The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–4

Stata Tip 127: Use `capture noisily` groups

Roger B. Newson
Department of Primary Care and Public Health, Imperial College London
London, United Kingdom
r.newson@imperial.ac.uk

1 The main idea

Did you know about the `capture noisily` group? It is defined as a group of Stata commands, starting with `capture noisily` { and ending with }. The commands in between are executed as usual, producing the standard output (because of the `noisily`). And, if any one of these commands fails, then execution resumes with the command immediately following the group (because of the `capture`). If you don't like typing `capture noisily`, then you can abbreviate it to `cap noi`, or even to `cap n`.

A simple application is where the user wishes to generate a Stata log file, which the user hopes to inspect afterwards, whether or not the logged commands work (and especially if they don't). In a do–file, the user may open the log file, and begin the `capture noisily` block, using the commands

```
log using mylog.log, replace
capture noisily {
```

and then add a sequence of Stata commands, such as

```
sysuse auto, clear
regress mpg weight
predict mpghat
twoway scatter mpg weight || line mpghat weight, sort
```

and then end the `capture noisily` block, and close the log file, using the commands

```
}
log close
```

The commands inside the block will then be executed until one of them fails (or until all of them end execution, if none fail), and their output will be stored in the file `mylog.log`. Whether or not any of the intervening commands fail, the log file `mylog.log` will be closed by the `log close` command. The user may then inspect the log file with a text editor, and view the results if execution was successful, or find out what went wrong otherwise. Typically, the number of Stata commands inside the block will be more than the four used here, and there may be program loops and other complicated programming constructions (see [P] **forvalue** and [P] **foreach**), increasing the probability of a failure somewhere.

2 Application in estimation command files

The `capture noisily` prefix is commonly used in do–files containing sequences of estimation commands. If the user is worried that one or more of them might fail (possibly because of insufficient observations), then the user may add a `capture noisily` prefix to each estimation command, so that, if one estimation command fails, then Stata resumes execution, starting with the next estimation command. And, if each estimation command is followed by one or more post–estimation commands (such as `predict` or `margins`), then each estimation command, and its own subsequent post–estimation commands, may be placed in its own `capture noisily` group. That way, if either the estimation command or the post–estimation commands fail, then Stata will continue to the next estimation command.

For instance, in the `auto` data, a user might want to fit a regression model of mileage (`mpg`) with respect to each of the car–size variables `weight`, `length` and `displacement`, together with the factor `foreign`, indicating whether a car model is made by a non–US company. After each regression model, the user might want to estimate the mean mileages expected if all cars were US models, and if all cars were non–US models, assuming that the car–size variable was distributed as in the real–world sample. The code to do this might be as follows:

```
sysuse auto, clear
describe, full
foreach X of var weight length displacement {
  capture noisily {
    regress mpg ibn.foreign `X´, noconst vce(robust)
    margins i.foreign
  }
}
```

As it happens, this code executes without any failed commands (not shown). However, if (for any reason) the analysis with respect to `weight` had failed, either in the `regress` command or in the `margins` command, then Stata would have proceeded to the analysis with respect to `length`. And, if the analysis with respect to `length` had failed, then Stata would have proceeded to the analysis with respect to `displacement`. This feature of `capture noisily` blocks can be very useful if the user is executing a long list of multi–step analyses, especially if these analyses involve commands with a higher failure probability than `regress`. Note that, if a multi–step analysis fails at an earlier command in a `capture noisily` block, then the later commands in the same `capture noisily` block are not attempted. Note, also, that, if the multiple analyses are simply the same command executed on multiple by–groups, then you do not need the `capture noisily` block, because you can use `statsby` (see [D] **statsby**).

3 Application in file generation programs

`capture noisily` may also be used to good effect in file generation programs. For example, it is used internally by the `dolog`, `dologx` and `dotex` packages, which can be downloaded from SSC, and used to execute a do–file while automatically generating a log file. However, more advanced users may want to write output to an arbitrary file, using the `file` command documented in [P] **file**. For instance, the new file may be a TeX, HTML, XML or Rich Text Format (RTF) file, produced as an automatically–generated report for a reproducible–research project. The user may be using a sequence of commands to generate this new file, and may want to close the file after executing those commands, whether or not they all work. The generated file will then be available for inspection by the user (although it may be incomplete), and the user will not have to close it manually. The `capture` block may begin with the commands

```
tempname buff1
file open `buff1´ using "myoutput.txt", write text replace
capture noisily {
```

and contain any amount of intervening code, including `file write` statements, such as

```
file write `buff1´ "Hello, world!!!!"
```

and end with the commands

```
}
file close `buff1´
```

In this case, a new file `myoutput.txt` is created with a buffer, whose name is stored in the local macro `buff1`, and filled with output from the intervening `file write` statements. However, if any statement in the intervening code fails, then Stata executes the `file close` statement, and the new file (usually incomplete) is available for inspection by the user.

Alternatively, the `capture noisily` block may be preceded and followed by file opening and closing commands other than `file open` and `file close`. For instance, if you are generating a HTML file, then the file opening and closing commands might be the `htopen` and `htclose` modules of the `ht` package (see Quintó et al. (2012)), or the `htmlopen` and `htmlclose` modules of the SSC package `htmlutil` (see Newson (2017)). Or, if you are generating a RTF file, then they might be the `rtfopen` and `rtfclose` modules of the SSC package `rtfutil` (see Newson (2012)). And more user–written file–generating packages are likely to be written on similar lines in the future, possibly for generating files in XML–based document formats yet to be invented. Such future packages are likely to contain their own file–opening and file–closing modules, suitable for use before and after a `capture noisily` block, respectively.

4 References

Newson, R. 2017. HTMLUTIL: Stata module to provide utilities for writing Hypertext Markup Language (HTML) files. Statistical Software Components S458085, Department of Economics, Boston College. http://econpapers.repec.org/software/bocbocode/s458085.htm.

Newson, R. B. 2012. From resultssets to resultstables in Stata. *Stata Journal* 12: 191–213.

Quintó, L., S. Sanz, E. D. Lazzari, and J. J. Aponte. 2012. HTML output in Stata. *Stata Journal* 12: 707–717.

About the author

Roger B. Newson is a medical statistician in the Department of Primary Care and Public Health at Imperial College London, UK. He wrote a number of SSC packages, including `dolog`, `dologx`, `dotex`, `listtab`, `htmlutil` and `rtfutil`.