

## Title

**merge** — Merge datasets

## Syntax

```
merge [varlist] using filename [filename ...] [, options]
```

<i>options</i>	description
----------------	-------------

---

### Options

<b>keep</b> ( <i>varlist</i> )	keep only the specified variables from data in <i>filename</i>
<b>_merge</b> ( <i>newvar</i> )	<i>newvar</i> marks source of resulting observation; default is <b>_merge</b>
<b>no</b> label	do not copy value label definitions from <i>filename</i>
<b>no</b> notes	do not copy notes from <i>filename</i>
<b>update</b>	replace missing data in memory with data from <i>filename</i>
<b>replace</b>	replace nonmissing data in memory with data from <i>filename</i>
<b>no</b> keep	drop obs. in using dataset that do not match
<b>no</b> summary	drop summary variables when multiple <i>filenames</i> are specified
* <b>unique</b>	match variables uniquely identify observations in both data in memory and in <i>filename</i>
* <b>uniquemaster</b>	match variables uniquely identify observations in memory
* <b>uniquising</b>	match variables uniquely identify observations in <i>filename</i>
* <b>sort</b>	sort master and using datasets by match variables before merge

---

\* **unique**, **uniquemaster**, **uniquising**, and **sort** require *varlist* (the match variables) be specified.

**sort** implies **unique**.

If *filename* is specified without an extension, **.dta** is assumed.

## Description

**merge** joins corresponding observations from the dataset currently in memory (called the *master* dataset) with those from Stata-format datasets stored as *filename* (called the *using* datasets) into single observations.

## Options

### Options

---

**keep**(*varlist*) specifies the variables to be kept from the **using** dataset. If **keep**() is not specified, all variables are kept.

The *varlist* in **keep**(*varlist*) differs from standard Stata varlists in two ways: variable names in *varlist* may not be abbreviated, except by the use of wildcard characters; and you may not refer to a range of variables, such as **price-weight**.

**\_merge**(*newvar*) specifies the name of the variable to be created that will mark the source of the resulting observation. The default is **\_merge(\_merge)**; that is, if you do not specify this option, the new variable will be named **\_merge**. See *The two kinds of merges* below for details.

`nolabel` prevents Stata from copying the value label definitions from the using dataset into the result.

Even if you do not specify this option, label definitions from the using dataset do not replace label definitions in the master dataset.

`nonotes` prevents `notes` in the using dataset from being incorporated into the result. The default is to incorporate notes from the using dataset that do not already appear in the master dataset.

`update` specifies that the values from the using dataset be retained in cases where the master dataset contains missing. By default, the master dataset is held inviolate—values from the master dataset are retained when variables are found in both datasets.

`replace`, allowed with `update` only, specifies that even when the master dataset contains nonmissing values, they are to be replaced with corresponding values from the using dataset when the corresponding values are not equal. A nonmissing value, however, will never be replaced with a missing value.

`nokeep` causes `merge` to ignore observations in the using dataset that have no corresponding observation in the master. The default is to add these observations to the merged result and mark such observations with `_merge = 2`.

`nosummary` causes `merge` to drop the summary variables created when multiple using datasets are specified. The default is to create `_merge1` recording results from merging the first disk dataset, `_merge2` recording results from merging the second disk dataset, and so on. `_merge1`, `_merge2`, . . . , contain 1 if an observation was found in the respective disk dataset and 0 otherwise.

Whether or not `nosummary` is specified, overall status variable `_merge` is created.

`unique`, `uniqmaster`, and `uniquising` specify that the match variables in a match-merge uniquely identify the observations. (See *Match-merge* below.)

`unique` specifies that the match variables uniquely identify the observations in the master dataset and in the using dataset. For most match-merges, you should specify `unique`. `merge` does nothing differently when you specify the option, unless the assumption you are making is false. In that case, an error message is issued, and the data are not merged.

`uniqmaster` specifies that the match variables uniquely identify the observations in memory, in the master dataset, but not necessarily the ones in the using dataset.

`uniquising` specifies that the match variables uniquely identify the observations in the using dataset, but not necessarily the ones in the master data.

`unique` is thus equivalent to specifying `uniqmaster` and `uniquising`.

Things are more complicated when multiple using datasets are specified. `unique` still means unique in all datasets, and `uniquising` still means unique in each of the using datasets, just as you would expect, but `uniqmaster` takes on a whole new meaning: `uniqmaster` means unique in the master and in all using datasets except the last! It asserts that the match variables uniquely identify observations in the master at each step, meaning when the master is merged with the first using dataset, then when the (new) master (equal to original plus first using) is merged with the second using dataset, and so on. In summary, `uniqmaster` is simply not useful when multiple using datasets are specified.

If none of the three unique options are specified, observations in neither the master nor the using datasets are required to be unique, although they could be. If they are not unique, records that have the same values of the match variables are joined by observation until all the records on one side or the other are matched; after that, the final record on the shorter side is duplicated over and over again to match with the remaining records needing to be matched on the longer side.

`sort` specifies that the master and using datasets be sorted by the match variables, before the datasets are merged, if they are not already sorted by them. Match variables are required with `sort`.

## Remarks

Remarks are presented under the headings

*The two kinds of merges*  
*One-to-one merge*  
*Match-merge*  
*Match-merging with multiple using datasets*  
*Updating data*

Distinguish carefully between merging and appending datasets and the corresponding Stata commands `merge` and `append`. Appending refers to the addition of new observations on existing variables. If you think of a dataset as a rectangle with observations going down and variables going across, appending increases the dataset's length. Merging adds new variables to existing observations, increasing the dataset's width. See [U] **22 Combining datasets** for more information.

Say that you have a dataset in which each observation records the characteristics of a particular automobile, such as the car's price, weight, etc. If you have two such datasets, one for domestic and another for imported cars, and you wish to combine them into a single dataset, see [D] **append**.

On the other hand, if you have two datasets on the same types of cars, one recording price and the other weight, mileage, etc., and you wish to combine them into a single set, continue reading; `merge` does this.

Another command, `joinby`, forms all pairwise combinations of observations within group. Say that you have one dataset on mothers and fathers and another on their children. If you wish to combine them to match each parent with every one of their children (each child is matched with both parents) so that a two-parent, three-child family results in  $2 \times 3 = 6$  observations, see [D] **joinby**.

## The two kinds of merges

`merge` joins the observations stored in memory with the observations stored in *filename*. The disk dataset must be a Stata-format dataset; that is, it must have been created with the `save` command.

Stata performs two kinds of merges. If no *varlist* is specified, Stata performs a *one-to-one* merge, meaning that the first observation of one dataset is joined with the first observation of the other dataset, the second observation is joined with the second, and so on. If a *varlist* is specified, however, Stata uses those variables to perform a *match* merge, in which observations are joined only if the values of the variables in the specified *varlist* match.

Regardless of the style of merge being performed, `merge` always adds a new variable called (by default) `_merge` to the dataset. This variable takes on the values 1, 2, or 3 to mark the source of the resulting observation. The coding is as follows:

1. The observation occurred only in the master dataset.
2. The observation occurred only in the using dataset.
3. The observation is the result of joining an observation from the master dataset with one from the using dataset.

When you use the `update` option, this coding is extended to include

4. The same as 3, except that missing values in the master dataset were updated with values from the using dataset.

5. The same as 3, except that some values in the master dataset disagree with values in the using dataset.

## One-to-one merge

In a one-to-one merge, the first observation in the master dataset is joined with the first observation in the using dataset, the second observation is joined with the second, and so on. If variables with the same name occur in both the master and the using datasets, the joined observation retains those variables' *original* values—the values of the variables in the master dataset. When the master and using datasets contain different numbers of observations, missing values are joined with the remaining observations from the longer dataset.

### ► Example 1

We have two datasets stored on disk that we wish to merge into a single dataset. The first dataset, `odd.dta`, contains the first five positive odd numbers. The second dataset, `even1.dta`, contains the fifth through eighth positive even numbers. (Our example is admittedly not realistic, but it does illustrate the concept.) The datasets are

```
. use odd, clear
(First five odd numbers)
. list
```

	number	odd
1.	1	1
2.	2	3
3.	3	5
4.	4	7
5.	5	9

```
. use even1, clear
(5th through 8th even numbers)
. list
```

	number	even
1.	5	10
2.	6	12
3.	7	14
4.	8	16

We will join these two datasets using a one-to-one merge. Since the even dataset is already in memory (we just used it above), we type `merge using odd`. The result is

```
. merge using odd
number was int now float
. list
```

	number	even	odd	_merge
1.	5	10	1	3
2.	6	12	3	3
3.	7	14	5	3
4.	8	16	7	3
5.	5	.	9	2

Notice the new variable `_merge`. Every time Stata merges two datasets, it creates this variable and assigns a value of 1, 2, or 3 to each observation. The value 1 indicates that the resulting observation occurred only in the master dataset, 2 indicates that the observation occurred only in the using dataset, and 3 indicates that the observation occurred in both datasets and is thus the result of joining an observation from the master dataset with an observation from the using dataset.

In this case, the first four observations are marked by `_merge` with a value of 3, and the last observation by `_merge` with a value of 2. The first four observations are the result of joining observations from the two datasets, and the last observation is a result of adding a new observation from the using dataset. These values reflect the fact that the original dataset in memory had four observations, and the odd dataset stored on disk had five observations. The new observation is from the odd dataset exclusively: `number` is 5, `odd` is 9, and `even` has been filled in with *missing*.

Notice that `number` takes on the values 5 through 8 for the first four observations. Those are the values of `number` from the original dataset in memory—the even dataset—and conflict with the value of `number` stored in the first four observations of the odd dataset. `number` in that dataset took on the values 1 through 4, and those values were lost during the merge process. When Stata joins observations and there is a conflict between the value of a variable in memory and the value stored in the using dataset, Stata retains the value stored in memory by default.

When the command `merge using odd` was issued, Stata responded with “number was int now float”. Let’s describe the datasets in this example:

```
. describe using odd
Contains data                      First five odd numbers
  obs:                            5                      2 Jan 2005 23:01
  vars:                            2
  size:                            60
```

---

variable name	storage type	display format	value label	variable label
number	float	%9.0g		
odd	float	%9.0g		Odd numbers

---

```
Sorted by:
. describe using even1
Contains data                      5th through 8th even numbers
  obs:                            4                      8 Jan 2005 09:20
  vars:                            2
  size:                            40
```

---

variable name	storage type	display format	value label	variable label
number	int	%8.0g		
even	float	%9.0g		Even numbers

---

```
Sorted by:
```

Note that `number` is stored as a `float` in `odd.dta` but as an `int` in `even1.dta`; see [U] **12.2.2 Numeric storage types**. When we `merge` two datasets, Stata engages in automatic variable promotion; that is, if there are conflicts in numeric storage types, the more precise storage type will be used. The resulting dataset, therefore, will have `number` stored as a `float`, and Stata told us this when it said “number was int now float”.

## Match-merge

In a match-merge, observations are joined if the values of the variables in the *varlist* are the same. Since the values must be the same, obviously the variables in the *varlist* must appear in both the master and the using datasets.

A match-merge proceeds by taking an observation from the master dataset and one from the using dataset and comparing the values of the variables in the *varlist*. If the *varlist* values match, the observations are joined. If the *varlist* values do not match, the observation from the *earlier* dataset (the dataset whose *varlist* value comes first in the sort order) is joined with a pseudo-observation from the *later* dataset. All the variables in the pseudo-observation contain missing values. The actual observation from the later dataset is retained and compared with the next observation in the earlier dataset, and the process repeats.

### ► Example 2

The result is not nearly as incomprehensible as the explanation. Let's return to the dataset used in example 1 and merge the two datasets on variable *number*. We first use the even dataset and then type `merge number using odd`:

```
. use even1, clear
(5th through 8th even numbers)
. merge number using odd
master data not sorted
r(5);
```

Instead of merging the datasets, Stata reports the error message “master data not sorted”. Match-merges require that the data be sorted in the order of the *varlist*, which in this case means ascending order of *number*. In the previous example, the data are in such an order, so the message is more than a little confusing. Before Stata can merge two datasets, however, both datasets must be sorted, and Stata must *know* that the data are sorted.

The basis of Stata's knowledge is the internal information it keeps on the sort order, and Stata reveals the extent of its knowledge whenever we describe the dataset:

```
. describe
Contains data from even1.dta
  obs:          4                      5th through 8th even numbers
  vars:         2                      8 Jan 2005 09:20
  size:        40 (99.9% of memory free)
```

variable name	storage type	display format	value label	variable label
number	int	%8.0g		
even	float	%9.0g		Even numbers

```
Sorted by:
```

The last line of the description shows that the data are “Sorted by:” nothing. As far as Stata can tell, the data is not sorted. Similarly, `odd.dta` is sorted in ascending order of *number*, but as before, typing `describe` shows that Stata does not know this.

If the datasets are not sorted on the merge variable, we need to use the `sort` option with `merge` to tell Stata to sort both datasets before doing the merge:

```
. merge number using odd, sort
number was int now float
```

```
. list
```

	number	even	odd	_merge
1.	5	10	9	3
2.	6	12	.	1
3.	7	14	.	1
4.	8	16	.	1
5.	1	.	1	2
6.	2	.	3	2
7.	3	.	5	2
8.	4	.	7	2

It worked! Let's understand what happened. Even though Stata sorted both datasets by `number` since we specified the `sort` option, we immediately discern that the result is no longer in ascending order of `number`. It will be easier to understand what happened if we re-sort the data and then `list` the data again:

```
. sort number
```

```
. list
```

	number	even	odd	_merge
1.	1	.	1	2
2.	2	.	3	2
3.	3	.	5	2
4.	4	.	7	2
5.	5	10	9	3
6.	6	12	.	1
7.	7	14	.	1
8.	8	16	.	1

Note that `number` now goes from 1 to 8, with no repeated values and no values left out of the sequence. Recall that the `odd` dataset defined observations for `number` between 1 and 5, whereas the `even` dataset defined observations between 5 and 8. Thus the variable `odd` is defined for `number` variables equal to 1 through 5, and `even` is defined for `number` variables equal to 5 through 8.

For instance, in the first observation, `number` is 1, `even` is *missing*, and `odd` is 1. The value of `_merge`, 2, indicates that this observation came from the using dataset—`odd.dta`. In the last observation, `number` is 8, `even` is 16, and `odd` is *missing*. The value of `_merge`, 1, indicates that this observation came from the master dataset—`even1.dta`.

The fifth observation is worth comment. `number` is 5, `even` is 10, and `odd` is 9. Both `even` and `odd` are defined since both the `even` and the `odd` datasets had information for `number` with a value of 5. The value of `_merge`, 3, also tells us that both datasets contributed to the formation of the observation.

◀

### ▷ Example 3

Although the previous example demonstrated, in glorious detail, how the match-merging process works, it was not a practical example of how you will ordinarily employ it. Here is a more realistic application.

We have two datasets containing information on automobiles. The identifying variable in each dataset is `make`, a string variable containing the manufacturer and the model. An *identifying* variable is one that is unique for every observation in the dataset. Values for `make`—for instance, Honda Accord—are sufficient for identifying each observation.

One dataset, `autotech.dta`, also contains `mpg`, `weight`, and `length`. The other dataset, `autocost.dta`, contains `price` and `rep78`, the 1978 repair record.

```
. describe using http://www.stata-press.com/data/r9/autotech
Contains data                               1978 Automobile Data
  obs:                74                      8 Jan 2005 11:19
  vars:                4
  size:               2,072
```

---

variable name	storage type	display format	value label	variable label
<code>make</code>	str18	%18s		Make and Model
<code>mpg</code>	int	%8.0g		Mileage (mpg)
<code>weight</code>	int	%8.0g		Weight (lbs.)
<code>length</code>	int	%8.0g		Length (in.)

---

```
Sorted by:  make

. describe using http://www.stata-press.com/data/r9/autocost
Contains data                               1978 Automobile Data
  obs:                74                      8 Jan 2005 08:20
  vars:                3
  size:               1,924
```

---

variable name	storage type	display format	value label	variable label
<code>make</code>	str18	%18s		Make and Model
<code>price</code>	int	%8.0g		Price
<code>rep78</code>	int	%8.0g		Repair Record 1978

---

```
Sorted by:  make
```

We want to merge these two datasets into a single dataset:

```
. use http://www.stata-press.com/data/r9/autotech, clear
(1978 Automobile Data)
. merge make using http://www.stata-press.com/data/r9/autocost
```

Let's now examine the result:

```
. describe
Contains data from http://www.stata-press.com/data/r9/autotech.dta
  obs:          74                1978 Automobile Data
  vars:         7                 8 Jan 2005 11:19
  size:        2,442 (99.8% of memory free)
```

variable name	storage type	display format	value label	variable label
make	str18	%18s		Make and Model
mpg	int	%8.0g		Mileage (mpg)
weight	int	%8.0g		Weight (lbs.)
length	int	%8.0g		Length (in.)
price	int	%8.0g		Price
rep78	int	%8.0g		Repair Record 1978
_merge	byte	%8.0g		

Sorted by:

Note: dataset has changed since last saved

We now have a single dataset containing all the information from the two original datasets—or at least it appears that we do. We need to verify the result. We think that we entered data for the same cars in each dataset, so every variable should be defined for every car. It is possible that we made a mistake and accidentally left some cars out of one or the other dataset. We can reassure ourselves of our infallibility by tabulating `_merge`:

```
. tabulate _merge
```

_merge	Freq.	Percent	Cum.
3	74	100.00	100.00
Total	74	100.00	

We see that `_merge` is 3 for every observation in the dataset. We made no mistake—for every observation in `autocost.dta`, there is an observation in `autotech.dta`, and vice versa.

Now pretend that we have another dataset containing additional information on these automobiles, `automore.dta`, which we want to merge as well. We use the `sort` option, since after a `merge` the sort order may have changed:

```
. merge make using automore, sort
_merge already defined
r(110);
```

Stata refused to merge the new dataset, complaining instead that `_merge` is already defined. Every time Stata merges datasets, it wants to create a variable called `_merge` (or `varname` if the `_merge(varname)` option was specified). In this case, there is an `_merge` variable left over from the last time we merged. We have three choices: We can rename the `_merge` variable, we can drop it, or we can specify a different variable name with the `_merge()` option. In this case, `_merge` contains no useful information—we have already verified that the previous merge went as expected—so we drop it and try again:

```
. drop _merge
. merge make using automore, sort
```

Stata performed our request; whatever new variables were contained in `automore.dta` should be contained in our single master dataset. After a match-merge, we should *always* tabulate `_merge` to verify that the expected actually happened, as we do below:

```
. tabulate _merge
```

<code>_merge</code>	Freq.	Percent	Cum.
1	1	1.33	1.33
2	1	1.33	2.67
3	73	97.33	100.00
Total	75	100.00	

Surprise! In this case, something strange did happen. Some 73 of the observations merged as we anticipated. However, the new dataset `automore.dta` added one new car to the dataset (identified by `_merge` equal to 2) and failed to define new variables for another car in our original dataset (identified by `_merge` equal to 1). Most likely, we have a mistake in `automore.dta`. We probably misidentified one car so that, to Stata, it appeared as data on a new car, resulting in one new observation and missing data on another.

If this happened to us, we could figure out why it happened. We could type `list make if _merge==1` to learn the identity of the car that did not appear in `automore.dta`, and we could type `list make if _merge==2` to learn the identity of the car that `automore.dta` added to our data. ◀

## □ Technical Note

It is difficult to overemphasize the importance of tabulating `_merge`, no matter how sure you are that you have no errors. It takes only a second and can save you hours of grief. Along the same lines, one-to-one merges are a bad idea. In the example above, we could have performed all the merges as one-to-one merges and saved a small amount of typing. Let's examine what would have happened.

We first merged `autotech.dta` with `autocost.dta` by typing `merge make using autocost`. We could perform a one-to-one merge by typing `merge using autocost`. The result would be the same; the datasets line up and are in the same sort order, so sequentially matching the observations from the two datasets would have resulted in a perfectly matched dataset.

In the second case, we merged the data in memory with `automore.dta` by typing `merge make using automore`. A one-to-one merge would have led to disaster, and we would never have known it! If we had typed `merge using automore`, Stata would have sequentially, and blindly, joined observations. Since there are the same number of observations in each dataset, everything would have appeared to merge perfectly.

We speculated in example 3 that we had an error in `automore.dta`. Remember that `automore.dta` included data on one new car and lacked data on an existing car. Even if there is no error, things have gone awry. No matter what, the data in memory and `automore.dta` do not match. For instance, assume that this new car is the first observation of `automore.dta`, and that it is some (perhaps mistaken) model of Ford. Assume that the first observation of the data in memory is on a Chevrolet. Stata could and would silently join data on the Chevrolet with data on the Ford, and thereafter, data on a Volvo with data on a Saab, and even data on a Volkswagen with data on a Cadillac, and you would never know.

Every dataset should carry a variable or a set of variables that *uniquely* identifies each observation, and you should always use those variables when merging data. Ignore this advice at your own peril. □

### □ Technical Note

You may need to merge two datasets knowing that there will be mismatches. For example, say that you have an analysis dataset on patients from the cancer ward of a particular hospital, and you have just received another dataset containing their demographic information. Actually, this other dataset contains not just their demographic information, but also the demographic information on every patient in the hospital during the year. You could

```
. merge patid using demog
. drop if _merge==2
```

or

```
. merge patid using demog, nokeep
```

The `nokeep` option tells `merge` not to store observations from the using data that do not appear in the master. There is an advantage in this. When we merged and dropped, we stored the irrelevant observations and then discarded them, so the data in memory temporarily grew. When we merge with the `nokeep` option, the data never grow beyond what is absolutely necessary. □

### □ Technical Note

In our automobile example, we had a single identifying variable. Sometimes there will be multiple identifying variables that, taken together, are unique for every observation.

Let's imagine that, rather than having a single variable called `make`, we had two variables: `manuf` and `model`. `manuf` contains the manufacturer, and `model` contains the model. Rather than having a single variable recording, say, "Honda Accord", we have two variables, one recording "Honda" and another recording "Accord". Stata can deal with this type of data. We could go back through our previous example and substitute `manuf model` everywhere we see `make`. For instance, rather than typing `merge make using autcost`, we would have typed `merge manuf model using autcost`.

Now let's make one more change in our assumptions. Let's assume that `manuf` and `model` are not string variables but are instead numerically coded variables. Perhaps the number 15 stands for Honda in the `manuf` variable, and the number 2 stands for Accord in the `model` variable. We do not have to remember our numeric codes because we have smartly created value labels telling Stata what number stands for what string of characters. We now go back to the step when we merged `autotech.dta` with `autocost.dta`:

```
. use autotech, clear
(1978 Automobile Data)
. merge manuf model using autcost
(label manuf already defined)
(label model already defined)
```

Stata makes two minor comments but otherwise carries out our request. It notes that the labels `manuf` and `model` are already defined. The messages refer to the *value labels* named `manuf` and `model`.

Both datasets contain value label definitions that turn the numeric codes for manufacturer and model into words. When Stata merged the two datasets, it already had one set of definitions in memory (obtained when we typed `use autotech`) and thus ignored the second set of definitions contained in `autocost.dta`. Stata felt obliged to mention the second set of definitions while otherwise ignoring them, since they *might* contain different codings. In this case, we know that they are the same since we created them. (*Hint*: You should never give the same name to value labels containing different codings.) □

## ▷ Example 4

We might want the match-merge to take place only if the match variable(s) uniquely identify the observations in both datasets. We can make this condition a requirement for `merge` by specifying the `unique` option. For example, suppose that we wanted to merge `autotech.dta` and `autocost.dta`, with `make` as the match variable. Now suppose that `autotech.dta` has two observations in which the value of `make` is “Dodge Colt”. If we specify the `unique` option, `merge` will refuse to execute the merge, and Stata will display a message telling us why.

```
. use http://www.stata-press.com/data/r9/autotech, clear
(1978 Automobile Data)
. generate fweight = 2 if make == "Dodge Colt"
(73 missing values generated)
. expand fweight
(73 missing counts ignored; observations not deleted)
(1 observation created)
. merge make using http://www.stata-press.com/data/r9/autocost, unique sort
variable make does not uniquely identify observations in the master data
r(459);
```

To find out which observations have duplicate values in the match variables, we use the `duplicates` command (see [D] **duplicates**).

```
. duplicates list make
Duplicates in terms of make
```

obs:	make
27	Dodge Colt
28	Dodge Colt

```
. drop in 28
(1 observation deleted)
```

Once we have resolved the duplicates problem, we can successfully merge the two datasets.

```
. merge make using http://www.stata-press.com/data/r9/autocost, unique sort
```

If your only criterion is that the match variable be unique in the master dataset or using dataset, use `uniquemaster` or `uniquusing`, respectively, rather than `unique`.

◀

In a match-merge, the master and using datasets may have multiple observations with the same *varlist* value. These multiple observations are joined sequentially, as in a one-to-one merge. If the datasets have an unequal number of observations with the same *varlist* value, the last such observation in the *shorter* dataset is replicated until the number of observations is equal.

## ▷ Example 5

The process of replicating the observation from the shorter dataset is known as *spreading* and can be put to practical use. Suppose that we have two datasets. `dollars.dta` contains the dollar sales and costs of our firm, by region, for the last year:

```
. use dollars, clear
(Regional Sales & Costs)
. list
```

	region	sales	cost
1.	NE	360,523	138,097
2.	N Cntrl	419,472	227,677
3.	South	532,399	330,499
4.	West	310,565	165,348

`sforce.dta` contains the names of the individuals in our sales force along with the region in which they operate:

```
. use sforce, clear
(Sales Force)
. list
```

	region	name
1.	N Cntrl	Krantz
2.	N Cntrl	Phipps
3.	N Cntrl	Willis
4.	NE	Ecklund
5.	NE	Franks
6.	South	Anderson
7.	South	Dubnoff
8.	South	Lee
9.	South	McNiel
10.	West	Charles
11.	West	Cobb
12.	West	Grant

We now wish to merge these two datasets by `region`, spreading the sales and cost information across all observations for which it is relevant; that is, we want to add the variables `sales` and `costs` to the sales-force data. The variable `sales` will assume the value \$360,523 for the first two observations, \$419,472 for the next three observations, and so on.

```
. merge region using dollars
(label region already defined)
. list
```

	region	name	sales	cost	_merge
1.	N Cntrl	Krantz	419,472	227,677	3
2.	N Cntrl	Phipps	419,472	227,677	3
3.	N Cntrl	Willis	419,472	227,677	3
4.	NE	Ecklund	360,523	138,097	3
5.	NE	Franks	360,523	138,097	3
6.	South	Anderson	532,399	330,499	3
7.	South	Dubnoff	532,399	330,499	3
8.	South	Lee	532,399	330,499	3
9.	South	McNiel	532,399	330,499	3
10.	West	Charles	310,565	165,348	3
11.	West	Cobb	310,565	165,348	3
12.	West	Grant	310,565	165,348	3

Even though there are twelve observations in the sales force data and only four observations in the sales and cost data, all the records merged. The `dollars.dta` contained one observation for the NE region. The `sforce.dta` contained two observations for the same region. Thus the single observation in `dollars.dta` was matched to both the observations in `sforce.dta`. In technical jargon, the single record in `dollars.dta` was replicated, or *spread*, across the observations in `sforce.dta`.

◀

## Match-merging with multiple using datasets

### ▶ Example 6

To demonstrate merging with multiple using files, we will use the `even` dataset as the master and some new using datasets `odd3` and `letter`. The `odd3` dataset is just the `odd2` dataset with an added entry for the sixth odd number.

```
. use odd3, clear
(First six odd numbers)
. list
```

	number	odd
1.	1	1
2.	2	3
3.	3	5
4.	4	7
5.	5	9
6.	6	11

The `letter` dataset contains a collection of letters from the alphabet, sorted by number.

```
. use letter, clear
(Some letters from the alphabet)
. list
```

	number	letter
1.	3	c
2.	4	d
3.	5	e
4.	8	h
5.	9	i

We will first use the `even` dataset and then merge with both the `odd3` dataset and the `letter` dataset as the using datasets.

```
. use even, clear
(6th through 8th even numbers)
. list
```

	number	even
1.	6	12
2.	7	14
3.	8	16

```
. merge number using odd3 letter
number was int now float
. list
```

	number	even	odd	_merge1	letter	_merge2	_merge
1.	1	.	1	1		0	2
2.	2	.	3	1		0	2
3.	3	.	5	1	c	1	3
4.	4	.	7	1	d	1	3
5.	5	.	9	1	e	1	3
6.	6	12	11	1		0	3
7.	7	14	.	0		0	1
8.	8	16	.	0	h	1	3
9.	9	.	.	0	i	1	2

Notice that we have three new variables, `_merge1`, `_merge2`, and `_merge`. `_merge` is the standard result variable that we have discussed before: 1 means the observation came from the master, 2 means it came from the using, and 3 means it came from both. In this case, however, we have two using datasets, and `_merge1` and `_merge2` clarify what is meant by “from the using” in `_merge`.

`_merge1` contains 1 if the first using dataset (`odd.dta` in this case) contributed to the result, and it contains 0 otherwise. `_merge2` works the same way, but for the second using dataset (`letter.dta` in this case).

## Updating data

`merge` with the `update` option varies `merge`'s actions when an observation in the master dataset is matched with an observation in the using dataset. Without the `update` option, `merge` leaves the values in the master dataset alone and adds the data for the new variables. With the `update` option, `merge` adds the new variables, but it also replaces missing values in the master observation with corresponding values from the using. Missing values are numeric missing (`.`) and empty strings (`""`).

The values for `_merge` are extended:

<code>_merge</code>	meaning
1	observations from master data
2	observations from using data
3	observations from both, master agrees with using
4	observations from both, missing in master updated
5	observations from both, master disagrees with using

In the case of `_merge = 5`, the master values are retained unless `replace` is specified, in which case the master values are updated as if they had been missing.

Suppose that dataset 1 contains variables `id`, `a`, and `b`, and that dataset 2 contains `id`, `a`, and `x`. You `merge` the two datasets by `id`, dataset 1 being the master dataset in memory and dataset 2 being the using dataset on disk. Consider two observations that match, and call the values from the first dataset `id1`, etc., and those from the second `id2`, etc. The resulting dataset will have variables `id`, `a`, `b`, `x`, and `_merge`. `merge`'s typical logic is as follows:

1. The fact that the observations match means  $id_1 = id_2$ . Set  $id = id_1$ .
2. Variable `a` occurs in both datasets. Ignore  $a_2$ , and set  $a = a_1$ .
3. Variable `b` occurs in only dataset 1. Set  $b = b_1$ .
4. Variable `x` occurs in only dataset 2. Set  $x = x_2$ .
5. Set `_merge = 3`.

With `update`, the logic is modified:

1. (unchanged.) Since the observations match,  $id_1 = id_2$ . Set  $id = id_1$ .
2. Variable `a` occurs in both datasets:
  - a. If  $a_1 = a_2$ , set  $a = a_1$ , and set `_merge = 3`.
  - b. If  $a_1$  contains missing and  $a_2$  is nonmissing, set  $a = a_2$  and set `_merge = 4`, indicating that an update was made.
  - c. If  $a_2$  contains missing, set  $a = a_1$  and set `_merge = 3`, indicating no update.
  - d. If  $a_1 \neq a_2$  and both contain nonmissing, set  $a = a_1$ , or, if `replace` was specified,  $a = a_2$ . Regardless, set `_merge = 5`, indicating a disagreement.

Rules 3 and 4 remain unchanged.

### ► Example 7

In `original.dta`, we have data on some cars that include the make, price, and mileage rating. In `updates.dta`, we have some updated data on these cars, along with a new variable recording engine displacement. The data contain

```
. use original, clear
(original data)
. list
```

	make	price	mpg
1.	Chev. Chevette	3,299	29
2.	Chev. Malibu	4,504	.
3.	Datsun 510	5,079	24
4.	Merc. XR-7	6,303	.
5.	Olds Cutlass	4,733	19
6.	Renault Le Car	3,895	26
7.	VW Dasher	7,140	23

```
. use updates, clear
(updates, mpg and displacement)
. list
```

	make	mpg	displa~t
1.	Chev. Chevette	.	231
2.	Chev. Malibu	22	200
3.	Datsun 510	24	119
4.	Merc. XR-7	14	302
5.	Olds Cutlass	19	231
6.	Renault Le Car	25	79
7.	VW Dasher	23	97

By updating our data, we obtain

```
. use original, clear
(original data)
. merge make using updates, update
. list
```

	make	price	mpg	displa~t	_merge
1.	Chev. Chevette	3,299	29	231	3
2.	Chev. Malibu	4,504	22	200	4
3.	Datsun 510	5,079	24	119	3
4.	Merc. XR-7	6,303	14	302	4
5.	Olds Cutlass	4,733	19	231	3
6.	Renault Le Car	3,895	26	79	5
7.	VW Dasher	7,140	23	97	3

All observations merged because all have `_merge`  $\geq$  3. The observations having `_merge` = 3 have mpg just as it was recorded in the original dataset. In observation 1, mpg is 29 because the updated dataset had mpg = .; in observation 3, mpg remains 24 because the updated dataset also stated that mpg is 24.

The observations having `_merge` = 4 have had their mpg data updated. The mpg variable was missing in observations 2 and 4, and new values were obtained from the update data.

The observation having `_merge = 5` has its `mpg` as it was recorded in the original dataset, as do the `_merge = 3` observations, but there is an important difference. There is a disagreement about the value of `mpg`; the original has a value of 26, and the updated has 25. Had we specified the `replace` option, `mpg` would now contain the updated 25, but the observation would still be marked `_merge = 5`. `replace` affects only the value that is retained in the case of disagreement.

◀

## Methods and Formulas

`merge` is implemented as an ado-file.

## References

- Nash, J. D. 1994. `dm19`: Merging raw data and dictionary files. *Stata Technical Bulletin* 20: 3–5. Reprinted in *Stata Technical Bulletin Reprints*, vol. 4, pp. 22–25.
- Weesie, J. 2000. `dm75`: Safe and easy matched merging. *Stata Technical Bulletin* 53: 6–17. Reprinted in *Stata Technical Bulletin Reprints*, vol. 9, pp. 62–77.

## Also See

- Complementary:** [D] `save`, [D] `sort`
- Related:** [D] `append`, [D] `cross`, [D] `joinby`
- Background:** [U] 22 **Combining datasets**