The Stata Journal (2020)
20, Number 2, pp. 499–503 DOI: 10.1177/1536867X20931028

Stata tip 138: Local macros have local scope

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

1 Introduction

Stata programs, usually, and other Stata code chunks, commonly, use local macros. This tip focuses on a common misunderstanding of how local macros work. The executive summary is given in the title.

The best introduction to local macros remains material in [U] **18 Programming Stata**, but the tip is written to be self-contained as far as possible.

2 What is a macro?

The term *macro* has various meanings, even within programming, let alone beyond. In some software, it has informal or even formal meaning as indicating a routine or script, typically containing a series of instructions or statements in that software. Be that as it may, in Stata a macro is a named container whose contents are text. A macro is just one item, although occasionally that item may be very large.

Stata's use of macros is not unusual, either. It is akin in spirit to other languages, particularly languages like C or those in the Unix tradition (Kernighan and Pike 1984; Kernighan and Ritchie 1988).

Macros in Stata have two flavors, global and local. Part of the misunderstanding discussed here is not realizing that those names are not just arbitrary jargon but explain the scope of the macro, namely, how widely it will be understood. We will get to the nub of the problem, slowly but surely, by looking at global macros first.

3 Global macros—and their limitations

Suppose you are working with Stata's auto data,

```
. sysuse auto
(1978 Automobile Data)
```

and you want to think of those variables that measure various size characteristics as one group. You can put their names into a global macro with

```
. global size "headroom trunk weight length displacement"
```


The text within `" "` has been copied as contents to the global macro just created with the name `size`. That allows you to type `$size` anywhere in your code knowing that Stata will use the definition of the macro to replace the macro name, `size`, with its contents, a string including various variable names. So you might be interested in a regression predicting car price as a response, contained in the variable `price`. Typing

```
. regress price $size
  (output omitted)
```

would save you typing out those variable names again or even selecting them individually otherwise.

The key here: When Stata sees that command, the reference to the global macro `size` will be evaluated, meaning that the macro name will be replaced by its contents so that the rest of Stata will see

```
. regress price headroom trunk weight length displacement
  (output omitted)
```

Note the syntax detail: the dollar sign `$` flags that a global macro name follows.

Here is another example. Suppose we want to obtain and hold the median price for some purpose, say, to split prices into those above and below the median or to use it in preparing a graph. For that, we could do

```
. summarize price, detail
  (output omitted)
. global median_price = r(p50)
. display "$median_price"
5006.5
```

If you are thinking that `5006.5` looks like a number, you are naturally right, but that is not a problem or an exception. The definition that macros hold text still holds: text being entirely or partially composed of numeric characters is fine. The rest of Stata will make its own judgments according to what it sees once the macro contents have been substituted.

A little analogy that may help is to think that a macro is a kind of bag or box in which text is stored, just as you might put a pen or some other possession in a bag or box for safekeeping. Once you have taken the text out of the bag or box, its ever having been inside has no implications for how it is used.

Where might a global macro be used? Within one or more of the following:

1. an interactive session in which commands are typed in the Command window,
2. a do-file you run,
3. code you run from a Do-file Editor, or
4. a program you write or use.

This may sound good, but it is not as good as it may sound. The sales pitch is that you can make one convenient definition of a global macro, and it will be understood everywhere you use that global macro. But that universal visibility might go horribly wrong. Commands can call other commands, do-files can call commands, and do-files can call other do-files. And yet other things can happen in code. In any of these ways, if you define a global macro, you unwittingly may undermine some other code that uses the same global macro name. Stata does not know or care who wrote that code, or when, and does not discriminate in any way. If you ask for global scope, then global scope is what you get.

You may think of work-arounds. One is just to use macro names so bizarre that in practice no one else can possibly have used them before, including yourself in some previous state of existence. But I do not want to have to use bizarre names. I guess that you really do not want that obligation either. Concise but evocative names are always valuable, and you do not want to have to ensure that a name differs from any already in use.

4 Local macros

So, to cut to the chase, the answer is to use local macros. Local macros have local scope (only). Shortly, we will spell out precisely what that means. But the aim is to be always safe, and never sorry. You are completely in charge of how that local macro is used. You are guaranteed that fears and worries about whether you used that name somewhere else—or more crucially whether someone else did in his or her code that you are using—are utterly groundless.

Otherwise, local macros are just like global macros. They contain text. That text may include numeric characters. What is crucial is what the rest of Stata does when the contents of local macros are revealed by evaluation.

So, if I wrote interactively

```
local size "headroom trunk weight length displacement"
```

then now I can refer to that local macro interactively in the same session, just as I could refer to a global macro. I could issue a regression command,

```
regress price `size´
```

or any other appropriate command,

```
summarize `size´
pca `size´
```

I can also use a definition given in a do-file later in that do-file.

There is always a downside. Here is where being "local" bites. Local means local scope, so in particular:

- A local macro defined in an interactive session is not visible in a do-file or to the code you are running from a Do-file Editor (which may he held in some kind of temporary file). And all of those limits apply in reverse. More generally, each of those is a separate place and a local macro defined in one is not visible in any of the others.
- Perhaps surprisingly: If you choose to run just chunks of code (which may be as short as individual lines of code) selected from a larger section in a Do-file Editor, "local" now means within the same chunk of code. It is not enough that a local macro was defined earlier in the window. The definition must be included in the chunk being run. When this bites, caution on the part of the user is not being rewarded. You may be feeling your way through some code, perhaps checking what happens at each step in a long process, or some of your code may have been revised and part of the analysis is being repeated in slightly different form. Style of working, however, has no implications. The question is only whether the (last) macro definition, the `local` command defining the macro, is visible within the same code chunk.

I close with two further comments.

What happens when the definition of a local macro is not visible?

It is as if that local macro did not exist. Hence, references to that local macro are replaced with empty strings. Otherwise put, the reference is just blanked out. The consequence is not necessarily illegal code. The consequence is unlikely, however, to be what you want. Suppose, for example, that the definition of local macro `size` were not visible, because it is not within the chunk of code being run. In that circumstance,

```
regress price `size´
```

is just interpreted as

```
regress price
```

That is legal: the instruction is to fit a regression to `price` alone. The result is just a regression fit in which the mean of price will be returned as intercept. No predictors will appear in the model because none were supplied. As another example,

```
summarize `size´
```

is just interpreted as

```
summarize
```

if the definition of the local macro is not visible. Again, that is legal: it means `summarize` all variables, which will work so long as there are some variables to summarize.

An exercise if you want one: what would happen if Stata saw only the bare command `pca`?

Referring to a local macro that does not exist is not itself a syntax error. The consequence may be, as above, something legal, which is not what you want. Or it may be something that is illegal for different reasons. Such subtleties can mean that a bug is hard to find, but being aware of this pitfall is a good start.

Why is the division between local and global so sharp? Are there not intermediate cases?

Indeed, sometimes a Stata user wants a combination of local and global approaches for good reasons, typically that a bundle of definitions makes sense for all the code files for a particular project. The `include` command is designed for these situations. For more detail, see [P] **include** or Herrin (2009).

References

Herrin, J. 2009. Stata tip 77: (Re)using macros in multiple do-files. *Stata Journal* 9: 497–498. https://doi.org/10.1177/1536867X0900900310.

Kernighan, B. W., and R. Pike. 1984. *The UNIX Programming Environment*. Englewood Cliffs, NJ: Prentice Hall.

Kernighan, B. W., and D. M. Ritchie. 1988. *The C Programming Language*. 2nd ed. Englewood Cliffs, NJ: Prentice Hall.