I am having problems with the reshape command. Can you give further guidance?

Title	Problems with reshape
Author	Nicholas J. Cox, Durham University, UK

reshape is a very powerful command for restructuring your datasets. Like many other powerful commands, it can seem rather complicated until you have experience with it. Most questions about its use can be answered from a careful reading of the manual entry at **[D] reshape**, but some further notes and examples are added here.

For the specific problem of reshaping (e.g., pharmacokinetic) Latin-square and crossover data, also check out **pkshape**.

Is it really a problem for reshape?

reshape moves back and forth between what Stata calls wide data structures and long data structures. In a wide structure, rectangular blocks of data are held in several variables (i.e., several columns), whereas in a long data structure, such blocks are stretched out into single variables (one column for each block). In both cases, one or more identifier variables exist alongside—at a minimum, a one row identifier for a wide structure and a one row and one column identifier for a long structure.

Perhaps the most common kind of problem in practice is to **reshape** from wide to long. Your data may come compactly stored with information for each individual in one row (case, observation), but Stata shows a marked preference for long structures, especially when observations are repeated across time for each individual in a dataset. One good reason for this preference is that a long structure is more general and easier to handle, which is often true in many fields where the number of observations may vary from individual to individual.

If you want to transform from one wide or block structure to another, this can often be done with two applications of **reshape**, and examples are given below. However, make sure first that your problem is not better handled by **xpose**, which transposes the dataset so that observations become variables, and vice versa.

> **Note on terminology**: What are here called wide and long structures are often called wide and long formats, terms that are generally clear but have the disadvantage that "format" is an overloaded word in computing (compare display formats, file formats, and so forth). I commend the suggestion of Clyde Schechter to use the clear and simple terms wide and long "layouts" if "structure" or "format" do not seem to be appropriate words.

A key point is that in reshaping from wide to long, **reshape** expects to find one or more groups of variables so that names in each group all begin with the same stubname. Thus the variable names **inc60**, **inc70**, **inc80**, and **inc90** all share the stubname **inc**.

The syntax diagram for the command in the manual and in the online help gives patterns for the so-called basic syntax (setting aside the options) of

```
reshape wide stubnames
reshape long stubnames
```

Thus, in the example above, the stubname would be **inc**. If we also have variables **pop60**, **pop70**, **pop80**, and **pop90**, then **pop** would be an extra stubname.

Suffixes should be numeric unless specified otherwise with string

Moreover, **reshape** expects what follows the stub—the suffix—will be numeric, unless you specify otherwise with the **string** option. Any nonnumeric characters may mess things up if you overlook this rule, including not only letters but also the **_** (underscore), which you might be using in your variable names. Suppose you had variables **pop90f** and **pop90m** for male and female population. You have three choices given the presence of the letters **f** and **m** in the suffixes:

- specify **pop** as stub together with **string**
- rename your variables, say, to **mpop90** and **fpop90**, or whatever makes sense
- use the @ character, as explained in the manual.

On occasion, people use numeric suffixes with leading zeros, such as **01**, **02**, and so forth. **reshape** will understand these properly only if they are declared as **string**.

Whatever you do, your choice of stubname will determine how the data can be restructured.

The following example is based on an exchange on Statalist.

Question

> If I have many variables all occurring in pairs for two years 1997 and 1998, so that the dataset looks like **A97**, **A98**, **B97**, **B98**, and so on, is there any easy way to **reshape** the data to **long** without typing all the stub names?

Answer

The variable names are collectively ***97 *98**, so we need a way of expanding that list of wildcard names automatically and then removing the suffix. We can work on either ***97** or ***98**.

unab is usually billed as a programmer's command, but it can be used interactively. It unabbreviates a varlist and puts the result in a local macro.

```
. unab vars : *97
```

```
. local stubs : subinstr local vars "97" "", all
```

In other words, each occurrence of "**97**" is replaced by an empty string; that is, they are removed. See **macro**.

Then we can

```
. reshape long `stubs', options
```

You may need to rename some variable sets before running reshape

We have seen data in which monthly temperatures were stored as variables **Jan** through **Dec**. Although the names are logical and clear, **reshape** can do nothing with such a variable set until these variables are renamed to a stub plus suffix form. In Stata 12 and up,

```
. rename (Jan-Dec) temp#, addnumber
```

would be one solution. Here we suppose that **Jan** through **Dec** are stored in that order in your dataset. (Users of earlier versions of Stata may find the community-contributed program **renvars** helpful for such problems: type **search renvars** to find download locations.)

Column identifiers split between variables must be combined

The help for **reshape** explains that **i()** can specify one or more variables. However, this is not true of **j()**, which can specify only one variable. If your column identifier is split between variables, you need to put them together in a single variable. The easiest solution, especially if the variables are categorical or discrete, is to use **egen, concat()** or **egen, group()**. See **egen**.

The following example is based on an exchange on Statalist.

Question

I have an identifier that is split between variables. This situation is common in ANOVA where treatment cells may be identified by more than one factor. I have the following data:

```
. list, sep(6)

     +---------------------------------------+
     |   level      delay   animal       peak |
     |---------------------------------------|
  1. |       0         50   1_1_0F     773.75 |
  2. |       0        100   1_1_0F    1001.63 |
  3. |      75         50   1_1_0F      472.5 |
  4. |      75        100   1_1_0F    927.875 |
  5. |      85         50   1_1_0F    611.375 |
```

```
      7. |        0          50    1_1_1F    1116.88 |
      8. |        0         100    1_1_1F    1101.38 |
      9. |       75          50    1_1_1F    544.875 |
     10. |       75         100    1_1_1F    567.875 |
     11. |       85          50    1_1_1F    443.875 |
     12. |       85         100    1_1_1F        466 |
         +-----------------------------------------+
```

The first two variables, **level** and **delay**, define treatment conditions for which **peak** was measured for each **animal**. To calculate a ratio from the conditions within each animal, I would like to **reshape** this dataset to one that looks like **animal peak0_50 peak0_100 peak75_50 peak75_100 peak85_50 peak85_100**.

Answer

```
. egen treatment = concat(level delay), p(_)
. drop level delay
. reshape wide peak, i(animal) j(treatment) string
```

The factors **level** and **delay** take integer values, so there is no difficulty in concatenating them into a string variable. Should we later desire the original data structure, we can type

```
. reshape long
. split treatment, p(_) destring
. rename treatment1 level
. rename treatment2 delay
```

split is, broadly speaking, the inverse of **egen, concat()**. See **split**.

You may need two reshapes to get to where you want to be

A traveller asks, “How do I get to X from here?”, and a local replies, “If I was going to X, I wouldn't start from here”. Many reshaping problems provoke this feeling. One broad strategic comment is to underline a point exemplified at **[D] reshape**: although **reshape** apparently offers only wide to long and long to wide reshapes, either a **reshape long** followed by a **reshape wide**, or the reverse, may be a good way of solving many problems, possibly with some manipulations in between. Having said that, some users ask for ways of getting from a wide structure to another wide structure. The best advice is often to question why they really want to do that, as Stata generally offers many more ways of working with long data structures than with wide data structures. The long-term view is that you are likely to be better off with a long structure.

Another possibility sometimes encountered is the need for **reshape long** followed by **reshape long**, a mapping to what may be called a "long long" structure. An example is given later.

stubname **x** and generate an identifier variable if it does not exist:

```
. generate id = _n
. reshape long x, i(id) j(varno)
. egen rankx = rank(x), by(id)
. reshape wide
```

We are back where we started but with the extra variable **rankx**.

The following examples are based on exchanges on Statalist.

Question

I have a question that deals with rearranging a dataset. My variables are electoral data: **v1**, **v2**, **v3** ... are vote percentages for parties, **s1**, **s2**, **s3** ... are seat percentages for parties, and **p1**, **p2**, **p3** ... are names of parties.

The observations are single elections in given countries. I would like to sort the variables so that **v1** contains the highest vote percentage for each election, **v2** the second highest, and so on. Above all, I have to sort the **s*** *and* **p*** accordingly, so that I am still able to match vote percentages, seat percentages, and party names.

Answer

This solution is easy with a **reshape** back and forth, but perhaps not otherwise.

First, as usual, generate an identifier variable if you do not have one:

```
. gen id = _n
```

Now **reshape** to long

```
. reshape long v s p , i(id) j(order)
```

and then sort to put highest vote first within each election:

```
. gsort id -v
```

Now recalculate rankings

```
. by id : replace order = _n
```

and **reshape** back to wide again:

Question

I have a dataset with multiple observations for each identifier **id** according to a second variable **pos**. I want to gather information by **id** and **pos** on a set of other variables **co***. Here is an example:

```
          id      pos      co1      co2      co3
   1.      1        1       56       86       65
   2.      1        1       44       55       66
   3.      1        2       33        .        .
```

What I want is to gather all the data in **co*** by **id** and **pos** into a new dataset:

```
          id      pos      co1      co2      co3      co4      co5      co6
   1.      1        1       56       86       65       44       55       66
   2.      1        2       33        .        .        .        .        .
```

Answer

From this example, neither **id** nor **id** and **pos** combined identify observations uniquely, so we need to produce a new identifier. We can then **reshape** to long:

```
. gen ID = _n
. reshape long co, i(ID) j(no)
```

It is the combinations of **id** and **pos** that define new observations in the data structure we want, so we

```
. egen group = group(id pos)

. list

     +---------------------------------+
     | ID   no   id   pos   co   group |
     |---------------------------------|
  1. |  1    1    1     1   56       1 |
  2. |  1    2    1     1   86       1 |
  3. |  1    3    1     1   65       1 |
  4. |  2    1    1     1   44       1 |
  5. |  2    2    1     1   55       1 |
```

```
     7. |  3     1     1      2    33        2 |
     8. |  3     2     1      2     .        2 |
     9. |  3     3     1      2     .        2 |
        +---------------------------------------+
```

Now let's get the column numbers we want and **reshape** again:

```
. by group, sort: replace no = _n
. drop ID
. reshape wide co, i(group) j(no)
. drop group
. list

    +-----------------------------------------------------+
    | co1   co2   co3   co4   co5   co6   id   pos |
    |-----------------------------------------------------|
 1. |  56    86    65    44    55    66    1     1 |
 2. |  33     .     .     .     .     .    1     2 |
    +-----------------------------------------------------+
```

Think of **reshape** of being like a Fourier transform, or even a logarithm: transform, do some stuff, and transform again.

Question

I have a dataset on which services are provided and which counties are covered by various agencies. Variables **ES1-ES10** are 1 or 0 if a particular service is or is not provided by an agency, and variables **A1-A5** are 1 or 0 if a particular county is or is not covered by an agency. I want to **tabulate** service by county to show how many providers of each service operate in each county.

Answer

What is important here—and what can easily be missed, as we did when first trying to understand this question—is that the two stubs **ES** and **A** are not on the same footing. The dataset is a three-way array, agencies X service X counties, even though each agency is taken to offer the same mix of services in the counties it covers. (If that were not true, then the data structure we start from would be misrepresenting reality.) Thus the reshaping is another double **reshape**, but this time **long** followed by **long**.

Assuming an identifier variable **id**, which, as always, we can easily create if absent, we should first **preserve** because we shall probably want to return to the dataset in its present form:

```
. preserve
```

```
. reshape long ES, i(id) j(service)
. reshape long A, i(id service) j(county)
```

The “long long” data structure has **ES** equal to 1 and **A** equal to 1 when (a) a service is provided by an agency and (b) a particular county is covered; thus we count only observations for which both (a) and (b) are true, which we can specify via the weights for the table:

```
. tabulate service county [w = ES * A]
```

If desired to return to the original data,

```
. restore
```

Acknowledgment

David Airey made helpful suggestions.