format — Set variables’ output format

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
2 format — Set variables’ output format

Syntax

Set formats

```
format varlist %fmt

format %fmt varlist
```

Set style of decimal point

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

Display long formats

```
format [varlist]
```

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

<table>
<thead>
<tr>
<th>Numerical %fmt</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>right-justified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%#.##g</td>
<td>general</td>
<td>%9.0g</td>
</tr>
<tr>
<td>%#.##f</td>
<td>fixed</td>
<td>%9.2f</td>
</tr>
<tr>
<td>%#.##e</td>
<td>exponential</td>
<td>%10.7e</td>
</tr>
<tr>
<td>%21x</td>
<td>hexadecimal</td>
<td>%21x</td>
</tr>
<tr>
<td>%16H</td>
<td>binary, hilo</td>
<td>%16H</td>
</tr>
<tr>
<td>%16L</td>
<td>binary, lohi</td>
<td>%16L</td>
</tr>
<tr>
<td>%8H</td>
<td>binary, hilo</td>
<td>%8H</td>
</tr>
<tr>
<td>%8L</td>
<td>binary, lohi</td>
<td>%8L</td>
</tr>
<tr>
<td>right-justified with commas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%#..##gc</td>
<td>general</td>
<td>%9.0gc</td>
</tr>
<tr>
<td>%#..##fc</td>
<td>fixed</td>
<td>%9.2fc</td>
</tr>
<tr>
<td>right-justified with leading zeros</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%0#.##f</td>
<td>fixed</td>
<td>%09.2f</td>
</tr>
<tr>
<td>left-justified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%-.##g</td>
<td>general</td>
<td>%-9.0g</td>
</tr>
<tr>
<td>%-.##f</td>
<td>fixed</td>
<td>%-9.2f</td>
</tr>
<tr>
<td>%-.##e</td>
<td>exponential</td>
<td>%-10.7e</td>
</tr>
<tr>
<td>left-justified with commas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%-.##gc</td>
<td>general</td>
<td>%-9.0gc</td>
</tr>
<tr>
<td>%-.##fc</td>
<td>fixed</td>
<td>%-9.2fc</td>
</tr>
</tbody>
</table>

You may substitute comma (,) for period (.) in any of the above formats to make comma the decimal point. In %9,2fc, 1000.03 is 1.000,03. Or you can set dp comma.
There are many variations allowed. See [D] Datetime display formats.

See [D] Datetime business calendars.

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 https://www.stata-press.com/data/r16/census10
   (1980 Census data by state)
. describe
Contains data from https://www.stata-press.com/data/r16/census10.dta
   obs: 50  1980 Census data by state
   vars: 4  9 Apr 2018 08:05

storage  display value
  variable name  type  format  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:
format — Set variables’ output format

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

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

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
```

```
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
```

<table>
<thead>
<tr>
<th>state</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>South</td>
</tr>
<tr>
<td>Alaska</td>
<td>West</td>
</tr>
<tr>
<td>Arizona</td>
<td>West</td>
</tr>
<tr>
<td>Arkansas</td>
<td>South</td>
</tr>
<tr>
<td>California</td>
<td>West</td>
</tr>
<tr>
<td>Colorado</td>
<td>West</td>
</tr>
<tr>
<td>Connecticut</td>
<td>NE</td>
</tr>
<tr>
<td>Delaware</td>
<td>South</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>state</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>South</td>
</tr>
<tr>
<td>Alaska</td>
<td>West</td>
</tr>
<tr>
<td>Arizona</td>
<td>West</td>
</tr>
<tr>
<td>Arkansas</td>
<td>South</td>
</tr>
<tr>
<td>California</td>
<td>West</td>
</tr>
<tr>
<td>Colorado</td>
<td>West</td>
</tr>
<tr>
<td>Connecticut</td>
<td>NE</td>
</tr>
<tr>
<td>Delaware</td>
<td>South</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>state</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>South</td>
</tr>
<tr>
<td>Alaska</td>
<td>West</td>
</tr>
<tr>
<td>Arizona</td>
<td>West</td>
</tr>
<tr>
<td>Arkansas</td>
<td>South</td>
</tr>
<tr>
<td>California</td>
<td>West</td>
</tr>
<tr>
<td>Colorado</td>
<td>West</td>
</tr>
<tr>
<td>Connecticut</td>
<td>NE</td>
</tr>
<tr>
<td>Delaware</td>
<td>South</td>
</tr>
</tbody>
</table>

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 https://www.stata-press.com/data/r16/fmtxmpl, clear
. describe empid
  variable name        storage  display value
                     type    format   label     variable label
     empid             float    %9.0g
. 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 https://www.stata-press.com/data/r16/fmtxmpl2, clear
. format hiredate login logout
  variable name   display format
                  %9.0g      %10.0g     %10.0g
```
. format login logout %tcDDmonCCYY_HH:MM:SS.ss
. list login logout in 1/5

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>logout</td>
<td>login</td>
</tr>
<tr>
<td>1.</td>
<td>08nov2006 08:16:42.30</td>
<td>08nov2006 05:32:23.53</td>
</tr>
<tr>
<td>2.</td>
<td>08nov2006 08:07:20.53</td>
<td>08nov2006 05:57:13.40</td>
</tr>
<tr>
<td>3.</td>
<td>08nov2006 08:10:29.48</td>
<td>08nov2006 06:17:07.51</td>
</tr>
<tr>
<td>4.</td>
<td>08nov2006 08:30:02.19</td>
<td>08nov2006 05:42:23.17</td>
</tr>
<tr>
<td>5.</td>
<td>08nov2006 08:29:43.25</td>
<td>08nov2006 05:29:39.48</td>
</tr>
</tbody>
</table>

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

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hiredate</td>
</tr>
<tr>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
</tr>
<tr>
<td>3.</td>
</tr>
<tr>
<td>4.</td>
</tr>
<tr>
<td>5.</td>
</tr>
</tbody>
</table>

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

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hiredate</td>
</tr>
<tr>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
</tr>
<tr>
<td>3.</td>
</tr>
<tr>
<td>4.</td>
</tr>
<tr>
<td>5.</td>
</tr>
</tbody>
</table>

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 https://www.stata-press.com/data/r16/census10, clear
   (1980 Census data by state)
. format pop %12,0gc
. format medage %9,2f
. list in 1/8

<table>
<thead>
<tr>
<th>state</th>
<th>region</th>
<th>pop</th>
<th>medage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>South</td>
<td>3,893.888</td>
<td>29,30</td>
</tr>
<tr>
<td>Alaska</td>
<td>West</td>
<td>401,851</td>
<td>26,10</td>
</tr>
<tr>
<td>Arizona</td>
<td>West</td>
<td>2,718,215</td>
<td>29,20</td>
</tr>
<tr>
<td>Arkansas</td>
<td>South</td>
<td>2,286,435</td>
<td>30,60</td>
</tr>
<tr>
<td>California</td>
<td>West</td>
<td>23,667,902</td>
<td>29,90</td>
</tr>
<tr>
<td>Colorado</td>
<td>West</td>
<td>2,889,964</td>
<td>28,60</td>
</tr>
<tr>
<td>Connecticut</td>
<td>NE</td>
<td>3,107,576</td>
<td>32,00</td>
</tr>
<tr>
<td>Delaware</td>
<td>South</td>
<td>594,338</td>
<td>29,80</td>
</tr>
</tbody>
</table>

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]
```

<table>
<thead>
<tr>
<th>Census region</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td>49135283</td>
<td>21,75</td>
<td>21,75</td>
</tr>
<tr>
<td>N Cntrl</td>
<td>58865670</td>
<td>26,06</td>
<td>47,81</td>
</tr>
<tr>
<td>South</td>
<td>74734029</td>
<td>33,08</td>
<td>80,89</td>
</tr>
<tr>
<td>West</td>
<td>43172490</td>
<td>19,11</td>
<td>100,00</td>
</tr>
<tr>
<td>Total</td>
<td>225907472</td>
<td>100,00</td>
<td></td>
</tr>
</tbody>
</table>

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 %12.2f format displays as

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

As with %f, if \( d = 0 \), the decimal point is not displayed. The number 5203.139 in %12.0f 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 %-12.0f 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 %f c 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 \(5.139 \times 10^{220}\) is

\[
\begin{array}{c}
\text{----+----1--} \\
5.1390e+220
\end{array}
\]

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

\[
\begin{array}{c}
\text{----+----1----+----2--} \\
+1.48e5604189375X+002
\end{array}
\]

The number 5.125 is

\[
\begin{array}{c}
\text{----+----1----+----2--} \\
+1.4800000000000X+002
\end{array}
\]

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

\[
\begin{array}{c}
\text{----+----1----+--} \\
40148e5604189375
\end{array}
\]

The number 5.139 in \(\%16L\) is

\[
\begin{array}{c}
\text{----+----1----+--} \\
75931804568e1440
\end{array}
\]

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

\[
\begin{array}{c}
\text{----+---} \\
40a472b0
\end{array}
\]

The number 5.139 in \(\%8L\) is

\[
\begin{array}{c}
\text{----+---} \\
b072a440
\end{array}
\]

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

\[
\begin{array}{c}
\hline
-----1-----
\hline
Mary Smith
\end{array}
\]

“Mary Smith” in \%-16s format is

\[
\begin{array}{c}
\hline
-----1-----
\hline
Mary Smith
\end{array}
\]

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

\[
\begin{array}{c}
\hline
-----1-----
\hline
Mary Smith
\end{array}
\]

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.

\[
\begin{verbatim}
. use https://www.stata-press.com/data/r16/fmtxmpl, clear
. summarize age

Variable | Obs Mean Std. Dev. | Min | Max
---------|-------------------|-----|-----
age      | 204 30.18627 10.38067 | 18 | 66
\end{verbatim}
\]

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

\[
\begin{verbatim}
. format age %9.2f
. summarize age, format

Variable | Obs Mean Std. Dev. | Min | Max
---------|-------------------|-----|-----
age      | 204 30.19 10.38 | 18.00 | 66.00
\end{verbatim}
\]

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, \ldots formats (and most people do), the format specifier itself can become very long, such as

\[
\begin{verbatim}
. format admittime %tcDDmonCCYY_HH:MM:SS.sss
\end{verbatim}
\]

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

\[
\begin{verbatim}
. format admittime

| variable name | display format
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>admittime</td>
<td>%tcDDmonCCYY_HH:MM:SS.sss</td>
</tr>
</tbody>
</table>
\end{verbatim}
\]

Type format * to see the formats for all the variables.
14 format — Set variables’ output format

Video example

How to change the display format of a variable

References


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