The Stata Journal (2009)
9, Number 3, pp. 497–498

Stata tip 77: (Re)using macros in multiple do-files

Jeph Herrin
Yale School of Medicine
Yale University
New Haven, CT
jeph.herrin@yale.edu

Local and global macros provide an extremely useful way to define (for example) groups of values or variable names that can be (re)used in data management and analysis. For example, you may want to fit many different models using a given subset, or several different subsets, of independent variables. So you might define

```
. local indvars1 "var1 var2"
. local indvars2 "var1 var2 var3 var4"
```

Thereafter, subsequent models can be easily specified with reference to the same sets of variables:

```
. logit y `indvars1'
. regress y `indvars1'
. logit y `indvars2'
. regress y `indvars2'
```

However, users often like to define macros that can be referenced within several do-files. One way to have macros persist across do-files is to use globals, but globals should generally be reserved for macros that truly are wanted to be available in all contexts. An alternative is to borrow from other programming environments the use of "header" files, which contain preamble code that can be included at the top (or head) of each file of code to make common definitions.

For this purpose, Stata offers the `include` command, which is similar to `run` except that all local macro definitions are retained. See the manual entry [P] **include** for complete details. In the above example, we could have a file called `locals.do`:

begin `locals.do`

```
local indvars1 "var1 var2"
local indvars2 "var1 var2 var3 var4"
```

end `locals.do`

Then, in any file in which we would like to use these local macros, we can simply type

```
include locals.do
```

and thereafter refer to `indvars1`, `indvars2`, etc. If we subsequently want to modify the independent variables in our lists, we need only edit the `locals.do` file. The changes are automatically carried into other do-files that `include` that `locals.do` file.

This very useful feature can also be used to define directory paths and other programming data that we want to be available in local macros to all our do-files. For instance, a useful definition is

```
local today=string(date("`c(current_date)'","DMY"),"%tdCCYY.NN.DD")
```

which creates a string containing the current date in a lexically ordered format. Including this in a header file, and thence in all do-files, gives a standard way of adding a sortable date suffix to all saved files. You may `include` more than one file, as well as nesting `include`s, so that each project can `include` a `project.do` file that not only defines project-specific macros but also `include`s a `master.do`, which defines more general macros.

In summary, the use of `include` to call a file containing macro definitions allows one instance of those local (or global) macros to be made easily available to multiple do-files.