Saving Time

Bill Rising
StataCorp LLC

2018 Italian Stata Users Group Meeting
Bologna
15 Nov 2018

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Stata's User Interface . . . 2

2 Saving Time in the File System **2**
- 2.1 Interacting with the OS . . . 2
- 2.2 The fileutils Package . . . 3
- 2.3 Moving Around Quickly . . . 4
- 2.4 Special Places . . . 5

3 Saving Time Writing Code **7**
- 3.1 Saving Time Writing Code . . . 7
- 3.2 Editing Stata Code . . . 8

4 Conclusion **9**
- 4.1 Conclusion . . . 9

1 Introduction

1.1 Background

Saving time

- Saving time is a Good Thing
- Using time to save time can be a good thing
  - It can also be a bad thing if it takes too much time to save time
- At all times, one needs to compare the total time saved against the time used to save time
  - A saving of 5 seconds on a task done 10 times per workday will save 3.5 hours per year
    - Not a huge amount of time, but worth spending a few hours programming nicely

---

Three Tools

- This talk will talk about three tools for saving time with Stata
  - The `fileutils` package for interacting with the operating system
  - The `smcl2do` package for extracting commands from a log file
  - The Emacs text editor with `ado-mode` installed
    - Time permitting

1.2 Stata's User Interface

Built-in Time Savers

- Stata has some time-savers
- Dialog boxes
  - Save time for complicated graphs
- Command-window shortcuts
  - Reusing commands with page up and page down
  - Tab-completion of variable names
  - Tab-completion of file names

---

2 Saving Time in the File System

2.1 Interacting with the OS

Stata's Working Directory

- The working directory in Stata is a great idea
  - One folder per project
  - Move there and work
  - Easy to open and save project-related files
- This is great for a single project
  - It can get complex for many projects

A Fractured Existence

- Imagine a computer where
  - Hobbies are stored one place
  - Official projects are stored another place
  - Author Support projects are another place
  - Homebrewed projects are another place
  - Bug reports are in another place
  - etc.
- So... a typical computer, but possibly with different types of projects

---

A Plethora of Files

- Imagine projects with mixtures of files
  - Some Stata-related
    - do-files
    - log files
    - graphs
  - Some not so much
    - pdf files
    - tex files
    - html files
    - MS Word files
- It would be nice to see these quickly

2.2 The fileutils Package

The `fileutils` Package

- This is a package of 4 commands for interacting with the OS
  - it is available from the SSC

    ```
    . ssc install fileutils
    ```
- Three commands for jumping around
  - `go`, `pushd`, and `popd`
- One command for interacting with the OS
  - `opendir`
- Let's start with the last one, because it is quick

`opendir`

- It's nice to look at the files in your working directory

  ```
  . dir
  ```
  - Sometimes, however, it would be nice really see the files
- The `opendir` command opens an Explorer/Finder/File window in any OS

  ```
  . opendir
  ```
- If you give it a path, it'll open that folder in the OS

  ```
  . opendir ..
  ```
- This works in any OS

2.3 Moving Around Quickly

Jumping from Place to Place

- Now suppose that we would like to move from one place to another
- This can be done via the OS
  - On the Mac, this is not too onerous
  - In Windows it is
    - The dialog has no remembrance of things past
- It can be done via the Command window, using tab completion

```
. cd "~/Desktop/2018_italy_saving_time/data"
```

Making a Quick Visit

- Sometimes it is worth visiting quickly ...

```
. cd "~/Documents/Scratch"
```

- ... doing some work ...

```
. * work work work
```

- ... and coming back

```
. cd "~/Desktop/2018_italy_saving_time/data"
```

- Doing this by hand is miserable
  - Copying and pasting can help, but you need to remember to copy!

`pushd` and `popd`

- Here are two simple commands for jumping back and forth:
  - `pushd` changes directory, but keeps track of the current directory for later
  - So... this is a substitute for `cd`
  - `popd` jumps back to the last pushed directory
- You can push multiple times in a row and build a stack of directories through which you can then backtrack
  - Though this isn't all that useful

Example of Pushing and Popping

- Here is the above example of jumping around using these commands
- First: go to the Scratch directory

```
. pushd "~/Documents/Scratch"
/Users/brising/Documents/Scratch
```

- Do some work

```
. * work work work
```

- Come back

```
. popd
/Users/brising/Desktop/2018_italy_saving_time/data
```

- This is nice, but not that nice

2.4 Special Places

Known Special Locations

- Better than this is some way to jump to specially named places
- This is the purpose of the go command
- Here is my (fake) current state of shortcuts

```
. go

packages  -> ~/Universal/Custom/Stata/ado/Packages
lessons   -> ~/Shuttle/Training/Lessons
planning  -> ~/Shuttle/Training/Planning/
personal  -> /Universal/Custom/Stata/ado/new/
tracking  -> ~/Documents/Me/StatFun/Tracking
sessions  -> /Volumes/Shuttle/Training/Sessions
bugs      -> /Volumes/Shuttle/Bugs and Testing
scratch   -> ~/Documents/Scratch
fileutils -> /Users/brising/Universal/Custom/Stata/ado/Packages/fileutils
getpkgs   -> ~/Universal/Custom/Stata/ado/Packages/getpkgs/cert
```

- I could jump to the scratch directory . . . by typing or clicking

```
. go scratch

/Users/brising/Documents/Scratch
```

- . . . and come back via popd

```
. popd

/Users/brising/Desktop/2018_italy_saving_time/data
```

Adding a Shortcut

- The simplest way to add a shortcut is to be in the directory
- Here is where we are now

```
. pwd

/Users/brising/Desktop/2018_italy_saving_time/data
```

- Adding a shortcut is simple

```
. go add timetalk

timetalk -> /Users/brising/Desktop/2018_italy_saving_time/data
```

Removing a Shortcut

- After a while, the list of shortcuts can get large or a project could be finished
- Imagine that I was done working on the getpkgs project
- I can get rid of the shortcut simply

```
. go drop getpkgs

Dropped nickname getpkgs
```

- Now I can see it is gone

```
. go list
```

```
packages  -> ~/Universal/Custom/Stata/ado/Packages
lessons   -> ~/Shuttle/Training/Lessons
timetalk  -> /Users/brising/Desktop/2018_italy_saving_time/data
planning  -> ~/Shuttle/Training/Planning/
personal  -> /Universal/Custom/Stata/ado/new/
tracking  -> ~/Documents/Me/StatFun/Tracking
sessions  -> /Volumes/Shuttle/Training/Sessions
bugs      -> /Volumes/Shuttle/Bugs and Testing
scratch   -> ~/Documents/Scratch
fileutils -> /Users/brising/Universal/Custom/Stata/ado/Packages/fileutils
```

Going to Subfolders

- The go command allows subfolders (subdirectories)
- There is a `neato` folder inside our current folder
- Let's go somewhere else

```
. go tracking

/Users/brising/Documents/Me/StatFun/Tracking
```

- Now I can go to `neato` simply enough

```
. go timetalk/neato

/Users/brising/Desktop/2018_italy_saving_time/data/neato
```

- And then return to `tracking`

```
. popd

/Users/brising/Documents/Me/StatFun/Tracking
```

- And return to the data folder for the talk

```
. popd

/Users/brising/Desktop/2018_italy_saving_time/data
```

Other Small Things

- The go command also allows copying, dropping and renaming shortcuts
  - ◇ Take a look at `help go`
  - ◇ The `noexist` and `nowrite` options are for having a do-file which creates the shortcuts quickly

---

Aside: How go Works

- It creates a do-file in your `PERSONAL` folder named `golookup_`*OS*`.do`
  - ◇ The *OS* gets replaced by your operating system
    - ⋆ This oddity is needed for someone working/testing for multiple operating systems on one machine
- The do-file gets read when setting up a Mata object to hold the lookups
  - ◇ The object is called an associative array by Stata or a heap by some other languages

---

Aside: Where the Shortcuts Get Saved

- By default, the do-file gets written every time you make a change
    - You can squelch a write with the `nowrite` option
        - But then you should go `write` at some point before quitting Stata
    - This is in case someone is, say, writing shortcuts en masse
- The do-file is useful because it allows hand-editing
- For this talk, I've put the lookup file with the talk
    - This is really a piece added for debugging
- Here is the lookup file we are using here

```
. doedit "golookup_MacOSX"
```

Wrapup of go

- I find go very handy, and it saves many many many small bits of time
- It did take a while to write, but it was done as an exercise to learn the programming methods in Bill Gould's book about programming Mata

---

3 Saving Time Writing Code

3.1 Saving Time Writing Code

Saving Time Writing Code

- Working in the Do-file Editor is all well and good, but it can be slow
    - Typos cause bugs
    - No tab-completion of filenames
    - No tab-completion of variable names
    - No immediate feedback on how a command works
- It is passable when you know exactly what to type
- It is faster to work with the Command window
    - Though weird

Official Stata

- In official Stata, there is the `cmdlog` command for generating a do-file as you type
- It is also slow
    - It saves all commands to the generated do-file
        - Commands ending in errors
        - Commands which do nothing (`help`, `edit`, etc.)

Enter `smcl2do`

- `smcl2do` is simple-minded
  - It extracts commands from a log file to make a do-file
  - It excludes commands ending in errors
  - It can exclude commands which do nothing
- It's available on `ssc`

```
. ssc install smcl2do
```

A Quick Example

- Imagine we were getting ready to do some work
- We can start a log

```
. log using example, name(example) replace
```

- Then type a few commands (not shown)
- Then close the log

```
. log close example
```

- Then convert the file

```
. smcl2do using example, clean
```

- Then open it in the do-file editor

```
. doedit example
```

Why Do This?

- It's quite quick for writing uninspired code
  - Grinding through a series of data management tasks, for example
- It's useful for writing test code
- It is a good way to sketch out a do-file
  - There will be a future extension for omitting/keeping commands automatically
    - This would make it better for experimenting

3.2 Editing Stata Code

Emacs and `ado-mode`

- If you find the Do-file Editor limited, try looking for other text editors
- I use Emacs, and edit my do-files with a "mode" called `ado-mode`
  - I use Aquamacs (`http://aquamacs.org`) which makes Emacs much nicer, but is Mac-only
- This is available at `https://www.louabill.org/Stata/`

Advantages

- Can submit code to Stata and have the commands in the Review window
- Can submit code with // and /// comments without issue
- Can open help and/or code for commands easily
    - Even personal or downloaded commands
- Has better syntax highlighting
- Has supplied templates for ado, do, and help files

Disadvantages

- Installation is not friendly
- Emacs is an old text editor built in the early 1980's
    - So it has strange keyboard shortcuts

---

4 Conclusion

4.1 Conclusion

Conclusion

- Saving time is a worthwhile endeavour
- Saving time should not be at the cost of using more time
- The trick is assessing the effort and the longevity of the shortcuts

---

Index

C
cd command, 3, 4
cmdlog command, 7

D
dir command, 3

F
fileutils package, 3

G
go command, 5–7

O
opendir command, 3

P
popd command, 4
pushd command, 4

S
smcl2do command, 8
ssc install command, 3, 8

W
working directory, 2