# The Workflow of Data Analysis Using Stata

J. SCOTT LONG
*Departments of Sociology and Statistics*
*Indiana University–Bloomington*

A Stata Press Publication
StataCorp LP
College Station, Texas



Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in LaTeX 2ε
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-59718-047-5
ISBN-13: 978-1-59718-047-4

*(Pages omitted)*

# Contents

*(Pages omitted)*

# Preface

This book is about methods that allow you to work efficiently and accurately when you analyze data. Although it does not deal with specific statistical techniques, it discusses the steps that you go through with any type of data analysis. These steps include planning your work, documenting your activities, creating and verifying variables, generating and presenting statistical analyses, replicating findings, and archiving what you have done. These combined issues are what I refer to as the *workflow of data analysis*. A good workflow is essential for replication of your work, and replication is essential for good science.

My decision to write this book grew out of my teaching, researching, consulting, and collaborating. I increasingly saw that people were *drowning* in their data. With cheap computing and storage, it is easier to create files and variables than it is to keep track of them. As datasets have become more complicated, the process of managing data has become more challenging. When consulting, much of my time was spent on issues of data management and figuring out what had been done to generate a particular set of results. In collaborative projects, I found that problems with workflow were multiplied. Another motivation came from my work with Jeremy Freese on the package of Stata programs known as SPost (Long and Freese 2006). These programs were downloaded more than 20,000 times last year, and we were contacted by hundreds of users. Responding to these questions showed me how researchers from many disciplines organize their data analysis and the ways in which this organization can break down. When helping someone with what appeared to be a problem with an SPost command, I often discovered that the problem was related to some aspect of the user's workflow. When people asked if there was something they could read about this, I had nothing to suggest.

A final impetus for writing the book came from Bruce Fraser's *Real World Camera Raw with Adobe Photoshop CS2* (2005). A much touted advantage of digital photography is that you can take a lot of pictures. The catch is keeping track of thousands of pictures. Imaging experts have been aware of this issue for a long time and refer to it as *workflow*—keeping track of your work as it flows through the many stages to the final product. As the amount of time I spent looking for a particular picture became greater than the time I spent taking pictures, it was clear that I needed to take Fraser's advice and develop a workflow for digital imaging. Fraser's book got me thinking about data analysis in terms of the concept of a workflow.

After years of gestation, the book took two years to write. When I started, I thought my workflow was very good and that it was simply a matter of recording what I did. As writing proceeded, I discovered gaps, inefficiencies, and inconsistencies in what I did.

Sometimes these involved procedures that I knew were awkward, but where I never took the time to find a better approach. Some problems were due to oversights where I had not realized the consequences of the things I did or failed to do. In other instances, I found that I used multiple approaches for the same task, never choosing one as the best practice. Writing this book forced me to be more consistent and efficient. The advantages of my improved workflow became clear when revising two papers that were accepted for publication. The analyses for one paper were completed before I started the workflow project, whereas the analyses for the other were completed after much of the book had been drafted. I was pleased by how much easier it was to revise the analyses in the paper that used the procedures from the book. Part of the improvement was due to having better ways of doing things. Equally important was that I had a consistent and documented way of doing things.

I have no illusions that the methods I recommend are the best or only way of doing things. Indeed, I look forward to hearing from readers who have suggestions for a better workflow. Your suggestions will be added to the book's web site. However, the methods I present work well and avoid many pitfalls. An important aspect of an efficient workflow is to find one way of doing things and sticking with it. Uniform procedures allow you to work faster when you initially do the work, and they help you to understand your earlier work if you need to return to it at a later time. Uniformity also makes working in research teams easier because collaborators can more easily follow what others have done. There is a lot to be said in favor of having established procedures that are documented and working with others who use the same procedures. I hope you find that this book provides such procedures.

Although this book should be useful for anyone who analyzes data, it is written within several constraints. First, Stata is the primary computing language because I find Stata to be the best, general-purpose software for data management and statistical analysis. Although nearly everything I do with Stata can be done in other software, I do not include examples from other packages. Second, most examples use data from the social sciences, because that is the field in which I work. The principles I discuss, however, apply broadly to other fields. Finally, I work primarily in Windows. This is not because I think Windows is a better operating system than Mac or Linux, but because Windows is the primary operating system where I work. Just about everything I suggest works equally well in other operating systems, and I have tried to note when there are differences.

I want to thank the many people who commented on drafts or answered questions about some aspect of workflow. I particularly thank Tait Runfeldt Medina, Curtis Child, Nadine Reibling, and Shawna L. Rohrman whose detailed comments greatly improved the book. I also thank Alan Acock, Myron Gutmann, Patricia McManus, Jack Thomas, Leah VanWey, Rich Watson, Terry White, and Rich Williams for talking with me about workflow. Many people at StataCorp helped in many ways. I particularly want to thank Lisa Gilmore for producing the book, Jennifer Neve for editing, and Annette Fett for designing the cover. David M. Drukker at StataCorp answered many of my questions. His feedback made it a better book and his friendship made it more fun to write.

Some of the material in this book grew out of research funded by NIH Grant Number R01TW006374 from the Fogarty International Center, the National Institute of Mental Health, and the Office of Behavioral and Social Science Research to Indiana University–Bloomington. Other work was supported by an anonymous foundation and The Bayer Group. I gratefully acknowledge support provided by the College of Arts and Sciences at Indiana University.

Without the unintended encouragement from my dear friend Fred, I would not have started the book. Without the support of my dear wife Valerie, I would not have completed it. Long overdue, this book is dedicated to her.

*Bloomington, Indiana*
*October 2008*

Scott Long

*(Pages omitted)*

# 3 Writing and debugging do-files

Before discussing how to use Stata for specific tasks in your workflow, I want to talk about using Stata itself. Part of an effective workflow is taking advantage of the powerful features of your software. Although you can learn the basics of Stata in an hour, to work efficiently you need to understand some of its more advanced features. I am not talking about specific commands for transforming data or fitting a model, but rather about the interface of the program, the principles for writing do-files, and how to automate your work. The time you spend learning these tools will quickly be recovered as you apply these tools to your substantive work. Moreover, each of these tools contributes to the accuracy, efficiency, and replicability of your work. This chapter discusses writing and debugging do-files. Chapter 4 introduces powerful tools for automating your work. The tools and techniques from chapters 3 and 4 are used and expanded upon in chapters 5–7 where different parts of the workflow of data analysis are discussed.

I begin the chapter reviewing three ways to execute commands: submit them from the Command window, construct them with dialog boxes, or include them in do-files. Each approach has its advantages, but I argue that the most effective way to work is with do-files. Because the examples in the rest of the book depend on do-files, I discuss in section 3.2 how to write more effective do-files that are easier to understand and that will continue to work on different computers, in later versions of Stata, and after you change the directories on your computer. Although these guidelines can prevent many errors, sometimes your do-files will not work. Section 3.3 describes how to debug do-files, and section 3.4 describes how to get help when the do-files still do not work.

I assume that you have used Stata before, although I do not assume that you are an expert. If you have not used Stata, I encourage you to read [GS] *Getting Started with Stata* and those sections of the [U] *User's Guide* that seem most useful. Appendix A discusses how the Stata program works, which directories it uses, how to use Stata on a network, and ways to customize Stata. Even experienced users may find some useful information there.

## 3.1 Three ways to execute commands

There are three ways to execute commands in Stata. You can submit commands interactively from the command line. This is ideal for trying new things and exploring your data. You can use dialog boxes to construct and submit commands, which is particularly useful for finding the options you need when exploring new commands. You can also run do-files, which are text files that contain Stata commands. Each method has

advantages, but I will argue that serious work requires do-files. Indeed, I only use the other methods to help me write do-files.

### 3.1.1 The Command window

You can type one command at a time in the Command window. Type the command and press Enter. When experimenting with how a command works or checking some aspect of my data, I often use this method. I try a command, press Page Up to redisplay the command in the Command window, revise it, press Enter to run it again, and so on. The disadvantage of working interactively is that you cannot easily rerun your commands at a later time.

Stata has a number of features that are very useful when working from the Command window.

#### Review window

The commands you submit from the Command window are echoed to the Review window. When you click on a command in the Review window, it is pasted into the Command window where you can revise it and then submit it by pressing Enter. If you double-click on a command in the Review window, it is sent to the Command window and automatically executed.

#### Page up and page down

The Page Up and Page Down keys let you scroll through the commands in the Review window. Pressing Page Up multiple times moves through multiple prior commands. Page Down moves you forward to more recent commands. When a command appears in the Command window, you can edit it and then rerun it by pressing Enter.

#### Copy and paste

You can highlight and copy text from the Command window or the Results window. This information can be pasted into other applications, such as your text editor. This allows you to debug a command interactively, then copy the corrected commands to your do-file.

#### Variables window

The Variables window lists the variables in the current dataset. If you click on a variable name in this window, the name is pasted into the Command window. This is often the fastest way to construct a list of variable names. You can then copy the list of names and paste it into your do-file.

#### Logging with log and cmdlog

If you want to reproduce the results you obtain interactively, you should save your session to a log file with the `log using` command. You can then edit the log file to create a do-file to rerun the commands. Suppose that you start an interactive session with the command

```
log using datacheck, replace text
```

After you are done with your session, you close the log file with `log close` to create the file `datacheck.log`. To create a do-file that will produce the same results, you can copy the log file to `datacheck.do`, remove the `.`'s in front of each command, and delete the output. This is tedious but sometimes quite useful. An alternative is to use `cmdlog` to save your interactive commands. For example, `cmdlog using datacheck.do, replace` saves all commands from the Command window (but no output) to a file named `datacheck.do`, which you can use to create your do-file. You close a `cmdlog` with the `cmdlog close` command.

### 3.1.2 Dialog boxes

You can use dialog boxes to construct commands using point-and-click. You open a dialog box from the menus in Stata by selecting the task you want to complete. For example, to construct a scatterplot matrix, you select **Graphics** (Alt+G) > **Scatterplot matrix** (s, Enter). Next you select options using your mouse. After you have selected your options, click on the **Submit** button to run the command. The command you submit is echoed to the Results window so that you can see how to type the command from the Command window or with a do-file. If you press Page Up, the command generated by the dialog box is brought into the Command window where you can edit it, copy it, or rerun it.

Although dialog boxes are easy to learn, they are slow to use. However, dialog boxes are very efficient when you are looking for an option used by a complex command. I use them frequently when creating graphs. I select the options I need, run the command by clicking on the **Submit** button, and then copy the command from the Results window to my do-file.

### 3.1.3 Do-files

Over 99% of the work I do in Stata uses do-files. Do-files are simply text files that contain your commands. Here is a simple do-file named `wf3-intro.do`.

```
log using wf3-intro, replace text
use wf-lfp, clear
summarize lfp age
log close
```

This program loads data on labor-force participation and computes summary statistics for two variables. If you have installed the Workflow package in your working directory, you can run this do-file by typing the command `do wf3-intro.do`.[1] The extension `.do` is optional, so you could simply type `do wf3-intro`. After submitting the file, I obtain these results:

```
      log:  e:\workflow\work\wf3-intro.log
  log type:  text
 opened on:   3 Apr 2008, 05:27:01

. use wf-lfp, clear
(Workflow data on labor force participation \ 2008-04-02)

. summarize lfp age
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         lfp |       753    .5683931    .4956295          0          1
         age |       753    42.53785    8.072574         30         60

. log close
      log:  e:\workflow\work\wf3-intro.log
  log type:  text
 closed on:   3 Apr 2008, 05:27:01
```

That is how simple it is to run a do-file. If you have avoided them in the past, this is a good time to take an hour and learn how they work. That hour will save you many hours later.

I use do-files for two major reasons. First, with do-files you have a record of the commands you ran, so you can rerun them in the future to replicate your results or to modify the program. Recall the research log on page 41 that documented a problem with how a variable was created. If I had not been using do-files, I would have needed to reconstruct weeks of work rather than changing a few lines of code and rerunning the do-files in sequence. Second, with do-files, you can use the powerful features of your text editor, including copying, pasting, global changes, and much more (see the Workflow web site for information on text editors). The editor built into Stata can be opened several ways: run the command `doedit`, select the Do-file Editor from the **Window** menu of Stata, or click on the Do-file Editor icon. For details on the Stata Do-file Editor, type `help doedit`, or see [R] **doedit**.

## 3.2 Writing effective do-files

The rest of the book assumes that you are using do-files to run commands, with the exceptions of occasionally testing commands from the Command window or using dialog boxes to track down options. In this section, I consider how to write do-files that are robust and legible. Here is what I mean by these terms:

1. Appendix A explains the idea of a working directory. The *Preface* has information on installing the Workflow package.

*Robust do-files* produce exactly the same result when run at a later time or on another computer.

*Legible do-files* are documented and formatted so that it is easy to understand what is being done.

Both criteria are important because they make it possible to replicate and correctly interpret your results. As a bonus, robust and legible do-files are easier to write and debug. To illustrate these characteristics of do-files, I use examples that contain basic Stata commands. Although you might encounter a command that you have not seen before, you should still be able to understand the general points I am making even if you do not follow the specific details.

## 3.2.1 Making do-files robust

A do-file is robust if it produces exactly the same result when it is rerun on your computer or run on a different computer. The key to writing robust do-files is to make sure that results do not depend on something left in memory (e.g., from another do-file or a command submitted from the Command window) or how your computer is set up (e.g., the directory structure you use). To operationalize this standard, imagine that after running a do-file you copy this file and all datasets used to a USB drive, insert the USB drive in another computer, and run the do-file again without any changes. If you cannot do this and get the same results, replication will be difficult or impossible. Here are my suggestions for making your do-files robust.

### Make do-files self-contained

Your do-file should not rely on something left in memory by a prior do-file or commands run from the Command window. A do-file should not use a dataset unless it loads the dataset itself. It should not compute a test of coefficients unless it estimates those coefficients. And so on. To understand why this is important, consider a simple example. Suppose that `wf3-step1.do` creates new variables and `wf3-step2.do` fits a model. The first program loads a dataset and creates two variables indicating whether a family has young children and whether a family has older children:

```
log using wf3-step1, replace text
use wf-lfp, clear
generate hask5 = (k5>0) & (k5<.)
label var hask5 "Has children less than 5 yrs old?"
generate hask618 = (k618>0) & (k618<.)
label var hask618 "Has children between 6 and 18 yrs old?"
log close
```

The program `wf3-step2.do` estimates the logit of `lfp` on seven variables, including the two created by `wf3-step1.do`:

```
log using wf3-step2, replace
logit lfp hask5 hask618 age wc hc lwg inc, nolog
log close
```

If these programs are run one after the other, with no commands run in between, everything works fine. What if the programs are not run in sequence? For example, suppose that I run `wf3-step1.do` and then run other do-files or commands from the Command window. Or I might later decide that the model should not include `age`, so I modify `wf3-step2.do` and run it again without running `wf3-step1.do` first. Regardless of the reason, if I run the second do-file without running `wf3-step1.do` first, I get the following error:

```
. logit lfp hask5 hask618 age wc hc lwg inc, nolog
no variables defined
r(111);
```

The error occurs because the dataset is no longer in memory. I might change the program so that the original dataset is loaded

```
log using wf3-step2, replace
use wf-lfp, clear
logit lfp hask5 hask618 age wc hc lwg inc, nolog
log close
```

Now the error is

```
. logit lfp hask5 hask618 age wc hc lwg inc, nolog
variable hask5 not found
r(111);
```

This error occurs because `hask5` is not in the original dataset but was created by `wf3-step1.do`.

To avoid this type of problem, I can modify the two programs to make them self-contained. I change the first program so that it saves a dataset with the new variables (file: `wf3-step1-v2.do`):

```
log using wf3-step1-v2, replace
use wf-lfp, clear
generate hask5 = (k5>0) & (k5<.)
label var hask5 "Has children less than 5 yrs old?"
generate hask618 = (k618>0) & (k618<.)
label var hask618 "Has children between 6 and 18 yrs old?"
save wf-lfp-v2, replace
log close
```

I change the second program so that it loads the dataset created by the first program (file: `wf3-step2-v2.do`):

```
log using wf3-step2-v2, replace
use wf-lfp-v2, clear
logit lfp hask5 hask618 age wc hc lwg inc, nolog
log close
```

The do-file `wf3-step2-v2.do` still requires running `wf3-step1-v2.do` to create the new dataset, but it does not require running `wf3-step2-v2.do` immediately after `wf3-step1-v2.do` or even that it be run in the same Stata session.

There are a few exceptions of do-files that need to be run in sequence. For example, if I am doing postestimation analysis of coefficients from a model that takes a long time to fit (e.g., `asmprobit`), I do not want to refit the model repeatedly while I debug the postestimation commands. I would use one do-file to fit the model and a second do-file for postestimation analysis. The second do-file only works if the prior do-file was run. To ensure that I remember that the programs need to be run in tandem, I add a comment to the second do-file:

```
// Note: This do-file assumes that program1.do was run first.
```

After debugging the second program, I would combine the two do-files to create one do-file that is self-contained.[2]

#### Use version control

If you run a do-file at a later time, perhaps to verify a result or to modify some part of the program, you could be using a newer version of Stata. If you share a do-file with a colleague, she might be using a different version of Stata. Sometimes new versions of Stata change the way in which a statistic is computed, perhaps reflecting advances in computational methods. When this occurs, the same commands can produce different results in different versions of Stata. Newer versions of Stata might change the name of a command (e.g., `clear` in Stata 9 was changed to `clear all` in Stata 10). The solution is to include a `version` command in your do-file. For example, if your do-file includes the command `version 6` and you run the do-file in Stata 10, you will get exactly the same answer that you would obtain in Stata 6. This is true even if Stata 10 computes the particular statistic differently (e.g., the computations in some `xt` commands changed between Stata 6 and Stata 10). On the other hand, if your do-file includes the command `version 10` and you try to run the program in Stata 8.2, you get an error:

```
. version 10
this is version 8.2 of Stata; it cannot run version 10.0 programs
     You can purchase the latest version of Stata by visiting
     http://www.stata.com.
r(9);
```

You could rerun the program after changing the `version 10` command to `version 8.2`. There is no guarantee that programs written for newer versions of Stata will work in older versions.

#### Exclude directory information

I almost never specify a directory location in commands that read or write files. This lets my do-files run even if the directory structure of the computer I am using changes. For example, suppose that my do-file loads data with the command

2. With Stata 10, I might use the new **estimates save** command to save the estimates in the first do-file and then load them at the start of the second do-file that does postestimation analysis. This would allow each program to be self-contained, even when debugging the second program. For details, see [R] **estimates save**.

```
use c:\data\wf-lfp, clear
```

Later, when I rerun the do-file on a computer where the dataset is stored in `d:\data\`, I get an error:

```
. use c:\data\wf-lfp, clear
file c:\data\wf-lfp.dta not found
r(601);
```

To avoid such problems, I do not include a directory location. For example, to load `wf-lfp.dta`, I use the command

```
use wf-lfp, clear
```

When no directory is specified, Stata looks in the working directory.

The working directory is the directory you are in when you launch Stata.[3] In Windows, you can determine your working directory by typing `cd`. For example,

```
. cd
e:\data
```

In Mac OS or Unix, you use the `pwd` command. For example, on a Mac:

```
. pwd
~:data
```

You can change your working directory with the `cd` command. For example, when testing commands for this book, I used the `e:\workflow\work` directory. To make this my working directory, I would type

```
cd e:\workflow\work
```

To change to the working directory used for the CWH project, I would type

```
cd e:\cwh\work
```

If the directory name includes blanks or special characters, you need to put the name in quotes. For example,

```
cd "c:\Documents and Settings\jslong\Projects\workflow\work"
```

The advantage of not including directory locations in your do-file is that you can run your do-files on other computers without any changes. Although it is tempting to say that you will always keep your data in the same place (e.g., `d:\data`), this is unlikely for several reasons.

1. If you change computers or add a new drive to your computer, the drive letters might change.

3. Appendix A has a detailed discussion of the directories used by Stata.

2. If you keep data on external drives, including USB flash drives, the operating system will not always assign the drive the same drive letter.
3. If you reorganize your files, the directory structure could change.
4. When you restore files from your archive, you might not remember what the directory structure used to be.

If you share do-files with a collaborator or someone helping you debug your program, they will probably have a different directory structure than yours. If you hardcode the directory, the person you send the do-file to must either create the same directory structure or change your program to load data from a different directory. When the collaborator sends you the corrected do-file, you will have to undo the directory changes that were made, and so on. All things considered, I think that it is best practice to write do-files that do not require a particular directory structure or location for the data. There are two exceptions that are useful. First, if you are loading a dataset from the web, you need to specify the specific location of the file. For example, `use http://www.stata-press.com/data/r10/auto, clear`. Second, you can specify relative directories. Suppose there is a subdirectory `\data` located in your working directory. To keep things organized, you place all your datasets in this directory, while your do-files and log files remain in your working directory. You can assess the datasets by specifying the subdirectory. For example, `use data\wf-lfp, clear`.

#### Include seeds for random numbers

Random numbers are used in a variety of ways in data analysis. For example, if you are bootstrapping standard errors, Stata draws repeated random samples. If you try to replicate results that use random numbers, you need to use the same random numbers or you will obtain different results. Stata uses pseudorandom numbers that are generated by a formula in which one pseudorandom number is transformed to create the next number. This transformation is done in such a way that the sequence of numbers behaves as if it were truly random. With pseudorandom numbers, if you start with the same number, referred to as the *seed*, you will re-create exactly the same sequence of numbers. Accordingly, to reproduce exactly the same results when you rerun a program that uses pseudorandom numbers, you need to start with the same seed. To set the seed, use the command

```
set seed #
```

where # is a number you choose. For example, `set seed 11020`. For further details and an example, see section 7.6.3.

### 3.2.2 Making do-files legible

I use the term legible to describe do-files that are internally documented and carefully formatted. When writing a do-file, particularly one that does complex statistical analyses or data manipulations, it is easy to get caught up in the logic of what you are doing

and forget about documenting your work and formatting the file to make the content clear. Applying uniform procedures for documenting and formatting your do-files makes them easier to debug and helps you and your collaborators understand what you did. There are many ways to make your do-files easier to understand. If you do not like my stylistic suggestions, feel free to create your own style. The important thing is to establish a style that you and others find legible. If you are collaborating, try to agree upon a common style for writing do-files that makes it simpler to share programs and results. Clear and well-formatted do-files are so important for working efficiently that one of the first things I do when helping someone debug a program is to reformat their do-file to make the code easier to read.

### Use lots of comments

I have never returned to a do-file and regretted how many comments it had, but I have often wished that I had written more. Commands that seem obvious when I write them can be obscure later. I try to add at least a few comments when I initially write a do-file. After the program works the way I want, I add additional comments. These comments are used both to label the output and to explain commands and options that might later be confusing.

Stata provides three ways to add comments. The first two create comments on a single line, whereas the third allows you to easily write multiline comments. The method you use is largely a matter of personal preference.

#### * comments

If you start a line with a `*`, everything that follows on that line is treated as a comment. For example,

```
* Select sample based on age and gender
```

or

```
* The following analysis includes only those people
* who responded to all four waves of the survey.
```

You can temporarily stop a command from being executed:

```
* logit lfp wc hc age inc
```

#### // comments

You can add comments after a `//`. For example,

```
// Select sample based on age and gender
```

This method can also be used at the end of a command. For example,

```
logit lfp wc hc // includes only education, add wages later
```

#### /* and */ comments

Everything between an opening `/*` and a closing `*/` is treated as a comment. This is particularly useful for comments that extend over multiple lines. For example,

```
/*
   These analyses are preliminary and are based on those countries
   for which complete data were available by January 17, 2005.
*/
```

#### Comments as dividers

Comments can be used as dividers to distinguish among different parts of your program. For example,

```
************************************
**  Descriptive statistics by gender
```

or

```
// ================================================
// = Logit models of depression on genetic factors
```

#### Obscure comments

Comments are useful only when they are accurate and clear. When writing a complex do-file, I use comments to remind me of things I need to do. For example,

```
* check this. wrong variable?
```

or

```
* see ekp´s comment and model specification
```

After the program is written, these comments should be deleted because later they will be confusing.

### Use alignment and indentation

It is easier to verify your commands if things line up. For example, here are two ways to format the same commands for renaming variables. Which is easier for spotting a mistake? This?

(*Continued on next page*)