16 Do-files

Contents

16.1 Description

Rather than typing commands at the keyboard, you can create a text file containing commands and instruct Stata to execute the commands stored in that file. Such files are called *do-files* because the command that causes them to be executed is `do`.

A do-file is a standard text file that is executed by Stata when you type `do` *filename*. You can use any text editor or the built-in Do-file Editor to create do-files; see **[GSW] 13 Using the Do-file Editor—automating Stata**. Using do-files rather than typing commands with the keyboard or using dialog boxes offers several advantages. By writing the steps you take to manage and analyze your data in the form of a do-file, you can reproduce your work later. Also, writing a do-file makes the inevitable debugging process much easier. If you decide to change one part of your analysis, changing the relevant commands in your do-file is much easier than having to start back at square one, as is often necessary when working interactively. In this chapter, we describe the mechanics of do-files. Long (2009) cogently argues that do-files should be used in all research projects and offers an abundance of time-tested advice in how to manage data and statistical analysis.

▷ Example 1

You can use do-files to create a batchlike environment in which you place all the commands you want to perform in a file and then instruct Stata to `do` that file. Assume that you use your text editor or word processor to create a file called `myjob.do` that contains these three lines:

begin `myjob.do`

```
use https://www.stata-press.com/data/r19/census5
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
```

end `myjob.do`

You then enter Stata and instruct Stata to do the file:

```
. do myjob

. use https://www.stata-press.com/data/r19/census5
(1980 Census data by state)

. tabulate region
```

Census region	Freq.	Percent	Cum.
NE	9	18.00	18.00
N Cntrl	12	24.00	42.00
South	16	32.00	74.00
West	13	26.00	100.00
Total	50	100.00	

```
. summarize marriage_rate divorce_rate median_age if state !="Nevada"
```

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	49	.0106791	.0021746	.0074654	.0172704
divorce_rate	49	.0054268	.0015104	.0029436	.008752
median_age	49	29.52653	1.708286	24.2	34.7

You typed only `do myjob` to produce this output. Because you did not specify the file extension, Stata assumed you meant `do myjob.do`; see **[U] 11.6 Filenaming conventions**.

◁

16.1.1 Version

We recommend that the first line in your do-file declare the Stata release you used when you wrote the do-file; `myjob.do` would read better as

begin myjob.do

```
version 19.5      // (or version 19 if you do not have StataNow)
use https://www.stata-press.com/data/r19/census5
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
```

end myjob.do

We admit that we do not always follow our own advice, as you will see many examples in this manual that do not include the `version` line. (In the above, the text that follows two forward slashes, `//`, is a comment; see **[U] 16.1.2 Comments and blank lines in do-files**.)

If you intend to keep the do-file, however, you should include this line because it ensures that your do-file will continue to work with future versions of Stata. Stata is under continual development, and sometimes things change in surprising ways.

For instance, in Stata 3.0, a new syntax for specifying the weights was introduced. If you had an old do-file written for Stata 2.1 that analyzed weighted data and did not have `version 2.1` at the top, you would find that today's Stata would flag some of the file's lines as syntax errors. If you had the `version 2.1` line, it would work just as it used to.

Skipping ahead to Stata 10, we introduced `xtset` and declared that, to use the `xt` commands, you must `xtset` your data first. Previously, you specified options on the end of each `xt` command that identified the group and, optionally, the time variables. Despite this change, if you include `version 9` or earlier at the top of your do-file, the `xt` commands will continue to work the old way.

For an overview of versioning and an up-to-date list of the issues that versioning does not address automatically, see `help version`.

When running an old do-file that includes a `version` statement, you need not worry about setting the version back after it has completed. Stata automatically restores the previous value of `version` when the do-file completes.

See **[U] 12.4.2.6 Advice for users of Stata 13 and earlier** for information about sharing your Stata 19 files with users of Stata 13 or earlier.

16.1.2 Comments and blank lines in do-files

You may freely include blank lines in your do-file. In the previous example, the do-file could just as well have read

```
begin myjob.do
version 19.5      // (or version 19 if you do not have StataNow)

use https://www.stata-press.com/data/r19/census5
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

There are four ways to include comments in a do-file.

1. Begin the line with a '`*`'; Stata ignores such lines. `*` cannot be used within Mata.
2. Place the comment in `/* */` delimiters.
3. Place the comment after two forward slashes, that is, `//`. Everything after the `//` to the end of the current line is considered a comment (unless the `//` is part of `https://...`). We used this type of comment on the first line of the do-file above.
4. Place the comment after three forward slashes, that is, `///`. Everything after the `///` to the end of the current line is considered a comment. However, when you use `///`, the next line joins with the current line. `///` lets you split long lines across multiple lines in the do-file.

Note that a whitespace character must separate any of the four comment methods from surrounding text.

❑ Technical note

The `/* */`, `//`, and `///` comment indicators can be used in do-files and ado-files only; you may not use them interactively. You can, however, use the '`*`' comment indicator interactively.

❑

`myjob.do` then might read

```
begin myjob.do
* a sample analysis job
version 19.5      // (or version 19 if you do not have StataNow)

use https://www.stata-press.com/data/r19/census5

/* obtain the summary statistics: */
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

or equivalently,

```
begin myjob.do
// a sample analysis job
version 19.5      // (or version 19 if you do not have StataNow)
use https://www.stata-press.com/data/r19/census5
// obtain the summary statistics:
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

The style of comment indicator you use is up to you. One advantage of the `/* */` method is that it can be put at the end of lines:

```
begin myjob.do
* a sample analysis job
version 19.5      // (or version 19 if you do not have StataNow)
use https://www.stata-press.com/data/r19/census5
tabulate region                   /* obtain summary statistics */
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

In fact, `/* */` can be put anywhere, even in the middle of a line:

```
begin myjob.do
* a sample analysis job
version 19.5      // (or version 19 if you do not have StataNow)
use /* confirm this is latest */ https://www.stata-press.com/data/r19/census5
tabulate region                   /* obtain summary statistics */
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

You can achieve the same results with the `//` and `///` methods:

```
begin myjob.do
// a sample analysis job
version 19.5      // (or version 19 if you do not have StataNow)
use https://www.stata-press.com/data/r19/census5
tabulate region                   // obtain summary statistics
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

or

```
begin myjob.do
// a sample analysis job
version 19.5      // (or version 19 if you do not have StataNow)
use /// confirm this is latest
https://www.stata-press.com/data/r19/census5
tabulate region                   // obtain summary statistics
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

16.1.3 Long lines in do-files

When you use Stata interactively, you press *Enter* to end a line and tell Stata to execute it. If you need to type a line that is wider than the screen, you simply do it, letting it wrap or scroll.

You can follow the same procedure in do-files—if your editor or word processor will let you—but you can do better. You can change the end-of-line delimiter to ';' by using `#delimit`, you can comment out the line break by using `/* */` comment delimiters, or you can use the `///` line-join indicator.

▷ Example 2

In the following fragment of a do-file, we temporarily change the end-of-line delimiter:

fragment of `example.do`

```
use mydata
#delimit ;
summarize weight price displ headroom rep78 length turn gear_ratio
          if substr(company,1,4)=="Ford" |
             substr(company,1,2)=="GM", detail ;
generate byte ford = substr(company,1,4)=="Ford" ;
#delimit cr
generate byte gm = substr(company,1,2)=="GM"
```

fragment of `example.do`

Once we change the line delimiter to semicolon, all lines, even short ones, must end in semicolons. Stata treats carriage returns as no different from blanks. We can change the delimiter back to carriage return by typing `#delimit cr`.

The `#delimit` command is allowed only in do-files—it is not allowed interactively. You need not remember to set the delimiter back to carriage return at the end of a do-file because Stata will reset it automatically.

◁

▷ Example 3

The other way around long lines is to comment out the carriage return by using `/* */` comment brackets or to use the `///` line-join indicator. Thus our code fragment could also read

fragment of example.do

```
use mydata
summarize weight price displ headroom rep78 length turn gear_ratio /*
      */  if substr(company,1,4)=="Ford" |    /*
      */     substr(company,1,2)=="GM", detail
generate byte ford = substr(company,1,4)=="Ford"
generate byte gm = substr(company,1,2)=="GM"
```

fragment of example.do

or

fragment of example.do

```
use mydata
summarize weight price displ headroom rep78 length turn gear_ratio ///
        if substr(company,1,4)=="Ford" |    ///
           substr(company,1,2)=="GM", detail
generate byte ford = substr(company,1,4)=="Ford"
generate byte gm = substr(company,1,2)=="GM"
```

fragment of example.do

◁

16.1.4 Error handling in do-files

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 19.5      // (or version 19 if you do not have StataNow)
. 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 19.5      // (or version 19 if you do not have StataNow)
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:

```
. do myjob2
. version 19.5      // (or version 19 if you do not have StataNow)
. 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 19.5      // (or version 19 if you do not have StataNow)
. 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**.

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

```
begin myjob3.do
version 19.5      // (or version 19 if you do not have StataNow)
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 19.5      // (or version 19 if you do not have StataNow)
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.

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\Stata19\Stata" /s do myjob
   ```

   or

   ```
   C:\data> "C:\Program Files\Stata19\Stata" /b do myjob
   ```

   to run in batch mode, assuming that you have installed Stata in the folder `C:\Program Files\Stata19`. `/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.
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.

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,

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.

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/.