

format — Set variables' output format

 Description
 Option

 Quick start
 Remarks and examples

 Menu
 References

 Syntax
 Also see

Description

`format varlist %fmt` and `format %fmt varlist` are the same commands. They set the display format associated with the variables specified. The default formats are a function of the type of the variable:

byte	%8.0g
int	%8.0g
long	%12.0g
float	%9.0g
double	%10.0g
str#	%#s
strL	%9s

`set dp` sets the symbol that Stata uses to represent the decimal point. The default is `period`, meaning that one and a half is displayed as 1.5.

`format [varlist]` displays the current formats associated with the variables. `format` by itself lists all variables that have formats too long to be listed in their entirety by `describe`. `format varlist` lists the formats for the specified variables regardless of their length. `format *` lists the formats for all the variables.

Quick start

Show 10-digit `v1` as whole numbers with commas
`format v1 %15.0gc`

Same as above
`format %15.0gc v1`

Left-align string variable `v2` of type `str20`
`format v2 %-20s`

Show 3-digit `v3` with 1 digit after the decimal
`format v3 %4.1f`

Left-align `v4` and `v5` and show with leading zeros if less than 4 digits in length
`format v4 v5 %-04.0f`

Show `v6` in Stata default date format like 19jun2014
`format v6 %td`

As above, but show `v6` in a date format like 06/14/2014
`format v6 %tdNN/DD/CCYY`

Menu

Data > Variables Manager

Syntax

Set formats

```
format varlist %fnt
```

```
format %fnt varlist
```

Set style of decimal point

```
set dp { comma | period } [ , permanently ]
```

Display long formats

```
format [ varlist ]
```

where %*fnt* can be a numerical, date, business calendar, or string format.

Numerical % <i>fnt</i>	Description	Example
right-justified		
%#.#g	general	%9.0g
%#.#f	fixed	%9.2f
%#.#e	exponential	%10.7e
%21x	hexadecimal	%21x
%16H	binary, hilo	%16H
%16L	binary, lohi	%16L
%8H	binary, hilo	%8H
%8L	binary, lohi	%8L
right-justified with commas		
%#.#g _c	general	%9.0g _c
%#.#f _c	fixed	%9.2f _c
right-justified with leading zeros		
%0#.#f	fixed	%09.2f
left-justified		
%-#.#g	general	%-9.0g
%-#.#f	fixed	%-9.2f
%-#.#e	exponential	%-10.7e
left-justified with commas		
%-#.#g _c	general	%-9.0g _c
%-#.#f _c	fixed	%-9.2f _c

You may substitute comma (,) for period (.) in any of the above formats to make comma the decimal point. In %9.2f_c, 1000.03 is 1.000,03. Or you can **set** dp **comma**.

date <i>%fmt</i>	Description	Example
right-justified		
<i>%tc</i>	date/time	<i>%tc</i>
<i>%tC</i>	date/time	<i>%tC</i>
<i>%td</i>	date	<i>%td</i>
<i>%tw</i>	week	<i>%tw</i>
<i>%tm</i>	month	<i>%tm</i>
<i>%tq</i>	quarter	<i>%tq</i>
<i>%th</i>	half-year	<i>%th</i>
<i>%ty</i>	year	<i>%ty</i>
<i>%tg</i>	generic	<i>%tg</i>
left-justified		
<i>%-tc</i>	date/time	<i>%-tc</i>
<i>%-tC</i>	date/time	<i>%-tC</i>
<i>%-td</i>	date	<i>%-td</i>
etc.		

There are many variations allowed. See [\[D\] datetime display formats](#).

business calendar <i>%fmt</i>	Description	Example
<i>%tbcalname</i> [<i>:datetime-specifiers</i>]	a business calendar defined in <i>calname.stbcal</i>	<i>%tbsimple</i>

See [\[D\] datetime business calendars](#).

string <i>%fmt</i>	Description	Example
right-justified		
<i>%#s</i>	string	<i>%15s</i>
left-justified		
<i>%-#s</i>	string	<i>%-20s</i>
centered		
<i>%~#s</i>	string	<i>%~12s</i>

The centered format is for use with `display` only.

Option

`permanently` specifies that, in addition to making the change right now, the `dp` setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

Remarks are presented under the following headings:

- [Setting formats](#)
- [Setting European formats](#)
- [Details of formats](#)
 - [The %f format](#)
 - [The %fc format](#)
 - [The %g format](#)
 - [The %gc format](#)
 - [The %e format](#)
 - [The %21x format](#)
 - [The %16H and %16L formats](#)
 - [The %8H and %8L formats](#)
 - [The %t format](#)
 - [The %s format](#)
- [Other effects of formats](#)
- [Displaying current formats](#)
- [Video example](#)

Setting formats

See [\[U\] 12.5 Formats: Controlling how data are displayed](#) for an explanation of *%fmt*. To review: Stata's three numeric formats are denoted by a leading percent sign, %, followed by the string *w.d* (or *w,d* for European format), where *w* and *d* stand for two integers. The first integer, *w*, specifies the width of the format. The second integer, *d*, specifies the number of digits that are to follow the decimal point; *d* must be less than *w*. Finally, a character denoting the format type (*e*, *f*, or *g*) is appended. For example, %9.2f specifies the *f* format that is nine characters wide and has two digits following the decimal point. For *f* and *g*, a *c* may also be suffixed to indicate comma formats. Other "numeric" formats known collectively as the *%t* formats are used to display dates and times; see [\[D\] datetime display formats](#). String formats are denoted by *%ws*, where *w* indicates the width of the format.

► Example 1

We have census data by region and state on median age and population in 1980.

```
. use http://www.stata-press.com/data/r15/census10
(1980 Census data by state)

. describe

Contains data from http://www.stata-press.com/data/r15/census10.dta
obs:          50                1980 Census data by state
vars:         4                 9 Apr 2016 08:05
size:        1,200
```

variable name	storage type	display format	value label	variable label
state	str14	%14s		State
region	int	%8.0g	cenreg	Census region
pop	long	%11.0g		Population
medage	float	%9.0g		Median age

Sorted by:

```
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3893888	29.3
2.	Alaska	West	401851	26.1
3.	Arizona	West	2718215	29.2
4.	Arkansas	South	2286435	30.6
5.	California	West	23667902	29.9
6.	Colorado	West	2889964	28.6
7.	Connecticut	NE	3107576	32
8.	Delaware	South	594338	29.8

The state variable has a display format of %14s. To left-align the state data, we type

```
. format state %-14s
```

```
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3893888	29.3
2.	Alaska	West	401851	26.1
3.	Arizona	West	2718215	29.2
4.	Arkansas	South	2286435	30.6
5.	California	West	23667902	29.9
6.	Colorado	West	2889964	28.6
7.	Connecticut	NE	3107576	32
8.	Delaware	South	594338	29.8

Although it seems like region is a string variable, it is really a numeric variable with an attached value label. You do the same thing to left-align a numeric variable as you do a string variable: insert a negative sign.

```
. format region %-8.0g
```

```
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3893888	29.3
2.	Alaska	West	401851	26.1
3.	Arizona	West	2718215	29.2
4.	Arkansas	South	2286435	30.6
5.	California	West	23667902	29.9
6.	Colorado	West	2889964	28.6
7.	Connecticut	NE	3107576	32
8.	Delaware	South	594338	29.8

The pop variable would probably be easier to read if we inserted commas by appending a 'c':

```
. format pop %11.0gc
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3,893,888	29.3
2.	Alaska	West	401,851	26.1
3.	Arizona	West	2,718,215	29.2
4.	Arkansas	South	2,286,435	30.6
5.	California	West	23667902	29.9
6.	Colorado	West	2,889,964	28.6
7.	Connecticut	NE	3,107,576	32
8.	Delaware	South	594,338	29.8

Look at the value of pop for observation 5. There are no commas. This number was too large for Stata to insert commas and still respect the current width of 11. Let's try again:

```
. format pop %12.0gc
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3,893,888	29.3
2.	Alaska	West	401,851	26.1
3.	Arizona	West	2,718,215	29.2
4.	Arkansas	South	2,286,435	30.6
5.	California	West	23,667,902	29.9
6.	Colorado	West	2,889,964	28.6
7.	Connecticut	NE	3,107,576	32
8.	Delaware	South	594,338	29.8

Finally, medage would look better if the decimal points were vertically aligned.

```
. format medage %8.1f
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3,893,888	29.3
2.	Alaska	West	401,851	26.1
3.	Arizona	West	2,718,215	29.2
4.	Arkansas	South	2,286,435	30.6
5.	California	West	23,667,902	29.9
6.	Colorado	West	2,889,964	28.6
7.	Connecticut	NE	3,107,576	32.0
8.	Delaware	South	594,338	29.8

Display formats are permanently attached to variables by the format command. If we save the data, the next time we use it, state will still be formatted as %-14s, region will still be formatted as %-8.0g, etc.

▷ Example 2

Suppose that we have an employee identification variable, `empid`, and that we want to retain the leading zeros when we list our data. `format` has a leading-zero option that allows this.

```
. use http://www.stata-press.com/data/r15/fmtxmpl
. describe empid
```

variable name	storage type	display format	value label	variable label
---------------	--------------	----------------	-------------	----------------

<code>empid</code>	float	<code>%9.0g</code>		
--------------------	-------	--------------------	--	--

```
. list empid in 83/87
```

	empid
83.	98
84.	99
85.	100
86.	101
87.	102

```
. format empid %05.0f
```

```
. list empid in 83/87
```

	empid
83.	00098
84.	00099
85.	00100
86.	00101
87.	00102

◀

□ Technical note

The syntax of the `format` command allows a *varlist* and not just one variable name. Thus you can attach the `%9.2f` format to the variables `myvar`, `thisvar`, and `thatvar` by typing

```
. format myvar thisvar thatvar %9.2f
```

□

▷ Example 3

We have employee data that includes `hiredate` and `login` and `logout` times. `hiredate` is stored as a float, but we were careful to store `login` and `logout` as doubles. We need to attach a date format to these three variables.

```
. use http://www.stata-press.com/data/r15/fmtxmpl2
```

```
. format hiredate login logout
```

variable name	display format
---------------	----------------

<code>hiredate</code>	<code>%9.0g</code>
<code>login</code>	<code>%10.0g</code>
<code>logout</code>	<code>%10.0g</code>

```
. format login logout %tcDDmonCCYY_HH:MM:SS.ss
. list login logout in 1/5
```

	login	logout
1.	08nov2006 08:16:42.30	08nov2006 05:32:23.53
2.	08nov2006 08:07:20.53	08nov2006 05:57:13.40
3.	08nov2006 08:10:29.48	08nov2006 06:17:07.51
4.	08nov2006 08:30:02.19	08nov2006 05:42:23.17
5.	08nov2006 08:29:43.25	08nov2006 05:29:39.48

```
. format hiredate %td
. list hiredate in 1/5
```

	hiredate
1.	24jan1986
2.	10mar1994
3.	29sep2006
4.	14apr2006
5.	03dec1999

We remember that the project manager requested that hire dates be presented in the same form as they were previously.

```
. format hiredate %tdDD/NN/CCYY
. list hiredate in 1/5
```

	hiredate
1.	24/01/1986
2.	10/03/1994
3.	29/09/2006
4.	14/04/2006
5.	03/12/1999

Setting European formats

Do you prefer that one and one half be written as 1,5 and that one thousand one and a half be written as 1.001,5? Stata will present numbers in that format if, when you set the format, you specify ‘,’ rather than ‘.’ as follows:

```
. use http://www.stata-press.com/data/r15/census10
(1980 Census data by state)
. format pop %12,0gc
. format medage %9,2f
```



```
. list in 1/8
```

	state	region	pop	medage
1.	Alabama	South	3.893.888	29,30
2.	Alaska	West	401.851	26,10
3.	Arizona	West	2.718.215	29,20
4.	Arkansas	South	2.286.435	30,60
5.	California	West	23.667.902	29,90
6.	Colorado	West	2.889.964	28,60
7.	Connecticut	NE	3.107.576	32,00
8.	Delaware	South	594.338	29,80

You can also leave the formats just as they were and instead type `set dp comma`. That tells Stata to interpret all formats as if you had typed the comma instead of the period:

```
. format pop %12.0gc           (put the formats back as they were)
. format medage %9.2f
. set dp comma                 (tell Stata to use European format)
. list in 1/8
(same output appears as above)
```

`set dp comma` affects all Stata output, so if you run a regression, display summary statistics, or make a table, commas will be used instead of periods in the output:

```
. tabulate region [fw=pop]
```

Census region	Freq.	Percent	Cum.
NE	49135283	21,75	21,75
N Cntrl	58865670	26,06	47,81
South	74734029	33,08	80,89
West	43172490	19,11	100,00
Total	225907472	100,00	

You can return to using periods by typing

```
. set dp period
```

Setting a variable's display format to European affects how the variable's values are displayed by `list` and in a few other places. Setting `dp` to `comma` affects every bit of Stata.

Also, `set dp comma` affects only how Stata displays output, not how it gets input. When you need to type one and a half, you must type `1.5` regardless of context.

□ Technical note

`set dp comma` makes drastic changes inside Stata, and we mention this because some older, user-written programs may not be able to deal with those changes. If you are using an older, user-written program, you might `set dp comma` only to find that the program does not work and instead presents some sort of syntax error.

If, using any program, you get an unanticipated error, try setting `dp` back to `period`.

Even with `set dp comma`, you might still see some output with the decimal symbol shown as a period rather than a comma. There are two places in Stata where Stata ignores `set dp comma` because the features are generally used to produce what will be treated as input, and `set dp comma` does not affect how Stata inputs numbers. First,

```
local x = sqrt(2)
```

stores the string "1.414213562373095" in `x` and not "1,414213562373095", so if some program were to display 'x' as a string in the output, the period would be displayed. Most programs, however, would use 'x' in subsequent calculations or, at the least, when the time came to display what was in 'x', would display it as a number. They would code

```
display ... 'x' ...
```

and not

```
display ... "x" ...
```

so the output would be

```
... 1,4142135 ...
```

The other place where Stata ignores `set dp comma` is the `string()` function. If you type

```
. generate res = string(numvar)
```

new variable `res` will contain the string representation of numeric variable `numvar`, with the decimal symbol being a period, even if you have previously `set dp comma`. Of course, if you explicitly ask that `string()` use European format,

```
. generate res = string(numvar,"%9,0g")
```

then `string()` honors your request; `string()` merely ignores the global `set dp comma`. □

Details of formats

The %f format

In `%w.df`, `w` is the total output width, including sign and decimal point, and `d` is the number of digits to appear to the right of the decimal point. The result is right-justified.

The number 5.139 in `%12.2f` format displays as

```
----+----1--
      5.14
```

When `d = 0`, the decimal point is not displayed. The number 5.14 in `%12.0f` format displays as

```
----+----1--
      5
```

`%-w.df` works the same way, except that the output is left-justified in the field. The number 5.139 in `%-12.2f` displays as

```
----+----1--
5.14
```

The %fc format

`%w.dfc` works like `%w.df` except that commas are inserted to make larger numbers more readable. `w` records the total width of the result, including commas.

The number 5.139 in `%12.2fc` format displays as

```
----+----1--
      5.14
```

The number 5203.139 in `%12.2fc` format displays as

```
----+----1--
      5,203.14
```

As with `%f`, if $d = 0$, the decimal point is not displayed. The number 5203.139 in `%12.0fc` format displays as

```
----+----1--
      5,203
```

As with `%f`, a minus sign may be inserted to left justify the output. The number 5203.139 in `%-12.0fc` format displays as

```
----+----1--
5,203
```

The %g format

In `%w.dg`, w is the overall width, and d is usually specified as 0, which leaves up to the format the number of digits to be displayed to the right of the decimal point. If $d \neq 0$ is specified, then not more than d digits will be displayed. As with `%f`, a minus sign may be inserted to left-justify results.

`%g` differs from `%f` in that 1) it decides how many digits to display to the right of the decimal point, and 2) it will switch to a `%e` format if the number is too large or too small.

The number 5.139 in `%12.0g` format displays as

```
----+----1--
      5.139
```

The number 5231371222.139 in `%12.0g` format displays as

```
----+----1--
5231371222
```

The number 52313712223.139 displays as

```
----+----1--
5.23137e+10
```

The number 0.0000029394 displays as

```
----+----1--
2.93940e-06
```

The %gc format

`%w.dgc` is `%w.dg` with commas. It works in the same way as the `%g` and `%fc` formats.

The %e format

`%w.de` displays numeric values in exponential format. w records the width of the format. d records the number of digits to be shown after the decimal place. w should be greater than or equal to $d+7$ or, if 3-digit exponents are expected, $d+8$.

The number 5.139 in `%12.4e` format is

```
----+----1--
5.1390e+00
```

The number 5.139×10^{220} is

```
----+-----1--  
5.1390e+220
```

The %21x format

The %21x format is for those, typically programmers, who wish to analyze routines for numerical roundoff error. There is no better way to look at numbers than how the computer actually records them.

The number 5.139 in %21x format is

```
----+-----1-----+-----2-  
+1.48e5604189375X+002
```

The number 5.125 is

```
----+-----1-----+-----2-  
+1.48000000000000X+002
```

Reported is a signed, base-16 number with base-16 point, the letter X, and a signed, 3-digit base-16 integer. Call the two numbers f and e . The interpretation is $f \times 2^e$.

The %16H and %16L formats

The %16H and %16L formats show the value in the IEEE floating point, double-precision form. %16H shows the value in most-significant-byte-first (hilo) form. %16L shows the number in least-significant-byte-first (lohi) form.

The number 5.139 in %16H is

```
----+-----1-----+-  
40148e5604189375
```

The number 5.139 in %16L is

```
----+-----1-----+-  
75931804568e1440
```

The format is sometimes used by programmers who are simultaneously studying a hexadecimal dump of a binary file.

The %8H and %8L formats

%8H and %8L are similar to %16H and %16L but show the number in IEEE single-precision form.

The number 5.139 in %8H is

```
----+-----  
40a472b0
```

The number 5.139 in %8L is

```
----+-----  
b072a440
```

The %t format

The %t format displays numerical variables as dates and times. See [\[D\] datetime display formats](#).

The %s format

The %*ws* format displays a string in a right-justified field of width *w*. %-*ws* displays the string left-justified.

“Mary Smith” in %16s format is

```
----+-----1-----+
      Mary Smith
```

“Mary Smith” in %-16s format is

```
----+-----1-----+
Mary Smith
```

Also, in some contexts, particularly `display` (see [P] [display](#)), %-*ws* is allowed, which centers the string. “Mary Smith” in %-16s format is

```
----+-----1-----+
      Mary Smith
```

Other effects of formats

You have data on the age of employees, and you type `summarize age` to obtain the mean and standard deviation. By default, Stata uses its default `g` format to provide as much precision as possible.

```
. use http://www.stata-press.com/data/r15/fmtxmpl
. summarize age
```

Variable	Obs	Mean	Std. Dev.	Min	Max
age	204	30.18627	10.38067	18	66

If you attach a %9.2f format to the variable and specify the `format` option, Stata uses that specification to format the results:

```
. format age %9.2f
. summarize age, format
```

Variable	Obs	Mean	Std. Dev.	Min	Max
age	204	30.19	10.38	18.00	66.00

Displaying current formats

`format varlist` is not often used to display the formats associated with variables because using `describe` (see [D] [describe](#)) is easier and provides more information. The exceptions are date variables. Unless you use the default %tc, %tC, ... formats (and most people do), the format specifier itself can become very long, such as

```
. format admittime %tcDDmonCCYY_HH:MM:SS.sss
```

Such formats are too long for `describe` to display, so it gives up. In such cases, you can use `format` to display the format:

```
. format admittime
variable name display format
-----
admittime      %tcDDmonCCYY_HH:MM:SS.sss
```

Type `format *` to see the formats for all the variables.

Video example

How to change the display format of a variable

References

- Cox, N. J. 2011. Speaking Stata: MMXI and all that: Handling Roman numerals within Stata. *Stata Journal* 11: 126–142.
- Gould, W. W. 2011a. How to read the %21x format. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2011/02/02/how-to-read-the-percent-21x-format/>.
- . 2011b. How to read the %21x format, part 2. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2011/02/10/how-to-read-the-percent-21x-format-part-2/>.
- Linhart, J. M. 2008. Mata Matters: Overflow, underflow and the IEEE floating-point format. *Stata Journal* 8: 255–268.

Also see

- [D] **datetime business calendars** — Business calendars
- [D] **datetime display formats** — Display formats for dates and times
- [D] **list** — List values of variables
- [D] **varmanage** — Manage variable labels, formats, and other properties
- [P] **display** — Display strings and values of scalar expressions
- [U] **12.5 Formats: Controlling how data are displayed**
- [U] **12.6 Dataset, variable, and value labels**