

The Do's and Don'ts of a Do-file

Emmanuel Milet*

This “guide” for lack of a better word is designed to help you write your do-files. Your co-authors will be happy and by the time the referee report from your paper arrives, you'll be able to quickly find the do-files that produce the results you included in your paper. Before going further into the ABCs of do-file writing, it is important that the folder in which you are working is properly organized.

Organization of the folder: Do-files, datasets, graphs, logs, tables, pdfs, etc. should be saved in different sub-folders. This will make your life much simpler than having everything in a single folder. You may be producing a large number of tables, graphs, do-files, etc. over the life time of your paper. If all those are thrown into the same folder, it becomes very difficult to remember (or simply find) which is the right graph to include in the paper, or which file has the last regression results.

Master file: The first thing to do when starting a new paper is to write a do-file which is not really a do-file, but a master-file. This is a do-file that calls other do-files. The purpose of the master file is to organize the sequence in which your do-files are executed. It is especially useful when you have many do-files and have lost track of what each do-file does (and why). A typical master file looks like this:

```
1          * -----
2          * Wage Premium paper: master file
3          * -----
4          *
5          clear*
6          set more off
7          global sysdate=c(current_date)
8          global path "C:\Users\milet\paper\Wage_premium" // Directory Unige
9          *global path=c(pwd) // Directory Baobab
10         cd $path
11         *
12         *
13         *
14         *
15         * Data Creation
16         do 1_make_data.do // loops overthe raw data, extract variables, and make the final dataset
17         do 1_period_definition.do // selects the surveys that we need to define the various periods
18         do 2_labels.do // puts the labels in the final dataset
19         *
20         *
21         * Compute the prices and quantities in terms of efficiency units
22         do 3_efficiency_units.do // efficiency units at the industry level (to get the elasticities of subst
23         do 3_panel_data.do // creates the dataset at the country*level with skill prem
24         *
25         do 3_raw_sk_premium.do // skill premium not in efficiency units
26         do 4_list_surveys_online.do // list of surveys in our dataset
27         do 5_regression.do // correlation between the skill premium and trade/fdi.
```

Several tricks are shown in this master file:

*University of Geneva. Contact: emmanuel.milet@unige.ch

1. Do-files start with a number: *0_master.do*, *1_make_data.do*, etc. One useful thing is to rank your do-files according to their contribution to your paper. First, you get the data set up, second you put labels to make sure you know what the variables actually do, then you produce stylized facts, and finally regressions.
2. The second command is: `global sysdate=c(current_date)`, which tells Stata to store in the global variable called “sysdate” the date of the day. This point will be particularly important for log files (see next paragraph).
3. The third command is: `global path=“C:\Users\paper\My_New_Paper”`. It tells Stata that (most of) the files you need for this paper are located in this folder, in which you have several subfolders (one for logs, graphs, results...).
4. There are comments after the execution of each do file, which you add using a double forward slash // **plus your comments**.

Log files: For every single do-file that you write, you should open a log file in order to keep track of everything you’re doing. Ideally, your do-file should start by opening a new log-file, and end with closing this log file. Here is an example of the four commands lines you may be to include at the beginning of every single do-file:

```
cap log close
set more off
cd $path
log using "./logs/1_make_data_`$sysdate`.smcl", replace
```

1. Close any existing log (otherwise Stata will not run)
2. Set the “more” off
3. Re-assign the correct working directory
4. Open a log.
5. The date of the day can be included in the name of the log file. This prevents you from erasing your log-file each time you run your program.
6. Log files should be stored in a specific folder.

Comments: Your do-file should be full of comments. The beginning of your do-file should be a description of what it does. Do not hesitate to be very descriptive, you may find this information extremely valuable in the future. Do not hesitate to include comments at each important step or operation that your do-file does. Explain why you are getting some variables from other data files, or why you created such a variable, etc. This may sound tedious at first because by the time you write your do-file you obviously know why you are writing it. But your memory may not be so reliable a year later when you have to revise your paper. Adding comments can also help you find errors in your codes. Once you are done with writing your do-file, there could (should) be more comments than actual coding. Here is an example:

```

cap log close
set more off
cd $path
log using "./logs/1_make_data_$(sysdate).smcl", replace
*
* -----
* This program merges together the following database:
*   GDP.dta
*   Conflict.dta
*   WDI.dta
* and create the final database used for descriptive statistics and regressions:
* dataset_final.dta
* -----

use GDP, clear
* we first take the log of some variables:
gen ln_gdp=ln(gdp)
gen ln_pop=ln(pop)
gen ln_gdpcap=ln_gdp-ln_pop // we compute the GDP per capita
*
sort country year
bys country: egen gdp_mean=mean(gdp) // compute the (unweighted) GDP average in each country

*
** end of file
log close

```

Variable names and labels: It is important to give explicit names to your variables and assign labels to them.

An important thing when creating new variables is to be consistent in how you name them. You should be able to know what is behind a variable just by looking at its name. For example, if you create the log of a variable, you can use “ln” or “ln_” as a prefix for the new variable. There is no right way to do it, but consistency in how you name your variables will make your life easier.

Labelling variables is also very important to remind yourself what variables do, and also to have a nicer display in your tables for instance. I personally recommend having a separate do-file that only takes care of this, in your tables for instance. It prevents you from labelling variables that you decide to drop later on.

Regression tables: A quick way to get nice tables is to use the **eststo** and **esttab** command (available at ssc install, or by typing **findit esttab** in the Stata command). These two commands are very complete, and a full description of what they do is beyond the purpose of this note. The **eststo** is used to **store** the **estimation** for a regression, and the **esttab** creates a **table** based on this stored **estimation**. Many options are available, but the most used one are:

- Choose the significance levels and the corresponding symbols (the one most commonly used are stars, but they could also be a, b, c for 1%, 5%, 10%)
- Save the table in a specific file (many formats are available: txt, tex, html, csv ...)
- Display labels, compress the format

Here is an example of such code. Four regressions are estimated, and results are stored using the `eststo` command. They are first displayed in Stata, and then saved in the `results` folder. The command `eststo clear` erases the estimates currently stored. It is important to always write this command after saving/displaying the regressions.

```
eststo: reg y x1
eststo: reg y x1 x2
eststo: reg y x1 x2 x3
eststo: reg y x1 x2 x3 x4
*
esttab, starlevels(* 0.1 ** 0.05 *** 0.01) b(%5.3f) t(%5.3f) label compress nogaps replace
esttab using "./results/baseline_`sysdate'", starlevels(* 0.1 ** 0.05 *** 0.01) b(%5.3f) t(%5.3f) label compress nogaps replace
eststo clear
```

1 Miscellaneous tricks

This last section deals less with the do's and don't's of a do-file. It is a short description of useful commands you should know and will make your life easier.

Set More Off: Stata is a software that wants to make sure you are working when it is. When the command you execute produces a lot of output on the Result window, Stata will – at some point – stop and wait for you to press the Enter key before proceeding forward. To avoid this, just write `set more off` at the beginning of your code or in the master-file.

Quietly: This command simply tells Stata not to show the execution of the command on the Result window. This is particularly useful when you want to create a lot of variables from a loop, or from a `tabulate` procedure (to get dummy variables for instance). Only use this command when you are absolutely sure of what the output looks like.

Collapse: This command collapses your database and calculates many statistics. The command is usually followed by the `by(varlist)` option, which allows you to get the statistics for various subgroups defined by `varlist`. Be careful, this command **does not** calculate weighted averages, despite the fact that you can ask for a (mean) and specify a weighting variable...

Tag(varlist): This command creates dummy variables that take the value 1 every time it encounters a different value of the specified variable or list of variables included in `varlist`. This is useful when dealing with datasets that have multiple dimensions. For instance, if you have a dataset has three dimensions (year, country and product), you can easily compute the number of years per countries, or the number of products per country, or per country and year.

Group(varlist): This command works with the same idea as the `tag` command. It creates a numeric variable which takes a different value for each occurrence it encounters an occurrence of a variable or of a group of variables defined in `varlist`. In panel regressions, string variables cannot be used as fixed effects and one way to get around this is to create a group variable for the string variable.

#delimit; By default Stata does not allow you to write a command on several lines. When it comes to graphs however this can become useful as the overall command can quickly become very long. This command basically tells Stata that the command ends with the symbol ; You can then write on multiple lines and add this symbol at the end of the last line. To get back to the standard way of writing, write #delimit cr

Here is an example, along with several useful options for graphs:

```
#delimit;
twoway (scatter b_owner year          if signif_owner==1,          msymbol(O) mcolor(navy))
       (scatter b_nocontract year     if signif_nocontract==1,    msymbol(Dh) mcolor(dkorange))
       (scatter b_self year           if signif_self==1,          msymbol(th) mcolor(sand)),
xtitle("") ytitle("") title("") yline(0, lpattern(dash))
legend(order(1 "Owner" 2 "No contract" 3 "Self-employed")
       |rows(1) region(lpattern(blank)))
       note("Reference group: workers with a contract", span)
       scheme(s1color);
#delimit cr
graph export "./graphs/premia.pdf", as(pdf) replace
```

The scheme(s1color) gets rid of the ugly hospital-like blue color Stata uses for the background of its graphs. The msymbol option changes the look of the markers (hollow/plain circles/diamonds for instance). The region(lpattern(blank)) gets rid of the box around the legend, which here is displayed on a unique row.

levelsof: This command allows you to loop over the **values** of a specific variables. The various variables are stored in a local variable that you can use in your loop. Here is an example of code where we plot the kernel distribution of the variable “lnwage” for each country. The first command line extracts the different values of the variable “countrycode”, and stores them into a local variable that we call “countrycode_local”. We then take every occurrence “c” of this local variable, and we plot the kernel density of lnwage just for that country.

```
levelsof countrycode, local(countrycode_local)
foreach c of local countrycode_local{
    kdensity lnwage if countrycode=="`c'"
}
*
```

Overall presentation: Finally, empty lines should be commentary lines, i.e. they should start with an asterisk. This prevents some random piece of code that you forgot to delete from being considered as code.