Do files in Stata

Contents

Why use do-files in Stata?

1. Do-files make programming replicable.
    a. Replicability confirms the integrity of the user's work and helps it stand up to questioning.
    b. Replicability enables others to help when the user cannot find why the program isn't working.
2. Do-files let the user better proof work and find errors.
3. Do-files help the user document research, especially through comments. This complements two other major documentation strategies: a clear and simple file structure and an occasional research note.
4. Do-files allow the user to separate different phases of work – data creation and data analysis, for example.

Making do-files robust

1. Make do-files self-contained
    a. A do-file should not rely on something left in memory
    b. A do-file should not use a dataset unless it loads the dataset.
    c. The do-file should not test coefficients unless it estimates those coefficients.
2. Use version control, i.e., include the command `version 12`, for example.
3. Include directory information.
4. Include seeds for random numbers.

5. If there are file dependences (needing to run one program before another) use a `master.do` file (a do-file that runs other do-files).
6. Never save over an original dataset

Make do-files legible

1. Use lots of comments
    a. Use comments as dividers to set apart different parts of the program.
    b. Use a header comment at the top to leave notes about
        i. author, last update, what the update accomplished, and information about the overall goal of the do-file
        ii. any hard-coded paths (file locations such as c:/Users/Johnb/…etc.) that will have to be changed if moved off of this computer
        iii. any ado-files needed for this program to work
    c. Use commands as reminders of tasks remaining to be done.
    d. Use a comment to describe the goal of each step of the program
2. Use alignment to align variables from one line to the next
3. Use indentation to make reading easier
    a. If a command takes more than one line, indent subsequent lines (`Alt+e f` then set **Auto-Indent**)
    b. Use one indent per level of a loop
4. Use short lines (`set linesize 80`)
5. Limit abbreviations to the most common so that others can read the code.
6. Be consistent.
7. Name do-files according to their function, e.g. `create[name].do` or `analyze[name].do`

A sample template for do-files:

```
capture log close
set more off
log using _name_, replace text
// program:  _name_.do
// task:
// project:
// author:    _who_/ _date_
// #0
// program setup

version 12
clear all
macro drop _all
set linesize 80

*my commands start here
// #1
// describe task 1
// #2
// describe task 2

log close
exit
```

Most common do-file errors

1. Not closing a log from a previous do-file. This can be solved by including the command `capture log close` at the beginning of the do-file and `clear`, or close Stata.
2. Mistyping a variable name. This can be prevented by usine Stata's autocomplete function, or highlighting names in the variable window and hitting the left-up arrow to paste into the command line and then copy into do-file. It can also be prevented by having variable names that are completely distinct from each other.
3. Confusing = with ==. A simple rule of thumb: when `if` is used, `==` is the equality that should be used (not =).

4. Mistyping a command.
    a. This can be noticed immediately because the Stata editor changes the font on correctly typed commands to blue.
    b. Check for simple typing errors such as using `tabl` instead of `tab1`
5. Getting a `Type mismatch` error. This is often due to a string and a numerical variable being confused and usually has a wrong missing valuation (the missing symbol for a numerical variable is a dot `.` while the missing text is for a text (string) variable is two parentheses “”
6. Forgetting to add `, replace`
7. Getting an option error:
    a. Using an invalid option. This can be prevented by typing `help [operation name]` to see the allowed options and their syntax .
    b. Forgetting to precede an option statement with a comma
    c. Forgetting that an option requires an order of operations (e.g., sort data using `by` or `bysort`)
8. Putting `by` in the wrong place (for some commands `by` goes before, some after, some can be either)
9. Getting errors writing loops. This can be prevented by using `trace` which shows Stata as it interprets and evaluates each step.

Troubleshooting do-file errors

1. Run the program a few commands at a time. This can be done simply by commenting out (typing the comment symbols before and after) sections to be omitted.
2. Run `clear all` and `macro drop _all`
3. Restart Stata
4. Reboot computer
5. Attempt the analysis with new made-up data to test if there is a problem with the variable.

6. Rewrite the do-file from scratch if a substantive error (not a Stata error) is appearing.

Checking for clerical errors

1. Compare results to tables or summary statistics to diagnose potential problems from simple recoding operations such as `gen, replace,` etc.
2. Use Stata's conditional function to create a list to check for errors. For example, check that `tab id list if id !=0` Then do opposite `tab id number ==0`. Or `tab id list if id ==entry` to check that the entry number and ID number are not the same

J. Scott Long, 2009. The Workflow of Data Analysis Using Stata, Stata Press.
Jennifer Earl, Programming in Stata, Sociology 596A course, University of Arizona