stack — Stack data

Description

`stack` stacks the variables in *varlist* vertically, resulting in a dataset with variables *newvars* and `_N` $\cdot (N_v/N_n)$ observations, where N_v is the number of variables in *varlist* and N_n is the number in *newvars*. `stack` creates the new variable `_stack` identifying the groups.

Quick start

Replace data in memory with `v`, `v2` appended to `v1` and identify original variable by order in `_stack`

```
stack v1 v2, into(v)
```

Same as above, but with `v1` appended to `v2` and do not display warning that data in memory will be replaced

```
stack v2 v1, into(v) clear
```

Same as above, but save result in `v2`

```
stack v2 v1, group(2) clear
```

Append `v2` to `v1` and `v4` to `v3` and save result in `newv1` and `newv2`

```
stack v1 v3 v2 v4, into(newv1 newv2) clear
```

Same as above, but save results in `v1` and `v3`

```
stack v1 v3 v2 v4, group(2) clear
```

Menu

Data > Create or change data > Other variable-transformation commands > Stack data

Syntax

```
stack varlist [if] [in], { into(newvars) | group(#) } [options]
```

options	Description
Main	
* into(*newvars*)	identify names of new variables to be created
* group(#)	stack # groups of variables in *varlist*
clear	clear dataset from memory
wide	keep variables in *varlist* that are not specified in *newvars*

* Either into(*newvars*) or group(#) is required.

stack does not allow alias variables; see [D] **frunalias** for advice on how to get around this restriction.

Options

Main

into(*newvars*) identifies the names of the new variables to be created. into() may be specified using variable ranges (for example, into(v1-v3)). Either into() or group(), but not both, must be specified.

group(#) specifies the number of groups of variables in *varlist* to be stacked. The created variables will be named according to the first group in *varlist*. Either group() or into(), but not both, must be specified.

clear indicates that it is okay to clear the dataset in memory. If you do not specify this option, you will be asked to confirm your intentions.

wide includes any of the original variables in *varlist* that are not specified in *newvars* in the resulting data.

Remarks and examples

▷ Example 1: Illustrating the concept

This command is best understood by examples. We begin with artificial but informative examples and end with useful examples.

```
. use https://www.stata-press.com/data/r19/stackxmpl
. list
```

	a	b	c	d
1.	1	2	3	4
2.	5	6	7	8

```
. stack a b  c d, into(e f) clear
. list
```

	_stack	e	f
1.	1	1	2
2.	1	5	6
3.	2	3	4
4.	2	7	8

We formed the new variable `e` by stacking `a` and `c`, and we formed the new variable `f` by stacking `b` and `d`. `_stack` is automatically created and set equal to 1 for the first (`a`, `b`) group and equal to 2 for the second (`c`, `d`) group. (When `_stack`==1, the new data `e` and `f` contain the values from `a` and `b`. When `_stack`==2, `e` and `f` contain values from `c` and `d`.)

There are two groups because we specified four variables in the *varlist* and two variables in the `into` list, and $4/2 = 2$. If there were six variables in the *varlist*, there would be $6/2 = 3$ groups. If there were also three variables in the `into` list, there would be $6/3 = 2$ groups. Specifying six variables in the *varlist* and four variables in the `into` list would result in an error because $6/4$ is not an integer.

◁

▷ Example 2: Stacking a variable multiple times

Variables may be repeated in the *varlist*, and the *varlist* need not contain all the variables:

```
. use https://www.stata-press.com/data/r19/stackxmpl, clear
. list
```

	a	b	c	d
1.	1	2	3	4
2.	5	6	7	8

```
. stack a b  a c, into(a bc) clear
. list
```

	_stack	a	bc
1.	1	1	2
2.	1	5	6
3.	2	1	3
4.	2	5	7

`a` was stacked on `a` and called `a`, whereas `b` was stacked on `c` and called `bc`.

If we had wanted the resulting variables to be called simply `a` and `b`, we could have used

```
. stack  a b  a c, group(2) clear
```

which is equivalent to

```
. stack  a b  a c, into(a b) clear
```

◁

Example 3: Keeping the original variables

In this artificial but informative example, the `wide` option includes the variables in the original dataset that were specified in *varlist* in the output dataset:

```
. use https://www.stata-press.com/data/r19/stackxmpl, clear
. list
```

	a	b	c	d
1.	1	2	3	4
2.	5	6	7	8

```
. stack a b  c d, into(e f) clear wide
. list
```

	_stack	e	f	a	b	c	d
1.	1	1	2	1	2	.	.
2.	1	5	6	5	6	.	.
3.	2	3	4	.	.	3	4
4.	2	7	8	.	.	7	8

In addition to the stacked `e` and `f` variables, the original `a`, `b`, `c`, and `d` variables are included. They are set to missing where their values are not appropriate.

Example 4: Using wide with repeated variables

This is the last artificial example. When you specify the `wide` option and repeat the same variable name in both the *varlist* and the `into` list, the variable will contain the stacked values:

```
. use https://www.stata-press.com/data/r19/stackxmpl, clear
. list
```

	a	b	c	d
1.	1	2	3	4
2.	5	6	7	8

```
. stack  a b  a c, into(a bc) clear wide
. list
```

	_stack	a	bc	b	c
1.	1	1	2	2	.
2.	1	5	6	6	.
3.	2	1	3	.	3
4.	2	5	7	.	7

Example 5: Using stack to make graphs

We want one graph of y against x1 and y against x2. We might be tempted to type scatter y x1 x2, but that would graph y against x2 and x1 against x2. One solution is to type

```
. save mydata
. stack  y x1  y x2, into(yy x12) clear
. generate y1 = yy if _stack==1
. generate y2 = yy if _stack==2
. scatter y1 y2 x12
. use mydata, clear
```

The names yy and x12 are supposed to suggest the contents of the variables. yy contains (y,y), and x12 contains (x1,x2). We then make y1 defined at the x1 points but missing at the x2 points—graphing y1 against x12 is the same as graphing y against x1 in the original dataset. Similarly, y2 is defined at the x2 points but missing at x1—graphing y2 against x12 is the same as graphing y against x2 in the original dataset. Therefore, scatter y1 y2 x12 produces the desired graph.

◁

Example 6: Plotting cumulative distributions

We wish to graph y1 against x1 and y2 against x2 on the same graph. The logic is the same as above, but let's go through it. Perhaps we have constructed two cumulative distributions by using cumul (see [R] **cumul**):

```
. use https://www.stata-press.com/data/r19/citytemp
(City temperature data)
. cumul tempjan, gen(cjan)
. cumul tempjuly, gen(cjuly)
```

We want to graph both cumulatives in the same graph; that is, we want to graph cjan against tempjan and cjuly against tempjuly. Remember that we could graph the tempjan cumulative by typing

```
. scatter cjan tempjan, c(l) m(o) sort
```
(*output omitted*)

We can graph the tempjuly cumulative similarly. To obtain both on the same graph, we must stack the data:

```
. stack  cjuly tempjuly   cjan tempjan, into(c temp) clear
. generate cjan  = c if _stack==1
(958 missing values generated)
. generate cjuly = c if _stack==2
(958 missing values generated)
. scatter cjan cjuly temp, c(l l) m(o o) sort
```
(*output omitted*)

Alternatively, if we specify the wide option, we do not have to regenerate cjan and cjuly because they will be created automatically:

```
. use https://www.stata-press.com/data/r19/citytemp, clear
(City temperature data)
. cumul tempjan, gen(cjan)
. cumul tempjuly, gen(cjuly)
. stack  cjuly tempjuly   cjan tempjan, into(c temp) clear wide
. scatter cjan cjuly temp, c(l l) m(o o) sort
  (output omitted)
```

◁

❑ Technical note

There is a third way, not using the wide option, that is exceedingly tricky but is sometimes useful:

```
. use https://www.stata-press.com/data/r19/citytemp, clear
(City temperature data)
. cumul tempjan, gen(cjan)
. cumul tempjuly, gen(cjuly)
. stack cjuly tempjuly  cjan tempjan, into(c temp) clear
. sort _stack temp
. scatter c temp, c(L) m(o)
  (output omitted)
```

Note the use of connect's capital L rather than lowercase l option. c(L) connects points only from left to right; because the data are sorted by _stack temp, temp increases within the first group (cjuly vs. tempjuly) and then starts again for the second (cjan vs. tempjan); see [G-4] ***connectstyle***.

❑

Reference

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

Also see

[D] **contract** — Make dataset of frequencies and percentages

[D] **frunalias** — Change storage type of alias variables

[D] **reshape** — Convert data from wide to long form and vice versa

[D] **xpose** — Interchange observations and variables