Introduction to postfile commands in Stata
Author: Beksahn Jang
Last updated: March 16th, 2016
Stata version: 13.0

NOTE: This was written to supplement Jurgita Abromavicute's review of postfile: http://stataresources.blogspot.com/2014/03/introduction-to-p-postfile-in-stata.html. Areas of overlap include the basic definitions and syntax of post. Additions to Jurgita's blog post include: 1) Distinction between *collapse* and *postfile*, 2) A basic template for creating a dataset using post, 3) A demonstration of why ordering of variables and observations matters, 4) A demonstration of one type of syntax error—failure to include parentheses around expressions, 5) A more in-depth example of creating a dataset using results from a regression.

What is "post"?
The postfile commands are a suite of commands designed to post results in Stata. You can think of it as creating a new dataset composed of items specified by either results saved from commands (such as from regressions) or specified as a subset of an original dataset.

When should one use it?
They generally have three major uses:
1) to create a subset of an original dataset
2) to save the results of statistical analyses, like regressions.
3) can be used to perform Monte Carlo-type experiments

How is it different from *collapse*?
Another command that creates a new dataset of results is the *collapse* command. The greatest difference between *collapse* and *post* is that *collapse* provides descriptive statistics and cannot provide results from a number of commands that *post* provides, most notably regression results. Type *help collapse* in Stata for a list of descriptive statistics results for *collapse*.

Syntax
There are multiple commands within the postfile command suite:
postfile postname newvarlist using filename [, replace]
post postname (exp) (exp) … (exp)
postclose postname
postutil dir
postutil clear
The first three—postfile, post, and postclose—are the main commands used in creating a new dataset while postutil dir and postutil clear are used to manage existing postfiles.
"Postfile" can be considered the "first step" in creating your dataset. It creates a dataset file where results will be saved and it declares variable names (i.e.

newvarlist) for that dataset. *NOTE*: pay attention to the order in which the variable names are listed. This order will correspond to their in the dataset.
"Post" can be considered the point at which values are specified for the variables in the "postfile" command. In this sense, "postfile" is like creating buckets (i.e. the variable names) and "post" is like filling those buckets with specified values or observations (like regression results, among other possible things).
"Postclose" simply declares the end of posting observations.
"Postutil dir" will list all open postfiles.
"Postutil clear" will close all open postfiles.

Syntax (continued): a useful template
Because the three commands—postfile, post, and postclose—are used together and because the postfile suite is very specific in its format, a basic template can be useful.
The following template comes from the "Data Today" blog:
(http://datatoday.blogspot.com/2011/11/using-postfile-in-stata.html)

```
1   tempname memhold
2   postfile `memhold' str10 stringvar numvar using
3   "C:/Filename.dta" , replace
4
5   post `memhold' (your_str_var) (your_num_var)
6
7   postclose `memhold'
```

The "tempname" command creates macro "memhold." It is not required that you use tempname prior to using the postfile commands. It is used in this demonstration because it creates a temporary object that we will use in the following postfile commands. It is also useful in that creating "memhold" you can be sure that the object following the postfile command is not accidentally specifying a macro or other object you were unaware existed prior.
If we recall the descriptions of the post commands, the "postfile" command in line 2 creates an empty dataset and names the variables. The "post" command in line 5 creates the objects that you will save to the variables in the empty dataset created by the "postfile" command. Finally, the "postclose" command marks the end of posting observations to the dataset.
Let's try using this in one of the most basic applications where we are <u>not</u> using a preexisting dataset in the creation of a new dataset.

Example: basic
I will run the following commands to create a dataset with three variables ("A" "B" and "C") with corresponding values 1, 2, and 3.

```
clear all
tempname memhold
postfile `memhold' A B C using "simplepractice.dta", replace
post `memhold' (1) (2) (3)
postclose `memhold'
```

Line 1: I chose to clear memory in stata for this basic example to show that I was not using existing results or an original dataset to create these values.
Line 2: I create the temporary object "memhold"
Line 3: I create an empty dataset named "simplepractice.dta" with variables A B and C. The replace option is specified in case I want to run this list of commands again and need to change some of the items in the postfile command.
Line 4: I assign values to A B and C in a very direct manner—by listing them after memhold in parentheses.
Line 5: postclose notes the end of adding observations to the dataset.
This should give you a dataset that when opened in the data browser looks like this:

	A	B	C
1	1	2	3

Example: basic—ordering variables and values
We can also use this opportunity to show how ordering matters when specifying variables and values.
Let's run the commands again but this time change the order of the values to 3 2 1:

```
clear all
tempname memhold
postfile `memhold' A B C using "simplepractice.dta", replace
post `memhold' (3) (2) (1)
postclose `memhold'
```

After opening our new dataset and opening the data browser we find that the order of the values has been switched:

	A	B	C
1	3	2	1

Example: basic—syntax errors and parentheses around values
Finally, we can use this example to show why the values 1, 2, and 3 are in parentheses following the "post" command (*post `memhold' (1) (2) (3)*). Let's change the values to 3 and -2.
Let's run the same command as our first example with these new values 3 and -2 **without** parentheses:

```
clear all
tempname memhold
postfile `memhold' A B  using "simplepractice.dta", replace
post `memhold' 3 -2
postclose `memhold'
```

Running this gives us the error response "post command requires expressions be bound in parentheses." This is because post assumes its arguments are expressions and spaces are not significant in expressions. As an expression, post would have read this as 3-2 and evaluated it as 1 instead of as two separate values. Given that we specified two variables, we require two corresponding values, not 1. As a result we get an error. In older versions of post this was problematic. In the current

version of post it simply requires that you include parentheses so as to prevent this issue altogether. In the older version of post you could list values without parentheses so long as they did not reduce the number of expressions. As such 3 and -2 could be achieved as 3 1-3 without including parentheses. In the current version, however, all values—even basic ones like 1, 2, and 3 that wouldn't create this old error—require parentheses.
Now let's try something a bit more complicated. We will create a dataset of regression results in a basic format and then create another dataset of regression results using a forval loop.

Example/Applications: Creating a dataset of saved results from a regression
Before we use the post commands, let's have a look at the regression and the results we want to put into a new dataset.
For this example, I will be using the auto dataset that comes with Stata:

sysuse auto

Let's say we are interested in the following regression:

regress price mpg trunk

It gives us the following output:

Source	SS	df	MS		
Model	141126459	2	70563229.4	Number of obs =	74
Residual	493938937	71	6956886.44	F(2, 71) =	10.14
				Prob > F =	0.0001
				R-squared =	0.2222
				Adj R-squared =	0.2003
Total	635065396	73	8699525.97	Root MSE =	2637.6

price	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
mpg	-220.1649	65.59262	-3.36	0.001	-350.9529	-89.3769
trunk	43.55851	88.71884	0.49	0.625	-133.3418	220.4589
_cons	10254.95	2349.084	4.37	0.000	5571.01	14938.89

We want to save the results of the coefficient, standard error, and p-value of the coefficient for the variable *mpg*.
The coefficient is stored as: "_b[mpg].
The standard error is stored as: "_se[mpg]
To create the stored result of the p-value requires that we run the following:

test mpg

```
. display r(p)
.00127069
```

The p-value for *mpg* is now stored as *r(p)*.
Now we can write out our basic commands to create a dataset that holds the SE, coefficient, and p-value of *mpg* in the regression above:

clear all
sysuse auto

```
tempname memhold
postfile `memhold' mpg_beta mpg_se mpg_p using "practiceregression.dta", ///
replace
regress price mpg trunk
local b=_b[mpg]
local se=_se[mpg]
test mpg
local p=r(p)
post `memhold' (`b') (`se') (`p')
postclose `memhold'
```

Tempname memhold creates the temporary object memhold that we use in the commands below.
Postfile creates that dataset "practiceregression.dta" with the variables *mpg_beta mpg_se mpg_p*.
Regress price mpg trunk is the regression that makes the stored results of the coefficients and standard errors available.
Local b creates a local macro equal to the coefficient of mpg in the above regression.
Local se creates a local macro equal to the standard error of mpg in the above regression.
Test mpg is run in order to make the stored result *r(p)*—i.e. the p-value for mpg—available.
Local p creates a local macro equal to the stored result *r(p)*—i.e. the p-value for mpg.
Post `memhold'… assigns the values of the local macros "b" "se" and "p" to the three variables specified in *postfile*.
Postclose notes the end of the additional of observations to the dataset.
Note: this new dataset will be saved to your current directory.
The new dataset should look like the following:

	mpg_beta	mpg_se	mpg_p
1	-220.1649	65.59262	.0012707

If we check with the above regression we can see that mpg_beta, mpg_se, and mpg_p correspond to the coefficient, se, and p-value for mpg.

Example/Application: Creating a dataset using a forval loop
Let us assume that we want to run regressions by groups based on the value of the variable *foreign*.
Let's take a look at the forval loop with the regression we will run:

```
forval j=0/1 {
        regress price mpg trunk if foreign==`j'
}
```

This runs two regressions of *mpg* and *trunk* on *price* using subsets of the original auto dataset based on whether or not the car was foreign. It produces the following outputs:
When not foreign (*foreign==0*):

Source	SS	df	MS	Number of obs =	52
				F(2, 49) =	8.42
Model	125089266	2	62544633	Prob > F =	0.0007
Residual	364105535	49	7430725.2	R-squared =	0.2557
				Adj R-squared =	0.2253
Total	489194801	51	9592054.92	Root MSE =	2725.9

price	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
mpg	-307.292	107.8128	-2.85	0.006	-523.95	-90.63405
trunk	36.35242	118.7538	0.31	0.761	-202.2924	274.9972
_cons	11628.88	3572.654	3.25	0.002	4449.364	18808.4

When the car is foreign (*foreign==1*):

Source	SS	df	MS	Number of obs =	22
				F(2, 19) =	7.30
Model	62721024.2	2	31360512.1	Prob > F =	0.0044
Residual	81642188.6	19	4296957.29	R-squared =	0.4345
				Adj R-squared =	0.3749
Total	144363213	21	6874438.7	Root MSE =	2072.9

price	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
mpg	-232.7421	70.27696	-3.31	0.004	-379.8335	-85.65075
trunk	158.6694	144.4289	1.10	0.286	-143.6237	460.9625
_cons	10340.07	2692.813	3.84	0.001	4703.943	15976.19

Now let's run our postfile commands with the forval loop:

```
clear all
sysuse auto
tempname memhold
postfile `memhold' cartype mpg_beta mpg_se using "practice.dta", replace
forval j=0/1 {
        regress price mpg trunk if foreign==`j'
        post `memhold' (`j') (_b[mpg]) (_se[mpg])
}
postclose `memhold'
```

Again, we start by creating a temporary object with the command *tempname*.
Postfile creates the empty dataset called "practice.dta." It has the empty variables "cartype" "mpg_beta" and "mpg_se" in it.
Forval j=0/1 is the start of our loop that tells the following commands to run through the loop from values 0 to 1.
Regress runs two regressions, one for when *foreign==0* and one for when *foreign==1*.

Post creates the observations corresponding to the variables "cartype" "mpg_beta" and "mpg_se" and stores them in these variables. You will notice that locals were not defined as they were in the previous example (*local b=_b[mpg]*). This shows us a simpler way of specifying the values for the coefficient and standard error of *mpg* without writing up a local for each: they can simply be written as (*_b[mpg]*) and (*_se[mpg]*) because these are already stored in memory after running the regression. If we wanted to store the p-value of *mpg* we would have to run *test mpg* and create a local for it.

Postclose marks the end of addition of observations to the dataset.

With any luck, your new dataset should look like this:

	cartype	mpg_beta	mpg_se
1	0	-307.292	107.8128
2	1	-232.7421	70.27696

This type of dataset is useful when you want to compare the regression results of many [more simple] regressions simultaneously. It is important to include a variable (in this case, *cartype*) that reminds you of the subset of the data you are looking at. Alternatively, if you did a large number of regressions in which the independent variable changes as a result of the loop you would want to include a variable in your postfile command line that reminds you of which independent variable is being run in that regression.

Additional resources and sources:

Data Today: http://datatoday.blogspot.com/2011/11/using-postfile-in-stata.html

Abromavicute's blog:
http://stataresources.blogspot.com/2014/03/introduction-to-p-postfile-in-stata.html

In-depth example of posting results of regressions:
http://www.michaelnormanmitchell.com/stow/posting-results-regression.html

An old syntax error (can be largely ignored):
http://www.stata.com/support/faqs/programming/post-command/

Unfortunately, the "help post" result is not particularly useful:
http://www.stata.com/help.cgi?post