cr.
```