A do-file stops executing when the end of the file is reached, an exit is executed, or an error

(nonzero return code) occurs. **If an error occurs**, *the remaining commands in the do-file are not*

*executed.*

**If you press Break while executing a do-file**, *Stata responds as though an error has occurred,*

*stopping the do-file*. This happens because the return code is nonzero; see [U] 8 Error messages and

return codes for an explanation of return codes.

Example 4

Here is what happens when we execute a do-file and then press Break:

. do myjob2

. version 17.0

. use census

(Census data)

. tabulate region

Census

region Freq. Percent Cum.

Break

r(1);

end of do-file

Break

r(1);

.

When we pressed Break, Stata responded by typing Break and showed a return code of 1. Stata

seemingly repeated itself, typing first “end of do-file”, and then Break and the return code of 1

again. Do not worry about the repeated messages. The first message indicates that Stata was stopping

the tabulate because you pressed Break, and the second message indicates that Stata is stopping

the do-file for the same reason.

Example 5

Let’s try our example again, but this time, let’s introduce an error. We change the file myjob2.do

to read

begin myjob2.do

version 17.0

use censas

tabulate region

summarize marriage\_rate divorce\_rate median\_age if state!="Nevada"

end myjob2.do

To introduce a subtle typographical error, we typed use censas when we meant use census5. We

assume that there is no file called censas.dta, so now we have an error. Here is what happens

when you instruct Stata to do the file:

[ U ] 16 Do-files 171

. do myjob2

. version 17.0

. use censas

file censas.dta not found

r(601);

end of do-file

r(601);

.

When Stata was told to use censas, it responded with “file censas.dta not found” and a return code

of 601. Stata then typed “end of do-file” and repeated the return code of 601. The repeated message

occurred for the same reason it did when we pressed Break in the previous example. The use resulted

in a return code of 601, so the do-file itself resulted in the same return code. The important thing to

understand is that Stata stopped executing the file because there was an error.

Technical note

*We can tell Stata to continue executing the file* **even if there are errors by typing do filename,**

**nostop**. Here is the result:

. do myjob2, nostop

. version 17.0

. use censas

file censas.dta not found

r(601);

. tabulate region

no variables defined

r(111);

summarize marriage\_rate divorce\_rate median\_age if state!="Nevada"

no variables defined

r(111);

end of do-file

.

None of the commands worked because the do-file’s first command failed. That is why Stata

ordinarily stops. However, **if our file had contained anything that could work**, *it would have worked*.

In general, we do not recommend coding in this manner, as unintended consequences can result when

errors do not stop execution.

16.1.5 Logging the output of do-files

You log the output of do-files just as you would an interactive session; see [U] 15 Saving and

printing output—log files.

172 [ U ] 16 Do-files

Many users include the commands to start and stop the logging in the do-file itself:

begin myjob3.do

version 17.0

log using myjob3, replace

\* a sample analysis job

use census

tabulate region // obtain summary statistics

summarize marriage\_rate divorce\_rate median\_age if state!="Nevada"

log close

end myjob3.do

We chose to open with log using myjob3, replace, the important part being the replace option.

Had we omitted the option, we could not easily rerun our do-file. **If myjob3.smcl had already existed**

**and log was not told that it is okay to replace the file**, *the do-file would have stopped and instead*

*reported that file myjob3.smcl already exists*. We could get around that, of course, by erasing the

log file before running the do-file.

16.1.6 Preventing –more– conditions

Stata has more turned off by default; see [U] 7 –more– conditions.

If you have set more on for interactive use, Stata’s feature of pausing every time the screen is

full will probably be an irritation when you are running a do-file and logging the output.

The way around this is to include the line set more off in your do-file, which prevents Stata from

issuing more . The previous set more setting will automatically be restored when the do-file is

finished.

16.2 Calling other do-files

Do-files may call other do-files. Say that you wrote makedata.do, which infiles your data,

generates a few variables, and saves step1.dta. Say that you wrote anlstep1.do, which performed

a little analysis on step1.dta. You could then create a third do-file,

begin master.do

version 17.0

do makedata

do anlstep1

end master.do

and so in effect combine the two do-files.

Do-files may call other do-files, which, in turn, call other do-files, and so on. Stata allows do-files

to be nested 64 deep.

Be not confused: master.do above could call 1,000 do-files one after the other, and still the level

of nesting would be only two.

[ U ] 16 Do-files 173

16.3 Creating and running do-files

16.3.1 Creating and running do-files for Windows

1. You can execute do-files by typing do followed by the filename, as we did above.

2. You can execute do-files by selecting File > Do....

3. You can use the Do-file Editor to compose, save, and execute do-files; see [GSW] 13 Using

the Do-file Editor—automating Stata. To use the Do-file Editor, click on the Do-file Editor

button, or type doedit in the Command window. Stata also has a Project Manager for managing

collections of do-files and other files. See [P] Project Manager.

4. You can double-click on the icon for the do-file to launch Stata and open the do-file in the

Do-file Editor.

5. You can run the do-file in batch mode. See [GSW] B.5 Stata batch mode for details, but the

short explanation is that you open a Window command window and type

C:\data> "C:\Program Files\Stata17\Stata" /s do myjob

or

C:\data> "C:\Program Files\Stata17\Stata" /b do myjob

to run in batch mode, assuming that you have installed Stata in the folder C:\Program

Files\Stata17. /b and /s determine the kind of log produced, but put that aside for a

second. When you start Stata in these ways, Stata will run in the background. When the do-file

completes, the Stata icon on the taskbar will flash. You can then click on it to close Stata. If

you want to stop the do-file before it completes, click on the Stata icon on the taskbar, and

*Stata will ask you* **if you want to cancel the job**. **If you want Stata to exit when the do-file is**

**complete rather than flashing on the taskbar**, also *specify /e on the command line*.

To log the output, you can start the log before executing the do-file or you can include the log using

and log close in your do-file.

When you run Stata in these ways, Stata takes the following actions:

a. Stata automatically opens a log. **If you specified /s**, *Stata will open a SMCL log*; **if you**

**specified /b**, *Stata will open a plain text log*. **If your do-file is named xyz.do**, *the log*

*will be called xyz.smcl (/s) or xyz.log (/b) in the same directory*.

b. **If your do-file explicitly opens another log**, *Stata will save two copies of the output*.

c. Stata ignores more conditions and anything else that would cause the do-file to stop

were it running interactively.

16.3.2 Creating and running do-files for Mac

1. You can execute do-files by typing do followed by the filename, as we did above.

2. You can execute do-files by selecting File > Do....

3. You can use the Do-file Editor to compose, save, and execute do-files; see [GSM] 13 Using the

Do-file Editor—automating Stata. Click on the Do-file Editor button, or type doedit in the

Command window. Stata also has a Project Manager for managing collections of do-files and

other files. See [P] Project Manager.

4. You can double-click on the icon for the do-file to open the do-file in the Do-file Editor.

174 [ U ] 16 Do-files

5. *Double-clicking on the icon for a do-file named Stata do will launch Stata* **if it is not already**

**running and set the current working directory to the location of the do-file**.

6. You can run the do-file in batch mode. See [GSM] B.3 Stata batch mode for details, but the

short explanation is that you open a Terminal window and type

% /Applications/Stata/Stata.app/Contents/MacOS/Stata -s do myjob

or

% /Applications/Stata/Stata.app/Contents/MacOS/Stata -b do myjob

to run in batch mode, assuming that you have installed Stata/BE in the folder

/Applications/Stata. -b and -s determine the kind of log produced, but put that aside for a

second. When you start Stata in these ways, Stata will run in the background. When the do-file

completes, the Stata icon on the Dock will bounce until you put Stata into the foreground. You

can then exit Stata. **If you want to stop the do-file before it completes**, *right-click on the Stata*

*icon on the Dock, and select Quit.*

To log the output, you can start the log before executing the do-file or you can include the log using

and log close in your do-file.

When you run Stata in these ways, Stata takes the following actions:

a. Stata automatically opens a log. **If you specified -s**, *Stata will open a SMCL log*; **if you**

**specified -b,** *Stata will open a plain text log*. **If your do-file is named xyz.do**, *the log*

*will be called xyz.smcl (-s) or xyz.log (-b) in the same directory*.

b. **If your do-file explicitly opens another log**, *Stata will save two copies of the output*.

c. Stata ignores more conditions and anything else that would cause the do-file to stop

were it running interactively.

16.3.3 Creating and running do-files for Unix

1. You can execute do-files by typing do followed by the filename, as we did above.

2. You can execute do-files by selecting File > Do....

3. You can use the Do-file Editor to compose, save, and execute do-files; see [GSU] 13 Using the

Do-file Editor—automating Stata. Click on the Do-file Editor button, or type doedit in the

Command window. Stata also has a Project Manager for managing collections of do-files and

other files. See [P] Project Manager.

4. At the Unix prompt, you can type

$ xstata do filename

or

$ stata do filename

to launch Stata and run the do-file. **When the do-file completes**, *Stata will prompt you for the*

*next command* just as **if you had started Stata the normal way**. **If you want Stata to exit instead**,

include exit, *STATA clear as the last line of your do-file*.

To log the output, you can start the log before executing the do-file or you can include the log using

and log close in your do-file.

[ U ] 16 Do-files 175

5. At the Unix prompt, you can type

$ stata -s do filename &

or

$ stata -b do filename &

to run the do-file in the background. The above two examples both involve the use of stata, not

xstata. *Type stata*, **even if you usually use the GUI version of Stata**, xstata. The examples

differ only in that one specifies the -s option and the other, the -b option, which determines

the kind of log that will be produced. In the above examples, Stata takes the following actions:

a. Stata automatically opens a log. **If you specified -s**, *Stata will open a SMCL log*; **if you**

**specified -b**, *Stata will open a plain text log*. **If your do-file is named xyz.do**, *the log*

*will be called xyz.smcl (-s) or xyz.log (-b) in the current directory (the directory from*

*which you issued the stata command).*

b. **If your do-file explicitly opens another log**, *Stata will save two copies of the output*.

c. Stata ignores more conditions and anything else that would cause the do-file to stop

were it running interactively.

To reiterate: one way to run a do-file in the background and obtain a text log is by typing

$ stata -b do myfile &

Another way uses standard redirection:

$ stata < myfile.do > myfile.log &

The first way is slightly more efficient. Either way, *Stata knows it is in the background and ignores*

*more conditions and anything else that would cause the do-file to stop* **if it were running**

**interactively**. However**, if your do-file contains either the #delimit command or the comment**

**characters (/\* at the end of one line and \*/ at the beginning of the next**), *the second method will*

*not work*. We recommend that you use the first method: stata -b do myfile &.

The choice between stata -b do myfile & and stata -s do myfile & is more personal. We

prefer obtaining SMCL logs (-s) because they look better when printed, and, in any case, they can

always be converted to text format with translate; see [R] translate.

16.4 Programming with do-files

This is an advanced topic, and we are going to refer to concepts not yet explained; see [U] 18 Programming Stata for more information.

16.4.1 Argument passing

Do-files accept arguments, just as Stata programs do; this is described in [U] 18 Programming

Stata and [U] 18.4 Program arguments. In fact, the logic Stata follows when invoking a do-file

is the same as when invoking a program: the local macros are stored, and new ones are defined.

Arguments are stored in the local macros ‘1’, ‘2’, and so on. When the do-file completes, the

previous definitions are restored, just as with programs.

Thus, **if you wanted your do-file to**

**1. use a dataset of your choosing,**

**2. tabulate a variable named region, and**

**3. summarize variables marriage rate and divorce rate**,

176 [ U ] 16 Do-files

*you could write the do-file*

*begin myxmpl.do*

*use ‘1’*

*tabulate region*

*summarize marriage\_rate divorce\_rate*

*end myxmpl.do*

*and you could run this do-file by typing, for instance,*

*. do myxmpl census*

*(output omitted* )

The first command—use ‘1’—would be interpreted as use census5 because census5 was the

first argument you typed after do myxmpl.

An even better version of the do-file would read

begin myxmpl.do

args dsname

use ‘dsname’

tabulate region

summarize marriage\_rate divorce\_rate

end myxmpl.do

The args command merely assigns a better name to the argument passed. args dsname does not

verify that what we type following do myxmpl is a filename—*we would have to use the syntax*

*command* **if we wanted to do that**—but substituting ‘dsname’ for ‘1’ does make the code more

readable.

**If our program were to receive two arguments**, *we could refer to them as ‘1’ and ‘2’, or we could*

*put an ‘args dsname other’ at the top of our do-file and then refer to ‘dsname’ and ‘other’*.

To learn more about argument passing, see [U] 18.4 Program arguments. Baum (2016) provides

many examples and tips related to do-files.

16.4.2 Suppressing output

There is an alternative to typing do filename; it is run filename. run works in the same way as

do, except that neither the instructions in the file nor any of the output caused by those instructions

is shown on the screen or in the log file.

For instance, with the above myxmpl.do, typing run myxmpl census5 results in

. run myxmpl census

.

All the instructions were executed, but none of the output was shown.

This is not useful here, **but if the do-file contained only the definitions of Stata programs**— *see*

*[U] 18 Programming Stata—and you merely wanted to load the programs without seeing the code,*

*run would be useful.*

[ U ] 16 Do-files 177

16.5 References

Baum, C. F. 2016. An Introduction to Stata Programming. 2nd ed. College Station, TX: Stata Press.

Long, J. S. 2009. The Workflow of Data Analysis Using Stata. College Station, TX: Stata Press.

Wiggins, V. L. 2018. How to automate common tasks. The Stata Blog: Not Elsewhere Classified.

https://blog.stata.com/2018/10/09/how-to-automate-common-tasks/.

17 Ado-files

Contents

17.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178

17.2 What is an ado-file? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178

17.3 How can I tell if a command is built in or an ado-file? . . . . . . . . . . . . . . . . . . . . . . . . 179

17.4 How can I look at an ado-file? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 179

17.5 Where does Stata look for ado-files? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180

17.5.1 Where is the official ado-directory? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180

17.5.2 Where is my personal ado-directory? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 181

17.6 How do I install an addition? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 181

17.7 How do I add my own ado-files? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182

17.8 How do I install official updates? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182

17.9 How do I install updates to community-contributed additions? . . . . . . . . . . . . . . . . . . . 182

17.10 References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182

17.1 Description

Stata is programmable, and **even if you never write a Stata program**, *Stata’s programmability is*

*still important*. Many of Stata’s features are implemented as Stata programs, and new features are

implemented every day, both by StataCorp and by others.

1. You can obtain additions from the Stata Journal. You subscribe to the printed journal, but the

software additions are available free over the Internet.

2. You can obtain additions from the Stata forum, Statalist, where an active group of users advise

each other on how to use Stata, and often, in the process, trade programs. Visit the Statalist

website, https://www.statalist.org, for instructions on how to participate.

3. The Boston College Statistical Software Components (SSC) Archive is a distributed database

making available a large and constantly growing number of Stata programs. You can browse

and search the archive, and you can find links to the archive from https://www.stata.com.

Importantly, Stata knows how to access the archive and other places, as well. You can search

for additions by using Stata’s search, net command; see [R] search. You can immediately

install materials you find with search, net by using the hyperlinks that will be displayed by

search in the Results window or by using the net command. A specialized command, ssc,

has several options available to help you find and install the community-contributed commands

that are available from this site; see [R] ssc.

4. You can write your own additions to Stata.

This chapter is written for people who want to use ado-files. All users should read it **If you later**

**decide you want to write ado-files**, *see [U] 18.11 Ado-files*.

17.2 What is an ado-file?

An ado-file defines a Stata command, but not all Stata commands are defined by ado-files.

When you type summarize to obtain summary statistics, you are using a command built into

Stata.

178

[ U ] 17 Ado-files 179

When you type ci to obtain confidence intervals, you are running an ado-file. The results of using

a built-in command or an ado-file are indistinguishable.

An ado-file is a text file that contains a Stata program. When you type a command that Stata does

not know, it looks in certain places for an ado-file of that name. **If Stata finds it**, *Stata loads and*

*executes it*, so it appears to you as if the ado-command is just another command built into Stata.

We just told you that Stata’s ci command is implemented as an ado-file. That means that,

somewhere, there is a file named ci.ado.

Ado-files usually come with help files. When you type help ci (or select Help > Stata command...,

and type ci), Stata looks for ci.sthlp, just as it looks for ci.ado when you use the ci command.

A help file is also a text file that tells Stata’s help system what to display.

17.3 How can I tell if a command is built in or an ado-file?

You can use the which command to determine whether a file is built in or implemented as an

ado-file. For instance, logistic is an ado-file, and here is what happens when you type which

logistic:

. which logistic

C:\Program Files\Stata17\ado\base\l\logistic.ado

\*! version 3.5.4 28feb2017

summarize is a built-in command:

. which summarize

built-in command: summarize

17.4 How can I look at an ado-file?

When you type which followed by an ado-command, Stata reports where the file is stored:

. which logistic

C:\Program Files\Stata17\ado\base\l\logistic.ado

\*! version 3.5.4 28feb2017

Ado-files are just text files containing the Stata program. You can view them in Stata’s Viewer window

(or even look at them in your editor or word processor) by typing

. type "C:\Program Files\Stata17\ado\base\l\logistic.ado"

\*! version 3.5.4 28feb2017

program define logistic, eclass prop(or svyb svyj svyr swml mi bayes) ///

byable(onecall)

version 6.0, missing

(output omitted )

end

or

. viewsource logistic.ado

(output omitted )

180 [ U ] 17 Ado-files

The type command displays the contents of a file. The viewsource command searches for a file

along the ado-directories and displays the file in the Viewer. *You can also look at the corresponding*

*help file in raw form* **if you wish**. **If there is a help file**, *it is stored in the same place as the ado-file*:

. type "C:\Program Files\Stata17\ado\base\l\logistic.sthlp", asis

{smcl}

{\* \*! version 1.4.5 18mar2021}{...}

{viewerdialog logistic "dialog logistic"}{...}

(output omitted )

or

. viewsource logistic.sthlp

(output omitted )

17.5 Where does Stata look for ado-files?

Stata looks for ado-files in seven places, which can be categorized in three ways:

I. The official ado-directory:

1. (BASE), the official directory containing the ado-files shipped with your version of Stata

and any updated ado-files that have been made available since then

II. Your personal ado-directories:

2. (SITE), the directory for ado-files your site might have installed

3. (PLUS), the directory for ado-files you personally might have installed

4. (PERSONAL), the directory for ado-files you might have written

5. (OLDPLACE), the directory where Stata users used to save their personally written ado-files

III. The current directory:

6. (.), the ado-files you have written just this instant or for just this project

The location of these directories varies from computer to computer, but Stata’s sysdir command

will tell you where they are on your computer:

. sysdir

STATA: C:\Program Files\Stata17\

BASE: C:\Program Files\Stata17\ado\base\

SITE: C:\Program Files\Stata17\ado\site\

PLUS: C:\ado\plus\

PERSONAL: C:\ado\personal\

OLDPLACE: C:\ado\

17.5.1 Where is the official ado-directory?

This is the directory listed as BASE by sysdir:

. sysdir

STATA: C:\Program Files\Stata17\

BASE: C:\Program Files\Stata17\ado\base\

SITE: C:\Program Files\Stata17\ado\site\

PLUS: C:\ado\plus\

PERSONAL: C:\ado\personal\

OLDPLACE: C:\ado\

1. BASE contains the ado-files we originally shipped to you and any updates you might have

installed since then. You can install updates by using the update command or by selecting

Help > Check for updates; see [U] 17.8 How do I install official updates?.

[ U ] 17 Ado-files 181

17.5.2 Where is my personal ado-directory?

These are the directories listed as PERSONAL, PLUS, SITE, and OLDPLACE by sysdir:

. sysdir

STATA: C:\Program Files\Stata17\

BASE: C:\Program Files\Stata17\ado\base\

SITE: C:\Program Files\Stata17\ado\site\

PLUS: C:\ado\plus\

PERSONAL: C:\ado\personal\

OLDPLACE: C:\ado\

1. PERSONAL is for ado-files you have written. Store your private ado-files here; see [U] 17.7 How

do I add my own ado-files?.

2. PLUS is for ado-files you personally installed but did not write. Such ado-files are usually

obtained from the SJ or the SSC Archive, but they are sometimes found in other places, too.

You find and install such files by using Stata’s net command, or you can select Help > SJ

and community-contributed features; see [U] 17.6 How do I install an addition?.

3. SITE is really the opposite of a personal ado-directory—it is a public directory corresponding

to PLUS. **If you are on a networked computer**, *the site administrator can install ado-files here,*

*and all Stata users will then be able to use them* **just as if they all found and installed them**

**in their PLUS directory for themselves**. Site administrators find and install the ado-files just as

you would, using Stata’s net command, but they specify an option when they install something

that tells Stata to write the files into SITE rather than PLUS; see [R] net.

4. OLDPLACE is for old-time Stata users. Prior to Stata 6, all “personal” ado-files, whether personally

written or just personally installed, were written in the same directory—OLDPLACE. So that the

old-time Stata users do not have to go back and rearrange what they have already done, Stata

still looks in OLDPLACE.

17.6 How do I install an addition?

Additions come in four types:

1. Community-contributed additions, which you might find in the SJ, etc.

2. Updates to community-contributed additions

See [U] 17.9 How do I install updates to community-contributed additions?.

3. Ado-files you have written

See [U] 17.7 How do I add my own ado-files **If you have an ado-file obtained from**

**the Stata forum or a friend**, *treat it as belonging to this case*.

4. Official updates provided by StataCorp

See [U] 17.8 How do I install official updates?.

Community-contributed additions you might find in the Stata Journal (SJ), etc., are obtained over

the Internet. To access them on the Internet,

1. select Help > SJ and community-contributed features, and click on one of the links

or

2. type net from https://www.stata.com.

What to do next will be obvious, but, in case it is not, see [GS] 19 Updating and extending

Stata—Internet functionality (GSM, GSU, or GSW). Also see [U] 29 Using the Internet to keep up

to date, [R] net, and [R] ado update.

182 [ U ] 17 Ado-files

17.7 How do I add my own ado-files?

You write a Stata program (see [U] 18 Programming Stata), store it in a file ending in .ado,

perhaps write a help file, and copy everything to the directory sysdir lists as PERSONAL:

. sysdir

STATA: C:\Program Files\Stata17\

BASE: C:\Program Files\Stata17\ado\base\

SITE: C:\Program Files\Stata17\ado\site\

PLUS: C:\ado\plus\

PERSONAL: C:\ado\personal\

OLDPLACE: C:\ado\

Here we would copy the files to C:\ado\personal.

While you are writing your ado-file, it is sometimes convenient to store the pieces in the current

directory. Do that **if you wish**; *you can move them to your personal ado-directory when the program*

*is debugged*.

17.8 How do I install official updates?

Updates are available over the Internet:

1. select Help > Check for updates, and then click on https://www.stata.com

or

2. type update query.

What to do next should be obvious, but in case it is not, see [GS] 19 Updating and extending

Stata—Internet functionality (GSM, GSU, or GSW). Also see [U] 29 Using the Internet to keep up

to date and [R] net.

The official updates include bug fixes and new features but do not change the syntax of an existing

command or change the way Stata works.

Once you have installed the updates, you can enter Stata and type help whatsnew (or select

Help > What’s new?) to learn about what has changed.

17.9 How do I install updates to community-contributed additions?

If you have previously installed community-contributed additions, you can check for updates to

them by typing adoupdate. **If updates are available**, *you can install them by typing ado update,*

*update*. See [R] ado update.

17.10 References

Cox, N. J. 2006. Stata tip 30: May the source be with you. Stata Journal 6: 149–150.

Wiggins, V. L. 2018. How to automate common tasks. The Stata Blog: Not Elsewhere Classified.

https://blog.stata.com/2018/10/09/how-to-automate-common-tasks/.

18 Programming Stata

Contents

18.1 Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 184

18.2 Relationship between a program and a do-file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 185

18.3 Macros . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 187

18.3.1 Local macros . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 188

18.3.2 Global macros . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189

18.3.3 The difference between local and global macros . . . . . . . . . . . . . . . . . . . . . . 189

18.3.4 Macros and expressions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 190

18.3.5 Double quotes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 191

18.3.6 Macro functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 193

18.3.7 Macro increment and decrement functions . . . . . . . . . . . . . . . . . . . . . . . . . . . 194

18.3.8 Macro expressions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 195

18.3.9 Advanced local macro manipulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 196

18.3.10 Advanced global macro manipulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 197

18.3.11 Constructing Windows filenames by using macros . . . . . . . . . . . . . . . . . . . . 198

18.3.12 Accessing system values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198

18.3.13 Referring to characteristics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 199

18.4 Program arguments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 199

18.4.1 Named positional arguments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201

18.4.2 Incrementing through positional arguments . . . . . . . . . . . . . . . . . . . . . . . . . . . 203

18.4.3 Using macro shift . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 204

18.4.4 Parsing standard Stata syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 205

18.4.5 Parsing immediate commands . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 207

18.4.6 Parsing nonstandard syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 207

18.5 Scalars and matrices . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 208

18.6 Temporarily destroying the data in memory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209

18.7 Temporary objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209

18.7.1 Temporary variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209

18.7.2 Temporary scalars and matrices . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210

18.7.3 Temporary files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210

18.7.4 Temporary frames . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 211

18.8 Accessing results calculated by other programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 211

18.9 Accessing results calculated by estimation commands . . . . . . . . . . . . . . . . . . . . . . . . . . 214

18.10 Storing results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 215

18.10.1 Storing results in r() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 216

18.10.2 Storing results in e() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 217

18.10.3 Storing results in s() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 220

18.11 Ado-files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 220

18.11.1 Version . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222

18.11.2 Comments and long lines in ado-files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222

18.11.3 Debugging ado-files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 223

18.11.4 Local subroutines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 223

18.11.5 Development of a sample ado-command . . . . . . . . . . . . . . . . . . . . . . . . . . . . 224

18.11.6 Writing system help . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 230

18.11.7 Programming dialog boxes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 233

18.12 Tools for interacting with programs outside Stata and with other languages . . . . . . . . 234

183

184 [ U ] 18 Programming Stata

18.13 A compendium of useful commands for programmers . . . . . . . . . . . . . . . . . . . . . . . . . 234

18.14 References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 234

Stata programming is an advanced topic. Some Stata users live productive lives without ever

programming Stata. After all, you do not need to know how to program Stata to import data, create

new variables, and fit models. On the other hand, *programming Stata is not difficult*—at least **if the**

**problem is not difficult—and Stata’s programmability is one of its best features**. The real power of

Stata is not revealed until you program it.

Stata has two programming languages. One, known informally as “ado”, is the focus of this chapter.

It is based on Stata’s commands, and you can write scripts and programs to automate reproducible

analyses and to add new features to Stata.

The other language, Mata, is a byte-compiled language with syntax similar to C/C++, but with

extensive matrix capabilities. The two languages can interact with each other. You can call Mata

functions from ado-programs, and you can call ado-programs from Mata functions. You can learn all

about Mata in the Mata Reference Manual.

Stata also has a Project Manager to help you manage large collections of Stata scripts, programs,

and other files. See [P] Project Manager.

**If you are uncertain whether to read this chapter**, *we recommend that you start reading and then*

*bail out when it gets too arcane for you*. *You will learn things about Stata that you may find useful*

**even if you never write a Stata program**.

**If you want even more**, *we offer courses over the Internet on Stata programming*; see [U] 3.6.2 NetCourses. Baum (2016) provides a wealth of practical knowledge related to Stata programming.

18.1 Description

When you type a command that Stata does not recognize, Stata first looks in its memory for a

program of that name **If Stata finds it**, *Stata executes the program*.

There is no Stata command named hello,

. hello

command hello is unrecognized

r(199);

but there could be if you defined a program named hello, and after that, the following might happen

when you typed hello:

. hello

hi there

.

This would happen if, beforehand, you had typed

. program hello

1. display "hi there"

2. end

.

That is how programming works in Stata. A program is defined by

program progname

Stata commands

end

and it is executed by typing progname at Stata’s dot prompt.

[ U ] 18 Programming Stata 185

18.2 Relationship between a program and a do-file

Stata treats programs the same way it treats do-files. Below we will discuss passing arguments,

consuming results from Stata commands, and other topics, but everything we say applies equally to

do-files and programs.

Programs and do-files differ in the following ways:

1. You invoke a do-file by typing do filename. You invoke a program by simply typing the

program’s name.

2. Programs must be defined (loaded) before they are used, whereas all that is required to run a

do-file is that the file exist. There are ways to make programs load automatically, however, so

this difference is of little importance.

3. When you type do filename, Stata displays the commands it is executing and the results. When

you type progname, Stata shows only the results, not the display of the underlying commands.

This is an important difference in outlook: in a do-file, how it does something is as important

as what it does. In a program, the how is no longer important. You might think of a program

as a new feature of Stata.

Let’s now mention some of the similarities:

1. Arguments are passed to programs and do-files in the same way.

2. Programs and do-files both contain Stata commands. Any Stata command you put in a do-file

can be put in a program.

3. Programs may call other programs. Do-files may call other do-files. Programs may call do-files

(this rarely happens), and do-files may call programs (this often happens). Stata allows programs

(and do-files) to be nested up to 64 deep.

Now here is the interesting thing: programs are typically defined in do-files (or in a variant of do-files

called ado-files; we will get to that later).

You can define a program interactively, and that is useful for pedagogical purposes, but in real

applications, you will compose your program in a text editor and store its definition in a do-file.

You have already seen your first program:

program hello

display "hi there"

end

You could type those commands interactively, but **if the body of the program were more complicated**,

*that would be inconvenient*. So instead, suppose that you typed the commands into a do-file:

begin hello.do

program hello

display "hi there"

end

end hello.do

Now returning to Stata, you type

. do hello

. program hello

1. display "hi there"

2. end

.

end of do-file

186 [ U ] 18 Programming Stata

Do you see that typing do hello did nothing but load the program? Typing do hello is the same as

typing out the program’s definition because that is all the do-file contains. The do-file was executed,

but the statements in the do-file only defined the program hello; they did not execute it. Now that

the program is loaded, we can execute it interactively:

. hello

hi there

So, that is one way you could use do-files and programs together. **If you wanted to create new**

**commands for interactive use**, *you could*

*1. Write the command as a program . . . end in a do-file.*

*2. do the do-file before you use the new command.*

*3. Use the new command during the rest of the session*.

There are more convenient ways to do this that would automatically load the do-file, but put that

aside. The above method would work.

Another way we could use do-files and programs together is to put the definition of the program

and its execution together into a do-file:

begin hello.do

program hello

display "hi there"

end

hello

end hello.do

Here is what would happen if we executed this do-file:

. do hello

. program hello

1. display "hi there"

2. end

. hello

hi there

.

end of do-file

Do-files and programs are often used in such combinations. Why? Say that program hello is long

and complicated and you have a problem where you need to do it twice. That would be a good reason

to write a program. Moreover, you may wish to carry forth this procedure as a step of your analysis

and, being cautious, do not want to perform this analysis interactively. You never intended program

hello to be used interactively—it was just something you needed in the midst of a do-file— so you

defined the program and used it there.

Anyway, there are many variations on this theme, but few people actually sit in front of Stata and

interactively type program and then compose a program. They instead do that in front of their text

editor. They compose the program in a do-file and then execute the do-file.

There is one other (minor) thing to know: once a program is defined, Stata does not allow you to

redefine it:

. program hello

program hello already defined

r(110);

[ U ] 18 Programming Stata 187

Thus, in our most recent do-file that defines and executes hello, we could not rerun it in the same

Stata session:

. do hello

. program hello

program hello already defined

r(110);

end of do-file

r(110);

That problem is solved by typing program drop hello before redefining it. We could do that

interactively, or we could modify our do-file:

begin hello.do

program drop hello

program hello

display "hi there"

end

hello

end hello.do

There is a problem with this solution. We can now rerun our do-file, but the first time we tried to

run it in a Stata session, it would fail:

. do hello

. program drop hello

hello not found

r(111);

end of do-file

r(111);

The way around this conundrum is to modify the do-file:

begin hello.do

capture program drop hello

program hello

display "hi there"

end

hello

end hello.do

capture in front of a command makes Stata indifferent to whether the command works; see

[P] capture. In real do-files containing programs, you will often see capture program drop before

the program’s definition.

To learn about the program command itself, see [P] program. It manipulates programs. program

can define programs, drop programs, and show you a directory of programs that you have defined.

A program can contain any Stata command, but certain Stata commands are of special interest to

program writers; see the Programming heading in the subject table of contents in the Stata Index.

18.3 Macros

Before we can begin programming, we must discuss macros, which are the variables of Stata

programs.

A macro is a string of characters, called the macroname, that stands for another string of characters,

called the macro contents.

Macros can be local or global. We will start with local macros because they are the most commonly

used, but nothing really distinguishes one from the other at this stage.

188 [ U ] 18 Programming Stata

18.3.1 Local macros

Local macro names can be up to 31 (not 32) characters long.

One sets the contents of a local macro with the local command. In fact, we can do this interactively.

We will begin by experimenting with macros in this way to learn about them **If we type**

**. local shortcut myvar thisvar thatvar**

*then ‘shortcut’ is a synonym for myvar thisvar thatvar*. Note the single quotes around

shortcut. We said that sentence exactly the way we meant to because

**if you type ‘shortcut’**,

i.e., left-single-quote shortcut right-single-quote,

*Stata hears myvar thisvar thatvar*.

To access the contents of the macro, we use a left single quote (located at the upper left on most

keyboards), the macro name, and a right single quote (located under the " on the right side of most

keyboards).

The single quotes bracketing the macroname shortcut are called the macro-substitution characters.

shortcut means shortcut. ‘shortcut’ means myvar thisvar thatvar.

So, **if you typed**

**. list ‘shortcut’**

*the effect would be exactly as if you typed*

*. list myvar thisvar thatvar*

Macros can be used anywhere in Stata. For instance, if we also defined

. local cmd "list"

we could type

. ‘cmd’ ‘shortcut’

to mean list myvar thisvar thatvar.

For another example, consider the definitions

. local prefix "my"

. local suffix "var"

Then

. ‘cmd’ ‘prefix’‘suffix’

would mean list myvar.

[ U ] 18 Programming Stata 189

One other important note is on the way we use left and right single quotes within Stata, which

you will especially deal with when working with macros (see [U] 18.3 Macros). Single quotes (and

double quotes, for that matter) may look different on your keyboard, your monitor, and our printed

documentation, making it difficult to determine which key to press on your keyboard to replicate

what we have shown you.

For the left single quote, we use the grave accent, which occupies a key by itself on most computer

keyboards. On U.S. keyboards, the grave accent is located at the top left, next to the numeral 1. On

some non-U.S. keyboards, the grave accent is produced by a dead key. For example, pressing the

grave accent dead key followed by the letter a would produce a; to get the grave accent by itself, `

you would press the grave accent dead key followed by a space. This accent mark appears in our

printed documentation as ‘.

For the right single quote, we use the standard single quote, or apostrophe. On U.S. keyboards,

the single quote is located on the same key as the double quote, on the right side of the keyboard

next to the Enter key.

18.3.2 Global macros

Let’s put aside why Stata has two kinds of macros—local and global—and focus right now on

how global macros work.

Global macros can have names that are up to 32 (not 31) characters long. You set the contents of

a global macro by using the global rather than the local command:

. global shortcut "alpha beta"

You obtain the contents of a global macro by prefixing its name with a dollar sign: $shortcut is

equivalent to “alpha beta”.

In the previous section, we defined a local macro named shortcut, which is a different macro.

‘shortcut’ is still “myvar thisvar thatvar”.

Local and global macros may have the same names, but even **if they do**, *they are unrelated and*

*are still distinguishable*.

Global macros are just like local macros except that you set their contents with global rather

than local, and you substitute their contents by prefixing them with a $ rather than enclosing them

in ‘’.

18.3.3 The difference between local and global macros

The difference between local and global macros is that local macros are private and global macros

are public.

Say that you have written a program

program myprog

code using local macro alpha

end

The local macro alpha in myprog is private in that no other program can modify or even look at

alpha’s contents. To make this point absolutely clear, assume that your program looks like this:

190 [ U ] 18 Programming Stata

program myprog

code using local macro alpha

mysub

more code using local macro alpha

end

program mysub

code using local macro alpha

end

myprog calls mysub, and both programs use a local macro named alpha. Even so, the local macros

in each program are different. mysub’s alpha macro may contain one thing, but that has nothing to

do with what myprog’s alpha macro contains. Even when mysub begins execution, its alpha macro

is different from myprog’s. It is not that mysub’s inherits myprog’s alpha macro contents but is then

free to change it. It is that myprog’s alpha and mysub’s alpha are entirely different things.

When you write a program using local macros, you need not worry that some other program

has been written using local macros with the same names. Local macros are just that: local to your

program.

Global macros, on the other hand, are available to all programs. **If both myprog and mysub use**

**the global macro beta**, *they are using the same macro*. Whatever the contents of $beta are when

mysub is invoked, those are the contents when mysub begins execution, and, whatever the contents

of $beta are when mysub completes, those are the contents when myprog regains control.

18.3.4 Macros and expressions

From now on, we are going to use local and global macros according to whichever is convenient;

whatever is said about one applies to the other.

Consider the definitions

. local one 2+2

. local two = 2+2

(which we could just as well have illustrated using the global command). In any case, note the

equal sign in the second macro definition and the lack of the equal sign in the first. Formally, the

first should be

. local one "2+2"

but Stata does not mind if we omit the double quotes in the local (global) statement