gnuplot

GNUPLOT is a command-driven interactive function plotting program.

For help on any topic, type help followed by the name of the topic.

The new GNUPLOT user should begin by reading the introduction topic (type **help introduction**) and about the plot command (type help plot). Additional help can be obtained from the USENET newsgroup comp.graphics.gnuplot.

copyright

AUTHORS

Original Software:
  Thomas Williams, Colin Kelley.

Gnuplot 2.0 additions:
  Russell Lang, Dave Kotz, John Campbell.

Gnuplot 3.0 additions:
  Gershon Elber and many others.

There is a mailing list for gnuplot users. Note, however, that the newsgroup
      comp.graphics.gnuplot
is identical to the mailing list (they both carry the same set of messages). We prefer that you read the messages through that newsgroup, to subscribing to the mailing list. (If you can read that newsgroup, and are already on the mailing list, please send a message info-gnuplot-request@dartmouth.edu, asking to be removed from the mailing list.)

The address for mailing to list members is
      info-gnuplot@dartmouth.edu
and for mailing administrative requests is

```
      info-gnuplot-request@dartmouth.edu
The mailing list for bug reports is
      bug-gnuplot@dartmouth.edu
The list of those interested in beta-test versions is
      info-gnuplot-beta@dartmouth.edu
```

introduction

GNUPLOT is a command-driven interactive function plotting program. It is case sensitive (commands and function names written in lowercase are not the same as those written in CAPS). All command names may be abbreviated, as long as the abbreviation is not ambiguous. Any number of commands may appear on a line, separated by semicolons (;). Strings are indicated with quotes.   They may be either single or double quotation marks, e.g.,

```
load "filename"
cd 'dir'
```

Any command-line arguments are assumed to be names of files containing GNUPLOT commands, with the exception of standard X11 arguments, which are processed first. Each file is loaded with the load command, in the order specified. GNUPLOT exits after the last file is processed.   When no load files are named, gnuplot enters into an interactive mode.

Commands may extend over several input lines, by ending each line but the last with a backslash (\). The backslash must be the LAST character on each line. The effect is as if the backslash and newline were not there. That is, no white space is implied, nor is a comment terminated. Therefore, commenting out a continued line comments out the entire command (see comment).

In this documentation, curly braces ({}) denote optional arguments to many commands, and a vertical bar (|) separates mutually exclusive choices.   GNUPLOT keywords or help topics are indicated by backquotes or **boldface** (where available).   Angle brackets (<>) are used to mark replaceable tokens.

For help on any topic, type help followed by the name of the topic.

The new GNUPLOT user should begin by reading about the plot command (type help plot).

cd

The **cd** command changes the working directory.

Syntax:

```
        cd "<directory-name>"
```

The directory name must be enclosed in quotes.

Examples:

```
        cd 'subdir'
        cd ".."
```

clear

The **clear** command erases the current screen or output device as specified by set output. This usually generates a formfeed on hardcopy devices. Use set terminal to set the device type.

command line-editing

The Unix, Atari, VMS, MS-DOS and OS/2 versions of GNUPLOT support command line-editing. Also, a history mechanism allows previous commands to be edited, and re-executed. After the command line has been edited, a newline or carriage return will enter the entire line regardless of where the cursor is positioned.

The editing commands are as follows:

```
 `Line-editing`:


 ^B moves back a single character.
 ^F moves forward a single character.
 ^A moves to the beginning of the line.
 ^E moves to the end of the line.
 ^H and DEL delete the previous character.
 ^D deletes the current character.
 ^K deletes from current position to the end of line.
 ^L,^R redraws line in case it gets trashed.
 ^U deletes the entire line.
 ^W deletes the last word.


 `History`:


 ^P moves back through history.
 ^N moves forward through history.
```

On the IBM PC the use of a TSR program such as DOSEDIT or CED may be desired for line editing. For such a case GNUPLOT may be compiled with no line editing capability (default makefile setup). Set READLINE in the makefile and add readline.obj to the link file if GNUPLOT line editing is to be used for the IBM PC. The following arrow keys may be used on the IBM PC and Atari versions if readline is used:

```
 Left  Arrow     - same as ^B.
 Right Arrow     - same as ^F.
 Ctl Left  Arrow - same as ^A.
 Ctl Right Arrow - same as ^E.
 Up    Arrow     - same as ^P.
 Down  Arrow     - same as ^N.
```

The Atari version of readline defines some additional key aliases:

```
 Undo            - same as ^L.
 Home            - same as ^A.
 Ctrl Home       - same as ^E.
 ESC             - same as ^U.
 Help            - `help' plus return.
 Ctrl Help       - `help '.
```

(The readline function in gnuplot is not the same as the readline used in GNU BASH and GNU EMACS.   It is somewhat compatible however.)

comment

Comments are supported as follows: a # may appear in most places in a line and GNUPLOT will ignore the rest of the line. It will not have this effect inside quotes, inside numbers (including complex numbers), inside command substitutions, etc. In short, it works anywhere it makes sense to work.

environment

A number of shell environment variables are understood by GNUPLOT. None of these are required, but may be useful.

If GNUTERM is defined, it is used as the name of the terminal type to be used. This overrides any terminal type sensed by GNUPLOT on start up, but is itself overridden by the .gnuplot (or equivalent) start-up file (see start-up), and of course by later explicit changes.

On Unix, AmigaDOS, AtariTOS, MS-DOS and OS/2, GNUHELP may be defined to be the pathname of the HELP file (gnuplot.gih).

On VMS, the symbol GNUPLOT$HELP should be defined as the name of the help library for GNUPLOT.

On Unix, HOME is used as the name of a directory to search for a .gnuplot file if none is found in the current directory. On AmigaDOS, AtariTOS, MS-DOS and OS/2, GNUPLOT is used. On VMS, SYS$LOGIN: is used. See help start-up.

On Unix, PAGER is used as an output filter for help messages.

On Unix, AtariTOS and AmigaDOS, SHELL is used for the shell command. On MS-DOS and OS/2, COMSPEC is used for the shell command.

On AmigaDOS, GNUFONT is used for the screen font.   For example: "setenv GNUFONT sapphire/14".

On MS-DOS, if the BGI interface is used, the variable **BGI** is used to point to the full path of the BGI drivers directory. Furthermore SVGA is used to name the Super VGA BGI driver in 800x600 res., and its mode of operation as 'Name.Mode'. E.g., if the Super VGA driver is C:\TC\BGI\SVGADRV.BGI and mode 3 is used for 800x600 res., then: 'set BGI=C:\TC\BGI' and 'set SVGA=SVGADRV.3'.

exit

The commands **exit** and **quit** and the END-OF-FILE character will exit GNUPLOT. All these commands will clear the output device (as the clear command does) before exiting.

expressions

In general, any mathematical expression accepted by C, FORTRAN, Pascal, or BASIC is valid. The precedence of these operators is determined by the specifications of the C programming language. White space (spaces and tabs) is ignored inside expressions.

Complex constants may be expressed as the {<real>,<imag>}, where <real> and <imag> must be numerical constants. For example, {3,2} represents 3 + 2i; {0,1} represents **i** itself. The curly braces are explicitly required here.
functions
operators

functions

The functions in GNUPLOT are the same as the corresponding functions in the Unix math library, except that all functions accept integer, real, and complex arguments, unless otherwise noted. The sgn function is also supported, as in BASIC.
abs
acos
arg
asin
atan
besj0
besj1
besy0
besy1
ceil
cos
cosh
erf
erfc
exp
floor
gamma
ibeta
inverf
igamma
imag
invnorm
int
lgamma
log
log10
norm
rand
real
sgn
sin
sinh
sqrt
tan
tanh

abs

The **abs** function returns the absolute value of its argument. The returned value is of the same type as the argument.

For complex arguments, abs(x) is defined as the length of x in the complex plane [i.e., sqrt(real(x)**2 + imag(x)**2)].

acos

The **acos** function returns the arc cosine (inverse cosine) of its argument. **acos** returns its argument in radians.

arg

The **arg** function returns the phase of a complex number, in radians.

asin

The **asin** function returns the arc sin (inverse sin) of its argument. **asin** returns its argument in radians.

atan

The **atan** function returns the arc tangent (inverse tangent) of its argument. **atan** returns its argument in radians.

besj0

The **besj0** function returns the j0th Bessel function of its argument. **besj0** expects its argument to be in radians.

besj1

The **besj1** function returns the j1st Bessel function of its argument. **besj1** expects its argument to be in radians.

besy0

The **besy0** function returns the y0th Bessel function of its argument. **besy0** expects its argument to be in radians.

besy1

The **besy1** function returns the y1st Bessel function of its argument. **besy1** expects its argument to be in radians.

ceil

The **ceil** function returns the smallest integer that is not less than its argument. For complex numbers, **ceil** returns the smallest integer not less than the real part of its argument.

cos

The **cos** function returns the cosine of its argument. **cos** expects its argument to be in radians.

cosh

The **cosh** function returns the hyperbolic cosine of its argument. **cosh** expects its argument to be in radians.

erf

The **erf** function returns the error function of the real part of its argument. If the argument is a complex value, the imaginary component is ignored.

erfc

The **erfc** function returns 1.0 - the error function of the real part of its argument. If the argument is a complex value, the imaginary component is ignored.

exp

The **exp** function returns the exponential function of its argument (**e** raised to the power of its argument).

floor

The **floor** function returns the largest integer not greater than its argument. For complex numbers, **floor** returns the largest integer not greater than the real part of its argument.

gamma

The **gamma** function returns the gamma function of the real part of its argument. For integer n, gamma(n+1) = n! . If the argument is a complex value, the imaginary component is ignored.

ibeta

The **ibeta** function returns the incomplete beta function of the real parts of its arguments. p, q > 0 and x in [0:1] If the arguments are complex, the imaginary components are ignored.

inverf

The **inverf** function returns the inverse error function of the real part of its argument.

igamma

The **igamma** function returns the incomplete gamma function of the real parts of its arguments. a > 0 and x >= 0 If the arguments are complex, the imaginary components are ignored.

imag

The **imag** function returns the imaginary part of its argument as a real number.

invnorm

The **invnorm** function returns the inverse normal distribution function of the real part of its argument.

int

The **int** function returns the integer part of its argument, truncated toward zero.

lgamma

The **lgamma** function returns the natural logarithm of the gamma function of the real part of its argument. If the argument is a complex value, the imaginary component is ignored.

log

The **log** function returns the natural logarithm (base **e**) of its argument.

log10

The **log10** function returns the logarithm (base 10) of its argument.

norm

The **norm** function returns the normal distribution function (or Gaussian) of the real part of its argument.

rand

The **rand** function returns a pseudo random number in the interval [0:1] using the real part of its argument as a seed. If seed < 0 the sequence is (re)initialized. If the argument is a complex value, the imaginary component is ignored.

real

The **real** function returns the real part of its argument.

sgn

The **sgn** function returns 1 if its argument is positive, -1 if its argument is negative, and 0 if its argument is 0. If the argument is a complex value, the imaginary component is ignored.

sin

The **sin** function returns the sine of its argument. **sin** expects its argument to be in radians.

sinh

The **sinh** function returns the hyperbolic sine of its argument. **sinh** expects its argument to be in radians.

sqrt

The **sqrt** function returns the square root of its argument.

tan

The **tan** function returns the tangent of its argument. **tan** expects its argument to be in radians.

tanh

The **tanh** function returns the hyperbolic tangent of its argument. **tanh** expects its argument to be in radians.

operators

The operators in GNUPLOT are the same as the corresponding operators in the C programming language, except that all operators accept integer, real, and complex arguments, unless otherwise noted. The ** operator (exponentiation) is supported, as in FORTRAN.

Parentheses may be used to change order of evaluation.
binary
unary

binary

The following is a list of all the binary operators and their usages:

Symbol	Example	Explanation
`**`	`a**b`	exponentiation
`*`	`a*b`	multiplication
`/`	`a/b`	division
`%`	`a%b`	* modulo
`+`	`a+b`	addition
`-`	`a-b`	subtraction
`==`	`a==b`	equality
`!=`	`a!=b`	inequality
`&`	`a&b`	* bitwise AND
`^`	`a^b`	* bitwise exclusive OR
`|`	`a|b`	* bitwise inclusive OR
`&&`	`a&&b`	* logical AND
`||`	`a||b`	* logical OR
`?:`	`a?b:c`	* ternary operation

(*) Starred explanations indicate that the operator requires integer arguments.

Logical AND (&&) and OR (||) short-circuit the way they do in C. That is, the second && operand is not evaluated if the first is false; the second || operand is not evaluated if the first is true.

The ternary operator evaluates its first argument (a). If it is true (non-zero) the second argument (b) is evaluated and returned, otherwise the third argument (c) is evaluated and returned.

unary

The following is a list of all the unary operators and their usages:

Symbol	Example	Explanation
-	-a	unary minus
~	~a	* one's complement
!	!a	* logical negation
!	a!	* factorial

(*) Starred explanations indicate that the operator requires an integer argument.

The factorial operator returns a real number to allow a greater range.

help

The **help** command displays on-line help. To specify information on a particular topic use the syntax:

```
        help {<topic>}
```

If <topic> is not specified, a short message is printed about GNUPLOT. After help for the requested topic is given, help for a subtopic may be requested by typing its name, extending the help request. After that subtopic has been printed, the request may be extended again, or simply pressing return goes back one level to the previous topic. Eventually, the GNUPLOT command line will return.

load

The **load** command executes each line of the specified input file as if it had been typed in interactively. Files created by the save command can later be **load**ed. Any text file containing valid commands can be created and then executed by the **load** command. Files being **load**ed may themselves contain **load** commands. See comment for information about comments in commands.

The **load** command must be the last command on the line.

Syntax:

```
        load "<input-file>"
```

The name of the input file must be enclosed in quotes.

Examples:

```
        load 'work.gnu'
        load "func.dat"
```

The **load** command is performed implicitly on any file names given as arguments to GNUPLOT. These are loaded in the order specified, and then GNUPLOT exits.

pause

The **pause** command displays any text associated with the command and then waits a specified amount of time or until the carriage return is pressed.   **pause** is especially useful in conjunction with load files.

Syntax:
```
        pause <time> {"<string>"}
```

<time> may be any integer constant or expression. Choosing -1 will wait until a carriage return is hit, zero (0) won't pause at all, and a positive integer will wait the specified number of seconds.

Note: Since **pause** is not part of the plot it may interact with different device drivers differently (depending upon how text and graphics are mixed).

Examples:
```
        pause -1    # Wait until a carriage return is hit
        pause 3     # Wait three seconds
        pause -1  "Hit return to continue"
        pause 10  "Isn't this pretty?  It's a cubic-spline."
```

plot

plot and **splot** are the primary commands of the program. They plot functions and data in many, many ways. **plot** is used to plot 2-d functions and data, while **splot** plots 3-d surfaces and data.

Syntax:

```
       plot {ranges} {<function> | {"<datafile>" {using ...}}}
                    {title} {style} {, <function> {title} {style}...}


       splot {ranges} {<function> | {"<datafile>" {index i} {using ...}}}
                    {title} {style} {, <function> {title} {style}...}
```

where either a <function> or the name of a data file enclosed in quotes is supplied.   A function is a mathematical expression, or a pair (**plot**) or triple (**splot**) of mathematical expressions in the case of parametric functions.   User-defined functions and variables may also be defined here.

plot and **splot** commands can be as simple as

```
        plot sin(x)
```

and

```
        splot x * y
```

or as complex as (!)

```
 plot [t=1:10] [-pi:pi*2] tan(t), "data.1" using 2:3 with lines,
       t**2 with points
```

data-file

Discrete data contained in a file can be displayed by specifying the name of the data file (enclosed in quotes) on the plot or splot command line. Data files should contain one data point per line. Lines beginning with # (or ! on VMS) will be treated as comments and ignored. For plots, each data point represents an (x,y) pair. For splots, each point is an (x,y,z) triple. For plots with error bars (see plot errorbars), each data point is either (x,y,ydelta) or (x,y,ylow,yhigh). In all cases, the numbers on each line of a data file must be separated by blank space. This blank space divides each line into columns.

For plots the x value may be omitted, and for splots the x and y values may be omitted. In either case the omitted values are assigned the current coordinate number. Coordinate numbers start at 0 and are incremented for each data point read.

To specify other formats, see plot datafile using.

In the plot command, blank lines in the data file cause a break in the plot. There will be no line drawn between the preceding and following points if the plot style is lines or linespoints (see plot style). This does not change the plot style, as would plotting the data as separate curves.

This example compares the data in the file population.dat to a theoretical curve:

```
        pop(x) = 103*exp((1965-x)/10)
        plot [1960:1990] 'population.dat', pop(x)
```

The file population.dat might contain:

```
        # Gnu population in Antarctica since 1965
        1965   103
        1970   55
        1975   34
        1980   24
        1985   10
```

When a data file is plotted, samples and isosamples are ignored. Curves plotted using the plot command are automatically extended to hold the entire curve. Similarly grid data plotted using the splot command is automatically extended, using the assumption that isolines are separated by blank lines (a line with only a CR/LF in it).

Implicitly, there are two types of 3-d datafiles. If all the isolines are of the same length, the data is assumed to be a grid data, i.e., the data has a grid topology. Cross isolines in the other parametric direction (the ith cross isoline passes through the ith point of all the provided isolines) will also be drawn for grid data. (Note contouring is available for grid data only.) If all the isolines are not of the same length, no cross isolines will be drawn and contouring that data is impossible.

For splot, data files may contain more than one mesh and by default all meshes are plotted. Meshes are separated from each other, in the file, by double blank lines. To control and splot a single mesh from a multi mesh file, use the index modifier. See splot index for more.

For splot if 3-d datafile and using format (see splot datafile using) specify only z (height field), a non parametric mode must be specified. If, on the other hand, x, y, and z are all specified, a parametric mode should be selected (see set parametric) since data is defining a parametric surface.

A simple example of plotting a 3-d data file is

```
set parametric
splot 'glass.dat'
```

or

```
set noparametric
splot 'datafile.dat'
```

where the file datafile.dat might contain:

```
# The valley of the Gnu.
10
10
10


10
5
10


10
1
10


10
0
10
```

Note datafile.dat defines a 4 by 3 grid (4 rows of 3 points each). Rows are separated by blank lines.

On some computer systems with a popen function (UNIX), the datafile can be piped through a shell command by starting the file name with a '<'. For example:

```
pop(x) = 103*exp(-x/10)
plot '< awk "{print $1-1965, $2}" population.dat', pop(x)
```

would plot the same information as the first population example but with years since 1965 as the x axis. If you want to execute this example, you have to delete all comments from the data file above or substitute the following command for the first part of the command above (the part up to the comma):

```
plot '< awk "$0 !~ /^#/ {print $1-1965, $2}" population.dat'
```

It is also possible to apply a single function to the "y" value only, e.g.

```
plot 'population.dat' thru p(x)
```

For more information about 3-d plotting, see splot.
using

using

The format of data within a file can be selected with the **using** option. An explicit scanf string can be used, or simpler column choices can be made.

Syntax:

```
        plot "datafile" { using { <ycol> |
                                  <xcol>:<ycol> |
                                  <xcol>:<ycol>:<ydelta> |
                                  <xcol>:<ycol>:<ylow>:<yhigh> |
                                  <xcol>:<ycol>:<ylow>:<yhigh>:<boxwidth> }
                                {"<scanf string>"} } ...
```

and

```
        splot "datafile" { using { <xcol>:<ycol>:<zcol> | <zcol> }
                                 {"<scanf string>"} } ...
```

<xcol>, <ycol>, and <zcol> explicitly select the columns to plot from a space or tab separated multicolumn data file. If only <ycol> is selected for plot, <xcol> defaults to 1. If only <zcol> is selected for splot, then only that column is read from the file. An <xcol> of 0 forces <ycol> to be plotted versus its coordinate number. <xcol>, <ycol>, and <zcol> can be entered as constants or expressions.

If errorbars (see also plot errorbars) are used for plots, ydelta (for example, a +/- error) should be provided as the third column, or ylow and yhigh as third and fourth columns.

If boxes or boxerrorbars are used for plots, a fifth column to specify the width of the box may be given.   This implies that columns three and four must also be provided even if they are not used. If you want to plot boxes from a data file with three columns, set ylow and yhigh to y using the following command:

```
        plot "datafile" using 1:2:2:2:3 with boxes
```

Scanf strings override any <xcol>:<ycol>(:<zcol>) choices, except for ordering of input, e.g.,

```
        plot "datafile" using 2:1 "%f%*f%f"
```

causes the first column to be y and the third column to be x.

If the scanf string is omitted, the default is generated based on the <xcol>:<ycol>(:<zcol>) choices. If the **using** option is omitted, "%f%f" is used for plot ("%f%f%f%f" for errorbars plots) and "%f%f%f" is used for splot.

Examples:

```
        plot "MyData" using "%*f%f%*20[^\n]%f" with lines
```

Data are read from the file "MyData" using the format "%*f%f%*20[^\n]%f". The meaning of this format is: "%*f" ignore the first number, "%f" then read in the second and assign to x, "%*20[^\n]" then ignore 20 non-newline characters, "%f" then read in the y value.

```
        n=3;
        plot "MyData", "MyData" using n
```

causes GNUPLOT to plot the second and third columns of MyData versus the first column.

The command 'n=4; replot' would then plot the second and fourth columns of MyData versus the first column.

```
        splot "glass.dat" using 1
```

causes GNUPLOT to plot the first coordinate of the points of glass.dat as the z coordinate while ignoring the other two coordinates.

Note: GNUPLOT first reads a line of the data file into a buffer and then does a

```
        sscanf(input_buffer, scanf_string, &x, &y{, &z});
```

where 'x', 'y', and 'z' are of type 'float'. Any scanf string that specifies two (three for splot, three or four for errorbars) float numbers may be used.

errorbars

Error bars are supported for 2-d data file plots by reading one or two additional columns specifying ydelta or ylow and yhigh respectively. No support exists for x error bars or any error bars for splots.

In the default situation, GNUPLOT expects to see three or four numbers on each line of the data file, either (x, y, ydelta) or (x, y, ylow, yhigh). The x coordinate must be specified. The order of the numbers must be exactly as given above. Data files in this format can easily be plotted with error bars:

```
        plot "data.dat" with errorbars
```

The error bar is a vertical line plotted from (x, ylow) to (x, yhigh). If ydelta is specified instead of ylow and yhigh, ylow=y-ydelta and yhigh=y+ydelta are derived. If there are only two numbers on the line, yhigh and ylow are both set to y. To get lines plotted between the data points, plot the data file twice, once with errorbars and once with lines.

If y autoscaling is on, the y range will be adjusted to fit the error bars.

The using option may be used to specify how columns of the data file are to be assigned to x, y, ydelta, ylow, and yhigh. The x column must be provided and both the x and y columns must appear before the errorbar columns. If three column numbers are given, they are x, y, and ydelta. If four columns are given, they are x, y, ylow, and yhigh.

Examples:

```
        plot "data.dat" using 1:2:3:4 with errorbars
        plot "data.dat" using 3:2:6 with errorbars
        plot "data.dat" using 3:4:8:7 with errorbars
```

The first example reads, x, y, ylow, and yhigh, from columns 1, 2, 3, and 4. This is equivalent to the default.   The second example reads x from the third column, y from second and ydelta from the sixth column. The third example reads x from the third column, y from the fourth, ylow from the eighth, and yhigh from seventh columns.

See also plot using and plot style.

parametric

When in parametric mode (set parametric) mathematical expressions must be given in pairs for plot and in triplets for splot:

```
        plot sin(t),t**2
```

or

```
        splot cos(u)*cos(v),cos(u)*sin(v),sin(u)
```

Data files are plotted as before, except any preceding parametric function must be fully specified before a data file is given as a plot. In other words, the x parametric function (sin(t) above) and the y parametric function (t**2 above) must not be interrupted with any modifiers or data functions; doing so will generate a syntax error stating that the parametric function is not fully specified.

Ranges take on a different meaning when in parametric mode. The first range on the plot command is the trange, the next is the xrange, and the last is the yrange. For splot the order is urange, vrange, xrange, yrange, and finally zrange. The following plot command shows setting the trange to [-pi:pi], the xrange to [-1.3:1.3] and the yrange to [-1:1] for the duration of the plot:

```
        plot [-pi:pi] [-1.3:1.3] [-1:1] sin(t),t**2
```

Other modifiers, such as with and title, may be specified only after the parametric function has been completed:

```
        plot sin(t),t**2 title 'Parametric example' with linespoints
```

ranges

The optional range specifies the region of the plot that will be displayed.

Ranges may be provided on the plot and splot command line and affect only that plot, or in the set xrange, set yrange, etc., commands, to change the default ranges for future plots.

Syntax:

```
        [{<dummy-var> =} {<xmin> : <xmax>}] { [{<ymin> : <ymax>}] }
```

where <dummy-var> is the independent variable (the defaults are x and y, but this may be changed with set dummy) and the min and max terms can be constant expressions.

Both the min and max terms are optional. The ':' is also optional if neither a min nor a max term is specified. This allows '[ ]' to be used as a null range specification.

Specifying a range in the plot command line turns autoscaling for that axis off for that plot. Using one of the set range commands turns autoscaling off for that axis for future plots, unless changed later. (See set autoscale).

Examples:

This uses the current ranges:

```
        plot cos(x)
```

This sets the x range only:

```
        plot [-10:30] sin(pi*x)/(pi*x)
```

This is the same, but uses t as the dummy-variable:

```
        plot [t = -10 :30]  sin(pi*t)/(pi*t)
```

This sets both the x and y ranges:

```
        plot [-pi:pi] [-3:3]  tan(x), 1/x
```

This sets only the y range, and turns off autoscaling on both axes:

```
        plot [ ] [-2:sin(5)*-8] sin(x)**besj0(x)
```

This sets xmax and ymin only:

```
        plot [:200] [-pi:]  exp(sin(x))
```

This sets the x, y, and z ranges:

```
        splot [0:3] [1:4] [-1:1] x*y
```

index

Splotting of multi mesh data files can be controlled via the index modifier. A data file can contain more than one mesh, and in that case all meshes in the file will be splotted by default. Meshes are separated from each other, in the data file, by double blank lines. To splot a single mesh in a multi mesh file use the index modifier which specify which mesh to splot. First mesh is mesh 0.

Example:

```
splot "data1" index 2 with points
```

will splot the third mesh in file data1 with points.

style

Plots may be displayed in one of eight styles: lines, points, linespoints, impulses, dots, errorbars, steps, boxes, or boxerrorbars.   The lines style connects adjacent points with lines. The points style displays a small symbol at each point. The linespoints style does both lines and points. The impulses style displays a vertical line from the x axis (or from the grid base for splot) to each point. The dots style plots a tiny dot at each point; this is useful for scatter plots with many points.

The errorbars style is only relevant to 2-d data file plotting. It is treated like points for splots and function plots. For data plots, errorbars is like points, except that a vertical error bar is also drawn: for each point (x,y), a line is drawn from (x,ylow) to (x,yhigh). A tic mark is placed at the ends of the error bar. The ylow and yhigh values are read from the data file's columns, as specified with the using option to plot. See plot errorbars for more information.

The boxes style is only relevant to 2-d plotting.   Another style called boxerrorbars is also available and is only relevant to 2-d data file plotting.   This style is a combination of the boxes and errorbars styles.   The boxes style draws a box centred about the given x coordinate from the yaxis to the given y coordinate. The width of the box is obtained in one of three ways.   First, if a   data file has a fifth column, this will be used to set the width of the box. Columns 3 and 4 (for boxerrorbars) are necessary but ignored in this instance. Secondly, if a width has been set using   the set boxwidth command, this will be used. Otherwise the width   of each box will be calculated automatically so that it touches the adjacent boxes.

The steps style is only relevant to 2-d plotting.   This style connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).

Default styles are chosen with the set function style and set data style commands.

By default, each function and data file will use a different line type and point type, up to the maximum number of available types. All terminal drivers support at least six different point types, and re-use them, in order, if more than six are required. The LaTeX driver supplies an additional six point types (all variants of a circle), and thus will only repeat after twelve curves are plotted with points.

If desired, the style and (optionally) the line type and point type used for a curve can be specified.

Syntax:

```
        with <style> {<linetype> {<pointtype>}}
```

where <style> is either lines, points, linespoints, impulses, dots, steps, or errorbars. The <linetype> and <pointtype> are positive integer constants or expressions and specify the line type and point type to be used for the plot. Line type 1 is the first line type used by default, line type 2 is the second line type used by default, etc.

Examples:

This plots sin(x) with impulses:
```
        plot sin(x) with impulses
```

This plots x*y with points, x**2 + y**2 default:
```
        splot x*y w points, x**2 + y**2
```

This plots tan(x) with the default function style, "data.1" with lines:

```
        plot [ ] [-2:5] tan(x), "data.1" with l
```

This plots "leastsq.dat" with impulses:

```
        plot 'leastsq.dat' w i
```

This plots the data file 'population' with boxes:

```
        plot "population" with boxes
```

This plots "exper.dat" with errorbars and lines connecting the points:

```
        plot 'exper.dat' w lines, 'exper.dat' w errorbars
```

Here 'exper.dat' should have three or four data columns.

This plots x**2 + y**2 and x**2 - y**2 with the same line type:

```
        splot x**2 + y**2 with line 1, x**2 - y**2 with line 1
```

This plots sin(x) and cos(x) with linespoints, using the same line type but different point types:

```
        plot sin(x) with linesp 1 3, cos(x) with linesp 1 4
```

This plots file "data" with points style 3:

```
        plot "data" with points 1 3
```

Note that the line style must be specified when specifying the point style, even when it is irrelevant. Here the line style is 1 and the point style is 3, and the line style is irrelevant.

See set style to change the default styles.

title

A title of each plot appears in the key. By default the title is the function or file name as it appears on the plot command line. The title can be changed by using the title option. This option should precede any with option.

Syntax:
```
        title "<title>"
```

where <title> is the new title of the plot and must be enclosed in quotes. The quotes will not be shown in the key.

Examples:

This plots y=x with the title 'x':
```
        plot x
```

This plots the "glass.dat" file with the title 'surface of revolution':
```
        splot "glass.dat" title 'surface of revolution'
```

This plots x squared with title "x^2" and "data.1" with title 'measured data':
```
        plot x**2 title "x^2", "data.1" t 'measured data'
```

The title can be omitted from the key with the "notitle" option for plot and splot.   This can be useful when some curves are plotted solely for decoration; for example, if one wanted a circular border for a polar plot, he could say:

Example:
```
       set polar
       plot my_function(x), 1 notitle
```

This would generate a key entry for "my_function" but not for "1". See the poldat.dem example.

print

The **print** command prints the value of <expression> to the screen.

Syntax:
```
        print <expression>
```

See expressions.

pwd

The **pwd** command prints the name of the working directory to the screen.

Syntax:

```
pwd
```

quit

The exit and quit commands and END-OF-FILE character will exit GNUPLOT. All these commands will clear the output device (as the clear command does) before exiting.

replot

The **replot** command without arguments repeats the last plot or splot command. This can be useful for viewing a plot with different set options, or when generating the same plot for several devices.

Arguments specified after a **replot** command will be added onto the last plot (splot) command (with an implied ',' separator) before it is repeated. **replot** accepts the same arguments as the plot (splot) commands except that ranges cannot be specified. See command line-editing for ways to edit the last plot (splot) command.

reread

The **reread** command causes the current gnuplot command file, as specified by a load command or on the command line, to be reset to its starting point before further commands are read from it.   This essentially implements an endless loop of the commands from the beginning of the command file to the **reread** command.   The **reread** command has no effect if input from standard input.

save

The **save** command saves user-defined functions, variables, set options or all three plus the last plot (splot) command to the specified file.

Syntax:

```
        save  {<option>} "<filename>"
```

where <option> is functions, variables or set. If no option is used, GNUPLOT saves functions, variables, set options and the last plot (splot) command.

saved files are written in text format and may be read by the load command.

The filename must be enclosed in quotes.

Examples:

```
        save "work.gnu"
        save functions 'func.dat'
        save var 'var.dat'
        save set "options.dat"
```

set-show

The **set** command sets LOTS of options.

The **show** command shows their settings. **show all** shows all the settings.
angles
arrow
autoscale
border
boxwidth
clabel
clip
cntrparam
contour
data style
dgrid3d
dummy
format
function style
functions
grid
hidden3d
isosamples
key
label
logscale
mapping
offsets
output
parametric
polar
rrange
samples
size
style
surface
terminal
tics
time
title
trange
urange
variables
view
vrange
xlabel
xrange
xtics
xdtics
xmtics
xzeroaxis
ylabel
yrange
ytics
ydtics

angles

By default, GNUPLOT assumes the independent variable in polar plots is in units of radians. If **set angles degrees** is specified before set polar then the default range is [0:360] and the independent variable has units of degrees. This is particularly useful for plots of data files. The angle setting also hold for the 3-d mapping as set via the set mapping command.

Syntax:

```
        set angles { degrees | radians }
        show angles
```

arrow

Arbitrary arrows can be placed on a plot using the **set arrow** command.

Syntax:

```
         set arrow {<tag>} {from <sx>,<sy>{,<sz>}}
                           {to <ex>,<ey>{,<ez>}} {{no}head}
         set noarrow {<tag>}
         show arrow
```

Unspecified coordinates default to 0. The x, y, and z values are in the graph's coordinate system. The z coordinate is only used in splot commands. <tag> is an integer that identifies the arrow. If no tag is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or change a specific arrow. To change any attribute of an existing arrow, use the **set arrow** command with the appropriate tag, and specify the parts of the arrow to be changed. Specifying nohead requests the arrow be drawn without a head (yielding a line segment). By default, arrows have heads.

Arrows outside the plotted boundaries are permitted but may cause device errors.

Examples:

To set an arrow pointing from the origin to (1,2), use:
```
         set arrow to 1,2
```
To set an arrow from (-10,4,2) to (-5,5,3), and tag the arrow number 3, use:
```
         set arrow 3 from -10,4,2 to -5,5,3
```
To change the preceding arrow begin at 1,1,1, without an arrow head, use:
```
         set arrow 3 from 1,1,1 nohead
```
To delete arrow number 2 use:
```
         set noarrow 2
```
To delete all arrows use:
```
         set noarrow
```
To show all arrows (in tag order) use:
```
         show arrow
```

autoscale

Auto scaling may be set individually on the x, y or z axis or globally on all axes. The default is to autoscale all axes.

When autoscaling, the plot range is automatically computed and the dependent axis (y for a plot and z for splot) is scaled to include the range of the function or data being plotted.

If autoscaling of the dependent axis (y or z) is not set, the current y or z range is used.

See set yrange or set zrange.

Autoscaling the independent variables (x for plot and x,y for splot) is a request to set the domain to match any data file being plotted. If there are no data files then autoscaling an independent variable has no effect. In other words, in the absence of a data file, functions alone do not affect the x range (or the y range if plotting z = f(x,y)).

See set xrange, or set yrange.

The behavior of autoscaling remains consistent in parametric mode, however, there are more dependent variables and hence more control over x, y, and z plot scales. In parametric mode, the independent or dummy variable is t for plots and u,v for splots.   Autoscale in parametric mode, then, controls all ranges (t, u, v, x, y, and z) and allows x, y, and z to be fully autoscaled.

See set parametric.

Syntax:

```
        set autoscale <axes>
        set noautoscale <axes>
        show autoscale
```

where <axes> is either **x**, **y**, **z** or **xy**. If <axes> is not given then all axes are assumed.

Examples:

This sets autoscaling of the y axis. x axis autoscaling is not affected.

```
        set autoscale y
```

This sets autoscaling of the x and y axes.

```
        set autoscale xy
```

This sets autoscaling of the x, y and z axes.

```
        set autoscale
```

This disables autoscaling of the x, y and z axes.

```
        set noautoscale
```

This disables autoscaling of the z axis only.

```
        set noautoscale z
```

parametric mode

parametric mode

When in parametric mode (set parametric) the xrange is as fully scalable as the yrange. In other words, in parametric mode the x axis can be automatically scaled to fit the range of the parametric function that is being plotted. Of course, the y axis can also be automatically scaled just as in the non-parametric case. If autoscaling on the x axis is not set, the current x range is used.

When there is a mix of data files and functions, the xrange of the functions is selected as that of the data files if autoscale is true for x. While this keeps the behavior compatible with non-parametric plotting, it may not be retained in the future. The problem is that, in parametric mode, the x and y ranges are not as distinguishable as in the non-parametric mode and this behavior may not be the most useful.

For completeness a last command **set autoscale t** is accepted. However, the effect of this "scaling" is very minor. When GNUPLOT determines that the t range would be empty it makes a small adjustment if autoscaling is true. Otherwise, GNUPLOT gives an error. Such behavior may, in fact, not be very useful and the command **set autoscale t** is certainly questionable.

splot extends the above idea similarly. If autoscaling is set then x, y, and z ranges are computed and each axis scaled to fit the resulting data.

border

The **set border** and **set noborder** commands controls the display of the plot borders for the plot and splot commands.

Syntax:
```
        set border
        set noborder
        show border
```

boxwidth

The **set boxwidth** command is used to set the default width of boxes in the boxes and boxerrorbars styles.

If a data file is plotted without the width being specified in the fifth column, or a function is plotted, the width of each box is set by the **set boxwidth** command.   If a width is given after the **set boxwidth** command then this is used as the width.   Otherwise the width of each box will be calculated automatically so that it touches the adjacent boxes.

Syntax:

```
        set boxwidth {<width>}
        show boxwidth
```

To set the box width to automatic use the command

```
        set boxwidth
```

clabel

GNUPLOT will vary the linetype used for each contour level when clabel is set. When this option on (the default), a legend labels each linestyle with the z level it represents.

Syntax:

```
        set clabel
        set noclabel
        show clabel
```

clip

GNUPLOT can clip data points and lines that are near the boundaries of a plot.

Syntax:
```
        set clip <clip-type>
        set noclip <clip-type>
        show clip
```

Three clip types are supported by GNUPLOT: points, **one**, and **two**. One, two, or all three clip types may be active for a single plot.

The points clip type forces GNUPLOT to clip (actually, not plot at all) data points that fall within but too close to the boundaries (this is so the large symbols used for points will not extend outside the boundary lines). Without clipping points near the boundaries may look bad; try adjusting the x and y ranges.

Setting the **one** clip type causes GNUPLOT to plot the line segments which have only one of the two endpoints within the plotting region. Only the in-range portion of the line is drawn. The alternative is to not draw any portion of the line segment.

Some lines may have both endpoints out of range, but pass through the plotting area. Setting the **two** clip-type allows the visible portion of these lines to be drawn.

In no case is a line drawn outside the plotting area.

The defaults are **noclip points**, **clip one**, and **noclip two**.

To check the state of all forms of clipping, use
```
        show clip
```

For backward compatibility with older versions, the following forms are also permitted.
```
       set clip
       set noclip
```
set clip is synonymous with **set clip points**. **set noclip** turns off all three types of clipping.

cntrparam

Sets the different parameters for the contouring plot (see also contour).

Syntax:
```
 set cntrparam { { linear | cubicspline | bspline } |
    points <n> |
    order <n>  |
    levels { [ auto ] <n> |
    discrete <z1>,<z2>, ... |
    incremental {<start>, <incr>{, <end>} } }
```

Examples:
```
    set cntrparam bspline
    set cntrparam points 7
    set cntrparam order 10
    set cntrparam levels auto 5                   # 5 automatic levels
    set cntrparam levels discrete .1,1/exp(1),.9  # 3 discrete at .1,.37,.9
    set cntrparam levels incremental  0,.1,.4
    # 5 incremental levels at 0, .1, .2, .3 and .4
    set cntrparam levels 10
    # sets n = 10 retaining current setting of auto, discr. and
    # increment's start and increment value, while changing end
    set cntrparam levels incremental 100,50
    # set start = 100 and increment = 50, retaining n levels
```

This command controls the way contours are plotted. <n> should be an integral constant expression and <z1>, <z2> any constant expressions. The parameters are:

linear, **cubicspline**, **bspline** - Controls type of approximation or interpolation. If **linear**, then the contours are drawn piecewise linear, as extracted from the surface directly. If **cubicspline**, then piecewise linear contours are interpolated to form a somewhat smoother contours, but which may undulate. The third option is the uniform **bspline**, which only approximates the piecewise linear data but is guaranteed to be smoother.

points - Eventually all drawings are done with piecewise linear strokes.   This number controls the number of points used to approximate a curve.   Relevant for **cubicspline** and **bspline** modes only.

order   - Order of the bspline approximation to be used. The bigger this order is, the smoother the resulting contour.   (Of course, higher order bspline curves will move further away from the original piecewise linear data.)   This option is relevant for **bspline** mode only. Allowed values are integers in the range from 2 (linear) to 10.

levels - Number of contour levels, 'n'.   Selection of the levels is controlled by 'auto' (default), 'discrete', and 'incremental'. For 'auto', if the surface is bounded by zmin and zmax then contours will be generated from zmin+dz to zmax-dz in steps of size dz, where dz = (zmax - zmin) / (levels + 1).   For 'discrete', contours will be generated at z = z1, z2 ... as specified.   The number of discrete levels is limited to MAX_DISCRETE_LEVELS, defined in plot.h to be 30.   If 'incremental', contours are generated at <n> values of z beginning at <start> and increasing by <increment>.

contour

Enable contour drawing for surfaces. This option is available for splot only.

Syntax:
```
        set contour { base | surface | both }
        set nocontour
```

If no option is provided to **set contour**, the default is **base**. The three options specify where to draw the contours: **base** draws the contours on the grid base where the x/ytics are placed, surface draws the contours on the surfaces themselves, and **both** draws the contours on both the base and the surface.

See also set cntrparam for the parameters that affect the drawing of contours.

data style

The **set data style** command changes the default plotting style for data plots.

Syntax:

```
        set data style
        show data style
        set data style <style-choice>
```

In the first case, **set data style** returns the possible style choices: lines, points, linespoints, dots, steps, impulses, errorbars, boxes or boxerrorbars. **show data style** shows the current default plotting style for data. **set data style dots** would actually change the default plotting style. See also plot.

dgrid3d

Enables and sets the different parameters for non grid to grid data mapping.

Syntax:

```
        set dgrid3d {,{<row_size>}{,{<col_size>}{,<norm>}}}
        set nodgrid3d
```

Examples:

```
       set dgrid3d 10,10,2
       set dgrid3d ,,4
```

The first selects a grid of size 10 by 10 to be constructed and the use of L2 norm in the distance computation. The second only modifies the norm to be used to L4.

By default this option is disabled. When enabled, 3d data read from a file is always treaded as a scattered data set. A grid with dimensions derived from a bounding box of the scattered data and size as specified by the row/col_size above is created for plotting and contouring. The grid is equally spaced in x and y while the z value is computed as a weighted average of the scattered points distance to the grid points. The closer the scatter points to a grid point are the more effect they have on that grid point. The third, norm, parameter controls the "meaning" of the distance, by specifying the distance norm. This distance computation is optimized for powers of 2 norms, specifically 1, 2, 4, 8, and 16, but any nonnegative integer can be used.

This dgrid3d option is a simple low pass filter that converts scattered data to a grid data set. More sophisticated approaches to this problem exists and should be used as a preprocess to and outside gnuplot if this simple solution is found inadequate.

dummy

By default, GNUPLOT assumes that the independent variable for the plot command is x, and the independent variables for the splot command are x and y. They are called the dummy variables because it is just a notation to indicate the independent variables. The **set dummy** command changes these default dummy variable names. For example, it may be more convenient to call the dummy variable t when plotting time functions:

```
        set dummy t
        plot sin(t), cos(t)
```

Syntax:

```
        set dummy <dummy-var>{,<dummy-var>}
        show dummy
```

Examples:

```
        set dummy u,v
        set dummy ,s
```

to set both dummy variables to u and v or set only the second variable to s.

The set parametric command also changes the dummy variables (to t for plot and u,v for splots).

format

The format of the tic-mark labels can be set with the **set format** command. The default format for both axes is "%g", but other formats such as "%.2f" or "%3.0fm" are often desirable. Anything accepted by printf when given a double precision number, and then accepted by the terminal, will work. In particular, the formats f, e, and g will work, and the d, o, x, c, s, and u formats will not work.

Syntax:

```
        set format {<axes>} {"<format-string>"}
        show format
```

where <axes> is either **x**, **y**, **z**, **xy**, or nothing (which is the same as **xy**). The length of the string representing a ticmark (after formatting with printf) is restricted to 100 characters.   If the format string is omitted, the format will be returned to the default "%g". For LaTeX users, the format "$%g$" is often desirable.   If the empty string "" is used, no label will be plotted with each tic, though the tic mark will still be plotted. To eliminate all tic marks, use set noxtics or set noytics.

See also set xtics and set ytics for more control over tic labels.

function style

The **set function style** command changes the default plotting style for functions.

Syntax:

```
        set function style
        show function style
        set function style <style-choice>
```

In the first case, **set function style** returns the possible style choices: lines, points, linespoints, dots, steps, impulses, errorbars, boxes, or boxerrorbars. **show function style** shows the current default plotting style for functions. **set function style linespoints** would actually change the default plotting style. See also plot.

functions

The **show functions** command lists all user-defined functions and their definitions.

Syntax:

```
        show functions
```

grid

The optional **set grid** draws a grid at the tic marks with the axis linetype.

Syntax:
```
        set grid
        set nogrid
        show grid
```

hidden3d

The **set hidden3d** command enables hidden line removal for explicit surface plotting (see splot). Hidden line removal may be used for both explicit functions and for explicit data.   It now works for parametric surfaces as well.

When this flag is set both the surface hidden portion and possibly its hidden contours (see set contour) as well as the hidden grid will be removed. Labels and arrows are always visible and are unaffected by this command.

Each surface has its hidden parts removed with respect to itself and to other surfaces, if more than one surface is plotted. This mode is meaningful when surfaces are plotted using line style drawing only.

Syntax:
```
        set hidden3d
        set nohidden3d
        show hidden3d
```

isosamples

An isoline is a curve parametrized by one of the surface parameters while the other surface parameter is fixed. Isolines are a simple means to display a surface. By fixing the u parameter of surface s(u,v), the iso-u lines of the form c(v) = s(u0,v) are produced, and by fixing the v parameter, the iso-v lines of the form c(u) = s(u,v0) are produced.

The isoline density of surfaces may be changed by the **set isosamples** command. By default, sampling is set to 10 isolines per u or v axis. A higher sampling rate will produce more accurate plots, but will take longer. This parameter has no effect on data file plotting.

Syntax:
```
        set isosamples <iso_1> {,<iso_2>}
        show isosamples
```

Each surface plot will have <iso_1> iso-u lines and <iso_2> iso-v lines. If you only specify <iso_1>, <iso_2> will be set to the same value as <iso_1>.

When a surface plot is being done without the removal of hidden lines, set samples also has an effect on the number of points being evaluated. See set samples.

key

The **set key** enables a key describing curves on a plot.   By default the key is placed in the upper right corner of the plot.

Syntax:

```
        set key
        set key <x>,<y>{,<z>}
        set nokey
        show key
```

The coordinates <x>, <y> (and <z> for splots) specify the location of the key on the plot. The key is drawn as a sequence of lines, with one plot described on each line. On the right hand side of each line is a representation that attempts to mimic the way the curve is plotted.   On the left side of each line is the text description, obtained from the plot command. See plot title to change this description. The lines are vertically arranged so an imaginary straight line divides the left- and right-hand sides of the key. It is the coordinates of this line that are specified with the **set key** command. In a plot, only the x and y coordinates are used to specify the line position.   For a splot, x, y and z are all being used as a 3-d location mapped using the same mapping as the plot itself to form the required 2-d screen position of the imaginary line.

Some or all of the key may be outside of the plot boundary, although this may interfere with other labels and may cause an error on some devices.

Examples:

This places the key at the default location:

```
        set key
```

This disables the key:

```
        set nokey
```

This places a key at coordinates 2,3.5,2

```
        set key 2,3.5,2
```

label

Arbitrary labels can be placed on the plot using the **set label** command. If the z coordinate is given on a plot it is ignored; if it is missing on a splot it is assumed to be 0.

Syntax:

```
set label {<tag>} {"<label_text>"} {at <x>,<y>{,<z>}}
                  {<justification>}
set nolabel {<tag>}
show label
```

The text defaults to "", and the position to 0,0,0. The <x>, <y>, and <z> values are in the graph's coordinate system. The tag is an integer that is used to identify the label. If no <tag> is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or change a specific label. To change any attribute of an existing label, use the **set label** command with the appropriate tag, and specify the parts of the label to be changed.

By default, the text is placed flush left against the point x,y,z. To adjust the way the label is positioned with respect to the point x,y,z, add the parameter <justification>, which may be **left**, **right** or **center**, indicating that the point is to be at the left, right or center of the text. Labels outside the plotted boundaries are permitted but may interfere with axes labels or other text.

Examples:

To set a label at (1,2) to "y=x" use:

```
set label "y=x" at 1,2
```

To set a label "y=x^2" with the right of the text at (2,3,4), and tag the label number 3, use:

```
set label 3 "y=x^2" at 2,3,4 right
```

To change the preceding label to center justification, use:

```
set label 3 center
```

To delete label number 2 use:

```
set nolabel 2
```

To delete all labels use:

```
set nolabel
```

To show all labels (in tag order) use:

```
show label
```

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

logscale

Log scaling may be set on the x, y, and z axes.

Syntax:
```
        set logscale <axes> <base>
        set nologscale <axes>
        show logscale
```

where <axes> may be any combinations of **x**, **y**, and **z**, in any order, and where <base> is the base of the log scaling. If <base> is not given, then 10 is assumed. If <axes> is not given then all three axes are assumed. The command **set logscale** turns on log scaling on the specified axes, while **set nologscale** turns off log scaling.

Examples:

To enable log scaling in both x and z axes:
```
       set logscale xz
```
To enable scaling log base 2 of the y axis:
```
       set logscale y 2
```
To disable z axis log scaling:
```
       set nologscale z
```

mapping

Syntax:

```
        set mapping { cartesian | spherical | cylindrical }
```

Data for splots are usually in regular Euclidean space and are provided in Cartesian coordinates. Such 3-d data require three coordinates (x, y and z) or one coordinate (only z) in each line in the data file.   In order to be able to use spherical or cylindrical coordinate systems, use the **set mapping** command. In both cases two coordinates are expected in each line of the data. For a spherical coordinate system, these are theta and phi (in units as specified by set angles) and the mapping is:

```
        x = cos( theta ) * cos( phi )
        y = sin( theta ) * cos( phi )
        z = sin( phi )
```

For a cylindrical coordinate system, the mapping uses two variables, theta (in units as specified by set angles) and z:

```
        x = cos( theta )
        y = sin( theta )
        z = z
```

Again, note that mapping will affect data file splots only.

offsets

The amount of the graph that the plot takes up may be controlled to some extent with the **set offsets** command. This command takes four offset arguments: <left>, <right>, <top> and <bottom>. By default, each offset is 0. Each offset may be a constant or an expression. Left and right offsets are given in units of the x axis, while top and bottom offsets are given in units of the y axis. The plot of sin(x), displayed with offsets of 0, 0, 2, 2 will take up 1/3 of the displayed y axis. Offsets are particularly useful with polar coordinates as a means of compensating for aspect ratio distortion. Offsets are ignored in splots.

Syntax:
```
        set offsets <left>, <right>, <top>, <bottom>
        show offsets
```

output

By default, plots are displayed to the standard output. The **set output** command redirects the display to the specified file or device.

Syntax:
```
        set output {"<filename>"}
        show output
```

The filename must be enclosed in quotes. If the filename is omitted, output will be sent to the standard output.

On machines with popen functions (UNIX), output can be piped through a shell command if the first letter of the filename is '|'.   For instance,

Syntax:
```
        set output "|lpr -Plaser filename"
        set output "|lp -dlaser filename"
```

(On MSDOS machines, set output "prn" will direct the output to the default printer.)

parametric

The **set parametric** command changes the meaning of plot (splot) from normal functions to parametric functions. The command **set noparametric** changes the plotting style back to normal, single-valued expression plotting.

In 2-d plotting, a parametric function is determined by a pair of parametric functions operating on a parameter. An example of a 2-d parametric function would be plot sin(t),cos(t) (which defines a circle).

For 3-d plotting, the surface is described as x=f(u,v), y=g(u,v), z=h(u,v). Therefore a triplet of functions are required. An example of 3-d parametric function would be cos(u)*cos(v),cos(u)*sin(v),sin(u) (which defines a sphere). It takes three parametric function specifications in terms of the parametric dummy arguments to describe a single graph.

The total set of possible plots is a superset of the simple f(x) style plots, since the two (three) functions can describe the x and y (and z) values to be computed separately. In fact, plots of the type t,f(t) (u,v,f(u,v)) are equivalent to those produced with f(x) when the x values are computed using the identity function as the first function.

Note that the order the parametric functions are specified is xfunction, yfunction (and zfunction) and that each operates over the common parametric domain.

Also, the **set parametric** function implies a new range of values. Whereas the normal f(x) and f(x,y) style plotting assume an xrange and yrange (and zrange), the parametric mode additionally specifies a trange, urange, and vrange. These ranges may be set directly with set trange, set urange and set vrange, or by specifying the range on the plot or splot commands. Currently the default range for these parametric variables is [-5:5]. Setting the ranges to something more meaningful is expected.

polar

The **set polar** command changes the meaning of the plot from rectangular coordinates to polar coordinates. In polar coordinates, the dummy variable (x) is an angle. The range of this angle is changed from whatever it was to [0:2*pi], or, if degree unit has been selected, to [0:360] (see set angles).

The command **set nopolar** changes the meaning of the plot back to the default rectangular coordinate system. The range of x is changed from whatever it was to [-10:10].

The **set polar** command is not supported for splots.   See the set mapping command for similar functionality for splots.

While in polar coordinates the meaning of an expression in x is really r = f(x), where x is an angle of rotation. The xrange controls the domain (the angle) of the function, and the yrange controls the range (the radius). The plot is plotted in a rectangular box, and the x and y axes are both in units of the radius. Thus, the yrange controls both dimensions of the plot output. The tics and units are written along the axes rather than at the left and bottom. These unit are offset by <rmin> specified by the rrange (See set rrange). It is not possible to specify different output dimensions in the x or y directions. The yrange can be used to shift the plot diagonally to display only the first or third quadrants.

Syntax:
```
        set polar
        set nopolar
        show polar
```
Example:
```
        set polar
        plot x*sin(x)
        plot [-2*pi:2*pi] [-3:3] x*sin(x)
```
The first plot uses the default polar angular domain of 0 to 2*pi. The radius (and the size of the plot) is scaled automatically. The second plot expands the domain, and restricts the range of the radius (and the size of the plot) to [-3:3].

rrange

The **set rrange** command sets the radial range used to compute x and y values when in polar mode. If not in polar mode (see set polar) then this range is not used. Use of this command offsets the polar singularity to the <rmin> value and shifts the units on the axes tic marks. For instance, **set rrange [-40:40]** would set the origin to -40 and would plot values of radial values between -40 to 40. Thus, if 360 degrees of data were plotted, then the plot would extend 80 units in radially from the origin.   To view the entire plot,   a **set yrange [-80:80]** command would create a square viewport with a circular plot tangent at the axes.   Because xrange is used specify the angular extent, only a square viewport can be specified by yrange.   For instance, **set yrange [0:80]** would display the first quadrant and **set yrange [-80:0]** would display the third quadrant.   Any square viewport of any size can be specified but it is constrained to be centered on a 45 degree line.

This range may also be specified on the plot command line when in polar mode.

Syntax:

```
        set rrange [{<rmin> : <rmax>}]
```

where <rmin> and <rmax> terms are constants or expressions.

Both the <rmin> and <rmax> terms are optional. Anything omitted will not be changed, so

```
        set rrange [:10]
```

changes rmax to 10 without affecting rmin.

samples

The sampling rate of functions may be changed by the **set samples** command. By default, sampling is set to 100 points. A higher sampling rate will produce more accurate plots, but will take longer. This parameter no longer has any effect on data-file plotting.

Syntax:

```
        set samples <samples_1> {,<samples_2>}
        show samples
```

When a 2-d plot is being done, only the value of <samples_1> is relevant.

When a surface plot is being done without the removal of hidden lines, the value of samples specifies the number of samples that are evaluated per iso line. Each iso-v line will have <sample_1> samples and each iso-u line will have <sample_2> samples. If you only specify <samples_1>, <samples_2> will be set to the same value as <samples_1>. See also set isosamples.

size

The **set size** command scales the displayed size of the plot.   On some terminals, changing the size of the plot will result in text being misplaced. Increasing the size of the plot may produce strange results. Decreasing is safer.

Syntax:

```
        set size {<xscale>,<yscale>}
        show size
```

The <xscale> and <yscale> values are the scaling factors for the size. The defaults (1,1) are selected if the scaling factors are omitted.

Examples:

To set the size to normal size use:
```
        set size
```
To make the plot half size use:
```
        set size 0.5,0.5
```
To make a landscape plot have a 1:1 aspect ratio in polar mode use:
```
        set size 0.721,1.0
```
To show the size use:
```
        show size
```

For the LaTeX and Fig terminals the default size (scale factor 1,1) is 5 inches wide by 3 inches high. The big Fig terminal (**bfig**) is 7 inches wide by 5 inches high. The postscript default is landscape mode 10 inches wide and 7 inches high. Note that the size of the plot includes the space used by the labels; the plotting area itself is smaller.

style

Plots may be displayed in one of eight styles: **lines**, **points**, **linespoints**, **impulses**, **dots**, **steps**, errorbars, **boxes**, or **boxerrorbars**.   The **lines** style connects adjacent points with lines. The **points** style displays a small symbol at each point. The **linespoints** style does both **lines** and **points**. The **impulses** style displays a vertical line from the x axis (or from the grid base for splot) to each point. The **dots** style plots a tiny dot at each point; this is useful for scatter plots with many points.

The errorbars style is only relevant to 2-d data file plotting. It is treated like **points** for splots and function plots. For data plots, errorbars is like **points**, except that a vertical error bar is also drawn: for each point (x,y), a line is drawn from (x,ylow) to (x,yhigh). A tic mark is placed at the ends of the error bar. The ylow and yhigh values are read from the data file's columns, as specified with the using option to plot. See plot errorbars for more information.

The **boxes** style is only relevant to 2-d plotting.   It draws a box centred about the given x coordinate from the yaxis to the given y coordinate.   The width of the box is obtained in one of three ways.   If a data file has a fifth column, this will be used to set the width of the box.   Otherwise, if a width has been set using the set boxwidth command, this will be used. Otherwise the width of each box will be calculated automatically so that it touches the adjacent boxes.   Another style called **boxerrorbars** is also available and is only   relevant to 2-d data file plotting.   This style is a combination of the **boxes** and errorbars styles.

The **steps** style is only relevant to 2-d plotting.   This style connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).

Default styles are chosen with the set function style and set data style commands. See plot style for information about how to override the default plotting style for individual functions.

Syntax:
```
        set function style <style>
        set data style <style>
        show function style
        show data style
```

where <style> is **lines**, **points**, **linespoints**, **impulses**, **dots**, **steps**, errorbars, **boxes**, or **boxerrorbars**.

surface

set surface controls the display of surfaces. It is useful if contours are to be displayed by themselves. Whenever **set nosurface** is issued, no surface isolines/mesh will be drawn. See also set contour.

Syntax:

```
        set surface
        set nosurface
        show surface
```

terminal

GNUPLOT supports many different graphics devices. Use the **set terminal** command to select the type of device for which GNUPLOT will produce output.

Syntax:
```
        set terminal {<terminal-type>}
        show terminal
```

If <terminal-type> is omitted, GNUPLOT will list the available terminal types. <terminal-type> may be abbreviated.

Use set output to redirect this output to a file or device.

Several terminals have additional options. For example, see dumb, iris4d, hpljii or postscript.
aifm
atari ST
dumb
epson
gpic
hpljii
latex
iris4d
mf
mif
nec-cp6
pbm
pcl5
postscript
regis
table
windows

aifm

Several options may be set in the Adobe Illustrator 3.0 driver.

Syntax:
```
        set terminal aifm {<color>}
                                {"<fontname>"} {<fontsize>}
```

Selecting default sets all options to their default values. <color> is either **color** or **monochrome**. "<fontname>" is the name of a valid PostScript font. <fontsize> is the size of the font in PostScript points, before scaling by the set size command. Defaults are **monochrome**, "Helvetica", and 14pt.

Also, since AI does not really support multiple pages, multiple graphs will be output directly on one another.   However, each graph will be grouped individually, making it easy to separate them inside AI (just pick them up and move them).

Examples:

```
        set term aifm
        set term aifm 22
        set size 0.7,1.4
        set term aifm color "Times-Roman" 14
```

atari ST

The **atari** terminal has an option to set the character size and the screen colors.   The driver expects a space separated list the char size and maximal 16 3 digit hex numbers where each digit represents RED, GREEN and BLUE (in that order).   The range of 0-15 is scaled to whatever color range the screen actually has.   On a normal ST screen, odd and even intensities are the same.

Examples:

```
set terminal atari 4 # (use small (6x6) font)
set terminal atari 6 0 # (set monochrome screen to white on black)
set terminal atari 13 0 fff f00 f0 f ff f0f ff0
# (set first eight colors to black, white, green, blue, cyan, \
  purple, and yellow and use large font (8x16).)
```

Additionally, if an environment variable GNUCOLORS exists, its contents are interpreted as an options string, but an explicit terminal option takes precedence.

dumb

The dumb terminal driver has an optional size specification.

Syntax:
```
        set terminal dumb {<xsize> <ysize>}
```

where <xsize> and <ysize> set the size of the dumb terminals. Default is 79 by 24.

Examples:
```
        set term dumb
        set term dumb 79 49 # VGA screen--why would anyone want to do that?
```

epson

This set of drivers support Epson printers and derivatives. See also the NEC driver. **epson** is a generic 9 wire printer with a resolution of 512x384. **starc** is a Star Color printer with the same resolution. **epson180** and **epson60** are 180 dpi and 60 dpi drivers for newer 24 wire printers. This also includes bubble jet printers. Their resolutions are 1260x1080 and 480x360, respectively. The **tandy60** is identical to the **epson60** driver with one additional escape sequence to start IBM mode. With all of these drivers, a binary copy is required on a PC to print. Do not use print.

```
        copy file /b lpt1:
```

gpic

This driver is only known to work the Free Software Foundation gpic/groff package. Modification for the Document Workbench pic/troff package would be appreciated. FSF gpic can also produce TeX output.

A simple graph can be formatted using

```
     groff -p -mpic -Tps file.pic > file.ps.
```

The output from pic can be pipe-lined into eqn, so it is possible to put complex functions in a graph with the set label and set {x/y}label commands. For instance,

```
   set ylab '@space 0 int from 0 to x alpha ( t ) roman d t@'
```

Will label the y-axis with a nice integral if formatted with the command:

```
   gpic filename.pic | geqn -d@@ -Tps | groff -m[macro-package] -Tps
        > filename.ps
```

Figures made this way can be scaled to fit into a document. The pic language is easy to understand, so the graphs can be edited by hand if need be. All coordinates in the pic-file produced by gnuplot are given as x+gnuplotx and y+gnuploty. By default x and y are given the value 0 If this line is removed with an editor in a number of files one can put several graphs i one figure like this (default size is 5.0x3.0 inches)

```
   .PS 8.0
   x=0;y=3
   copy "figa.pic"
   x=5;y=3
   copy "figb.pic"
   x=0;y=0
   copy "figc.pic"
   x=5;y=0
   copy "figd.pic"
   .PE
```

This will produce an 8 inches wide figure with four graphs in two rows on top of each other

One can also achieve the same thing by the command

```
  set term pic x y
```

For example, using

```
  .PS 6.0
  copy "trig.pic"
  .PE
```

hpljii

The HP LaserJet II and HP DeskJet drivers have a single option.

Syntax:
```
        set terminal hpljii {<resolution>}
        set terminal hpdj   {<resolution>}
```

where <resolution> is the resolution of the output in dots per inch. It must be **75**, **100**, **150** or **300**.   Note: there must be enough memory available to rasterize at the higher resolutions.

Example:
```
        set terminal hpljii 150
```

latex

The LaTeX and EMTeX driver allows one to specify a font type and a font size for the labels around a gnuplot graph.

Options are: Fonts:

```
      default          (Roman 10 point)
      courier
      roman
```

at any size you specify. (BEWARE METAFONT will not like odd sizes.) eg.

```
      gnuplot> set term latex courier 5
```

Unless your driver is capable of building fonts at any size (e.g. dvips), stick to the standard 10, 11 and 12 point size.

iris4d

The iris4d driver can operate in two modes.

Syntax:
```
        set terminal iris4d {24}
```

If the hardware supports only 8 bits, use the default **set terminal iris4d**. If, however, the hardware supports 24 bits (8 per red/green/blue), use **set terminal iris4d 24**.

When using 24-bit mode, the colors can be directly specified via the file .gnuplot_iris4d that is searched in the current directory and then in the home directory specified by the HOME environment variable. This file holds RGB values for the background, border, labels and nine plotting colors, in that order. For example, here is a file containing the default colors:

```
        85   85   85     /* Back Ground */
        0    0    0      /* Boundary */
        170  0    170    /* Labeling */
        85   255  255    /* Plot Color 1 */
        170  0    0      /* Plot Color 2 */
        0    170  0      /* Plot Color 3 */
        255  85   255    /* Plot Color 4 */
        255  255  85     /* Plot Color 5 */
        255  85   85     /* Plot Color 6 */
        85   255  85     /* Plot Color 7 */
        0    170  170    /* Plot Color 8 */
        170  170  0      /* Plot Color 9 */
```

This file has exactly 12 lines of RGB triples. No empty lines are allowed and anything after the third number in line is ignored.

mf

The mf terminal driver creates a input file to the MetaFont program. Thus a figure may be used in the TeX document in the same way as a character is.

To use the plot in a document the MetaFont program must be run with the output file from GnuPlot as input. Thus, the user needs a basic knowledge of the font creating process and inclusion of a new font in a document. However, if the Metafont program is set up properly at the local site an unexperienced user could perform the operation without much trouble.

The text support is based on a MetaFont character set. Currently the Computer Modern Roman font set is input but the user are in principal free to chose whatever fonts he/she needs. The MetaFont source files for the chosen font must be available. Each character is stored in a separate picture variable in MetaFont. These variables may be manipulated (rotated, scaled etc.) when characters are needed. The drawback is the interpretation time in the MetaFont program. On some machines (i.e. PC) the limited amount of memory available may also cause problem if too many pictures are stored.

Metafont Instructions

Metafont Instructions

- Set your terminal to metafont:
```
  set terminal mf
```
- Select an output-file, e.g.:
```
  set output "myfigures.mf"
```
- Do your plots. Each plot will generate a separate character. Its default size will be 5*3 inches. You can change the size by saying set size 0.5,0.5 or whatever fraction of the default size you want to have.

- Quit gnuplot.

- Generate a tfm- and gf-file by running metafont on the output of gnuplot. Since the plot is quite large (5*3 in), you will have to use a version of metafont that has a value of at least 150000 for memmax. On Unix-systems these are conventionally installed under the name bigmf. For the following assume that the command virmf stands for a big version of metafont. For example:

- Invoke metafont:
```
    virmf '&plain'
```
- Select the output device: At the metafont prompt ('*') type:
```
    \mode:=CanonCX;     % or whatever printer you use
```
- Optionally select a magnification:
```
    mag:=1;             % or whatever you wish
```
- Input the gnuplot-file:
```
    input myfigures.mf
```
On a typical Unix machine there will usually be a script called mf that executes virmf '&plain', so you probably can substitute mf for virmf &plain. This will generate two files: mfput.tfm and mfput.$$$gf (where $$$ indicates the resolution of your device). The above can be conveniently achieved by typing everything on the command line, e.g.: virmf '&plain' '\mode:=CanonCX; mag:=1; input myfigures.mf' In this case the output files will be named myfigures.tfm and myfigures.300gf.

- Generate a pk-file from the gf-file using gftopk:
```
  gftopk myfigures.300gf myfigures.300pk
```
The name of the output-file for gftopk depends on the dvi-driver you use. Ask your local TeX-administrator about the naming conventions. Next, either install the tfm- and pk-files in the appropriate directories, or set your environment-variables properly. Usually this involves setting TEXFONTS to include the current directory and do the same thing for the environment-variable that your dvi-driver uses (no standard name here...). This step is necessary so that TeX will find the font-metric file and your dvi-driver will find the pk-file.

- To include your plots in your document you have to tell TeX the font:
```
  \font\gnufigs=myfigures
```
Each plot you made is stored in a single character. The first plot is character 0, the second is character 1, and so on... After doing the above step you can use the plots just like any other characters. Therefore, to place plots 1 and 2 centered in your document, all you have to do is:
```
  \centerline{\gnufigs\char0}
  \centerline{\gnufigs\char1}
```
in plain TeX. For LaTeX you can, of course, use the picture environment and place the plot according to your wishes using the \makebox and \put macros.

It saves you a lot of time, once you have generated the font, since TeX handles the plots as characters and uses minimal time to place them. Also the documents you make change more often, than the plots do. Also it saves a lot of TeX-memory. One last advantage of using the metafont-driver is that the dvi-file really remains device independent, because no \special-commands are used as in the eepic- and tpic-drivers.

mif

Several options may be set in the MIF 3.00 driver.

Syntax:
```
     set terminal mif {<pentype>} {<curvetype>} {<help>}
```

<pentype> selects "colour" of the graphics.
```
     `colour`     plot lines with line types >= 0 in colour (MIF sep. 2-7).
     `monochrome` plot all line types in black (MIF sep. 0).
```
<curvetype> selects how "curves" are plotted.
```
     `polyline`   plot curves as continuous curves.
     `vectors`    plot curves as collections of vectors
```
<help> print online help on standard error output.
```
     `help`       print a short description of the usage, and the options
     `?`          print a short description of the usage
```

This terminal driver produces Frame Maker MIF format version 3.00. It plots in MIF Frames with the size 15*10 [cm], and plot primitives with the same pen will be grouped in the same MIF group. Plot primitives in a gnuplot plot will be plotted in a MIF Frame, and several MIF Frames are collected in one large MIF Frame. Plot primitives with line types >= 0 will as default be drawn in colour. As default curves are plotted as continuous lines. The MIF font used for text is "Times".

Examples:

```
     set term mif
     set term mif vectors
     set term mif help
```

nec-cp6

One option may be set in the nec-cp6 driver.   The resolution of this driver is 400x320.

Syntax:

```
        set terminal nec-cp6 monochrome
        set terminal nec-cp6 color
        set terminal nec-cp6 draft
```

pbm

Several options may be set in the PBMplus driver.

Syntax:
```
        set terminal pbm {<fontsize>} {<colormode>}
```

where <fontsize> is **small**, **medium**, or **large** and <colormode> is **monochrome**, **gray** or **color**. Default size is 640 pixels wide and 480 pixels high. The output for **monochrome** is a portable bitmap (one bit per pixel). The output for **gray** is a portable graymap (three bits per pixel). The output for **color** is a portable pixmap (color, four bits per pixel). The output of these drivers can be used with Jef Poskanzer's excellent PBMPLUS package which provides programs to convert the above PBMPLUS formats to GIF, TIFF, MacPaint, Macintosh PICT, PCX, X11 bitmap and many others.

Examples:

```
        set term pbm small
        set size 2,2
        set term pbm color medium
```

pcl5

Three options may be set in the pcl5 driver.   The driver actually uses HPGL-2 but there is a name conflict among the terminal devices.

Syntax:
```
        set terminal pcl5 {<mode>} {<font>} {<fontsize>}
```

where <mode> is **landscape**, or **portrait**, <font> is **stick**, **univers**, or **cg_times**, and fontsize is the size in points.
```
        set terminal pcl5 landscape
```

postscript

Several options may be set in the PostScript driver.

Syntax:
```
        set terminal postscript {<mode>} {<color>} {<dashed>}
                                {"<fontname>"} {<fontsize>}
```

where <mode> is **landscape**, **portrait**, **eps** or **default**. Selecting default sets all options to their defaults. <color> is either **color** or **monochrome**. <dashed> is either **solid** or **dashed**. "<fontname>" is the name of a valid PostScript font. <fontsize> is the size of the font in PostScript points. Defaults are **landscape**, **monochrome**, **dashed**, "Helvetica", and 14pt. Default size of PostScript plot is landscape mode 10 inches wide and 7 inches high.

To get EPS output, use the **eps** mode and make only one plot per file. In **eps** mode the whole plot is halved in size; the fonts are half the given size, and the plot is 5 inches wide and 3.5 inches high.

Examples:

```
        set term postscript default       # old postscript
        set term postscript landscape 22  # old psbig
        set term postscript eps 14   # old epsf1
        set term postscript eps 22   # old epsf2
        set size 0.7,1.4
        set term post portrait color "Times-Roman" 14
```

regis

The **regis** terminal device has the option of using 4 or 16 colors.   The default is 4.   For example:

```
      set term regis 16
```

table

Instead of producing a picture, term type table prints out the evaluation results in a multicolumn ASCII table of X Y Z values. For those times when you really want to see the numbers, now you can see them on the screen or save to a file.

windows

Three options may be set in the windows driver.

Syntax:
```
        set terminal windows {<color>} {"<fontname>"} {<fontsize>}
```

<color> is either **color** or **monochrome**. **"<fontname>"** is the name of a valid Windows font. **<fontsize>** is the size of the font in points.

graph-menu
printing
text-menu
wgnuplot.ini
windows3.0

graph-menu

The **gnuplot graph** window has the following options on a pop up menu accessed by pressing the right mouse button or selecting **Options** from the system menu:

Bring to Top when checked brings the graph window to the top after every plot.

Color when checked enables color linestyles. When unchecked it forces monochrome linestyles.

Copy to Clipboard copies a bitmap and a Metafile picture.

Background... sets the window background color.

Choose Font... selects the font used in the graphics window.

Line Styles... allows customization of the line colors and styles.

Print... prints the graphics windows using a Windows printer driver and allows selection of the printer and scaling of the output. The output produced by **Print** is not as good as that from gnuplot's own printer drivers.

Update wgnuplot.ini saves the current window locations, window sizes, text window font, text window font size, graph window font, graph window font size, background color and linestyles to the initialisation file **WGNUPLOT.INI**.

printing

In order of preference, graphs may be be printed in the following ways.

1. Use the gnuplot command set terminal to select a printer and set output to redirect output to a file.

2. Select the **Print...** command from the **gnuplot graph** window. An extra command **screendump** does this from the text window.

3. If **set output "PRN"** is used, output will go to a temporary file. When you exit from gnuplot or when you change the output with another set output command, a dialog box will appear for you to select a printer port.   If you choose OK, the output will be printed on the selected port, passing unmodified through the print manager.   It is possible to accidently (or deliberately) send printer output meant for one printer to an incompatible printer.

text-menu

The **gnuplot text** window has the following options on a pop up menu accessed by pressing the right mouse button or selecting **Options** from the system menu:

Copy to Clipboard copies marked text to the clipboard.

Paste copies text from the clipboard as if typed by the user.

Choose Font... selects the font used in the text window.

System Colors when selected makes the text window honor the System Colors set using the Control Panel.   When unselected, text is black or blue on a white background.

Update wgnuplot.ini saves the current text window location, text window size, text window font and text window font size to the initialisation file **WGNUPLOT.INI**.

MENU BAR

If the menu file **WGNUPLOT.MNU** is found in the same directory as WGNUPLOT.EXE, then the menu specified in **WGNUPLOT.MNU** will be loaded.

Menu commands are:
```
  [Menu]        Start a new menu with the name on the following line
  [EndMenu]     End current menu.
  --            Insert a horizontal menu separator
  |             Insert a vertical menu separator
  [Button]      Put next macro on a push button instead of a menu.
```

Macros take two lines with the macro name (menu entry) on the first line and the macro on the second line.   Leading spaces are ignored.

Macros commands are:
```
  [INPUT]       Input string with prompt terminated by [EOS] or {ENTER}
  [EOS]         End Of String terminator.  Generates no output.
  [OPEN]        Get name of file to open from list box, with title of
                list box terminated by [EOS], followed by default
                filename terminated by [EOS] or {ENTER}
                This uses COMMDLG.DLL from Windows 3.1.
  [SAVE]        Get name of file to save.  Similar to [OPEN]
```

Macros character substitutions are:
```
  {ENTER}       Carriage Return '\r'
  {TAB}         Tab     '\011'
  {ESC}         Escape  '\033'
  {^A}          '\001'
  ...
  {^_}          '\031'
```

Macros are limited to 256 characters after expansion.

wgnuplot.ini

Windows gnuplot will read some of its options from the **[WGNUPLOT]** section of **WGNUPLOT.INI** in the Windows directory. An example **WGNUPLOT.INI** file is shown below.

```
 [WGNUPLOT]
 TextOrigin=0 0
 TextSize=640 150
 TextFont=Terminal,9
 GraphOrigin=0 150
 GraphSize=640 330
 GraphFont=Arial,10
 GraphColor=1
 GraphToTop=1
 GraphBackground=255 255 255
 Border=0 0 0 0 0
 Axis=192 192 192 2 2
 Line1=0 0 255 0 0
 Line2=0 255 0 0 1
 Line3=255 0 0 0 2
 Line4=255 0 255 0 3
 Line5=0 0 128 0 4
```

The **GraphFont** entry specifies the font name and size in points. The 5 numbers given in the **Border**, **Axis** and **Line** entries are the **Red** intensity (0-255), **Green** intensity, **Blue** intensity, **Color Linestyle** and **Mono Linestyle**. **Linestyles** are 0=SOLID, 1=DASH, 2=DOT, 3=DASHDOT, 4=DASHDOTDOT. In the example **WGNUPLOT.INI** file above, Line 2 is a green solid line in color mode, or a dashed line in monochrome mode. The default line width is 1 pixel.   If **Linestyle** is negative it specifies the width of a SOLID line in pixels. Line1 and any linestyle used with the points style must be SOLID with unit width.

windows3.0

Windows 3.1 is preferred, but WGNUPLOT will run under Windows 3.0 with the following restrictions:

1. COMMDLG.DLL and SHELL.DLL (available with Windows 3.1 or Borland C++ 3.1) must be in the windows directory.

2. WGNUPLOT.HLP produced by Borland C++ 3.1 is in Windows 3.1 format. You need to use the WINHELP.EXE supplied with Borland C++ 3.1.

3. It won't run in real mode due to lack of memory.

4. Truetype fonts are not available in the graph window.

5. Drag-drop does not work.

tics

By default, tics are drawn inwards from the border on all four sides. The **set tics** command can be used to change the tics to be drawn outwards on the left and bottom borders only. This is useful when doing impulse plots.

Syntax:
```
        set tics {<direction>}
        show tics
```

where <direction> may be **in** or **out**. **set tics** defaults to **in**.

See also the set xtics, set ytics, and set ztics command for more control of tic marks. KUsing splot, in 3-d plots, one can adjust the relative height of the vertical (Z) axis using **set ticslevel**. The numeric argument provided specifies the location of the bottom of the scale. a zero will put it on the bottom grid and any positive number somewhere along the z axis.

Syntax:
```
        set ticslevel {<level>}
        show tics
```

where <level> is a non negative numeric argument. For example,

```
        set ticslevel 0.5
```

sets the tics level to the default value.

See also the set view.

time

The optional **set time** places the time and date of the plot either at the top or bottom of the left margin. The exact location is device dependent.

Syntax:
```
        set time {<xoff>}{,<yoff>}
        set notime
        show time
```

Specifying constants <xoff> or <yoff> as optional offsets for the time will move the time <xoff> or <yoff> character screen coordinates. For example,

```
        set time ,-3
```

will change only the y offset of the time, moving the title down by roughly the height of three characters.

title

The **set title** command produces a plot title that is centered at the top of the plot. Using the optional x,y screen offsets, the title can be placed anywhere on the plot. **set title** with no parameters clears the title.

Syntax:

```
        set title {"<title-text>"} {<xoff>}{,<yoff>}
        show title
```

Specifying constants <xoff> or <yoff> as optional offsets for the title will move the title <xoff> or <yoff> character screen coordinates. Note these are screen coordinates and not plot coordinates. For example,

```
        set title ,-1
```

will change only the y offset of the title, moving the title down by roughly the height of one character.

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

trange

The **set trange** command sets the parametric range used to compute x and y values when in parametric mode. If not in parametric mode (see set parametric) then this range is not used. This command does not affect x/y autoscaling or x/y ranges.

This range may also be specified on the plot command line when in parametric mode.

Syntax:
```
        set trange [{<tmin> : <tmax>}]
```

where <tmin> and <tmax> terms are constants or expressions.

Both the <tmin> and <tmax> terms are optional. Anything omitted will not be changed, so
```
        set trange [:10]
```
changes tmax to 10 without affecting tmin.   See also set urange and set parametric.

urange

The **set urange** and set vrange commands sets the parametric ranges used to compute x, y, and z values when in splot parametric mode. If not in parametric mode (see set parametric) then these ranges are not used. This command does not affect x/y autoscaling or x/y ranges.

This range may also be specified on the splot command line when in parametric mode.   See plot for more information

Syntax:

```
        set urange [{<umin> : <umax>}]
```

where <umin> and <umax> terms are constants or expressions.

Both the <umin> and <umax> terms are optional. Anything omitted will not be changed, so

```
        set urange [:10]
```

changes umax to 10 without affecting umin.   See also set trange.

variables

The **show variables** command lists all user-defined variables and their values.

Syntax:

```
        show variables
```

view

The **set view** command sets the view point for splots. This command controls the way the 3-d coordinates of the plot are mapped into the 2-d screen space. This command provides controls to both rotation and scaling of the plotted data but supports orthographic projections only.

Syntax:
```
        set view <rot_x> {,{<rot_z>}{,{<scale>}{,<scale_z>}}}
        show view
```

where <rot_x> and <rot_z> control the rotation angles (in degrees) along a virtual 3-d coordinate system aligned with the screen such that the screen horizontal axis is x, screen vertical axis is y, and the axis perpendicular to the screen is z. <rot_x> is bounded to the [0:180] range with a default of 60 degrees, while <rot_z> is bounded to the [0:360] range with a default of 30 degrees. <scale> controls the scaling of the entire splot, while <scale_z> scales the z axis only. Both scales default to 1.0.

Examples:
```
        set view 60, 30, 1, 1
        set view ,,0.5
```

The first sets all the four default values. The second changes only scale, to 0.5.

See also set ticslevel.

vrange

The **set vrange** command is similar to the set urange command. Please see set urange.

xlabel

The **set xlabel** command sets the x-axis label that is centered along the x axis. Using the optional x,y screen offsets, the label can be placed anywhere on the plot. **set xlabel** with no parameters clears the label.

Syntax:
```
        set xlabel {"<label>"} {<xoff>}{,<yoff>}
        show xlabel
```

Specifying constants <xoff> or <yoff> as optional offsets for the label will move the label <xoff> or <yoff> character screen coordinates. For example,

```
        set xlabel -1
```

will change only the x offset of the xlabel, moving the label roughly one character width to the left.

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

xrange

The **set xrange** command sets the horizontal range that will be displayed. This command turns x axis autoscaling off.

This range may also be specified on the plot command line.

Syntax:
```
        set xrange [{<xmin> : <xmax>}]
```

where <xmin> and <xmax> terms are constants or expressions.

Both the <xmin> and <xmax> terms are optional. Anything omitted will not be changed, so
```
        set xrange [:10]
```
changes xmax to 10 without affecting xmin.

xtics

Fine control of the x axis tic marks is possible with the **set xtics** command. The x-axis tic marks may be turned off with the **set noxtics** command. They may be turned on (the default state) with **set xtics**.

Syntax:
```
         set xtics { {<start>, <incr>{, <end>}} |
                     {({"<label>"} <pos> {, {"<label>"} <pos>}...)} }
         set noxtics
         show xtics
```

The <start>, <incr>, <end> form specifies that a series of tics will be plotted on the x axis between the x values <start> and <end> with an increment of <incr>. If <end> is not given it is assumed to be infinity. The increment may be negative. For example,
```
          set xtics 0,.5,10
```
makes tics at 0, 0.5, 1, 1.5, ..., 9.5, 10.

The ("<label>" <pos>, ...) form allows arbitrary tic positions or non-numeric tic labels. A set of tics are a set of positions, each with its own optional label. Note that the label is a string enclosed by quotes, and may be a constant string, such as "hello", or contain formatting information for the tic number (which is the same as the position), such as "%3f clients". See set format for more information about this case. The label may even be empty. Examples:
```
         set xtics ("low" 0, "medium" 50, "high" 100)
         set xtics (1,2,4,8,16,32,64,128,256,512,1024)
         set xtics ("bottom" 0, "" 10, "top" 20)
```

Tics will only be plotted when in range.

The set ytics and set noytics commands work identically. See also the set format command.

xdtics

The **set xdtics** commands converts the x axis tic marks to days of the week where 0=Sun and 6=Sat.   Overflows are converted modulo 7 to dates.

Examples:

```
        set xdtics
```

Sets x axis tics in days.

The set ydtics set zdtics and set noydtics set nozdtics commands work identically. See also the set format command.

xmtics

The **set xmtics** commands converts the x axis tic marks to months of the years where 1=Jan and 12=Dec.   Overflows are converted modulo 12 to months.

Examples:

```
        set xmtics
```

Sets x axis tics into months.

The set ymtics set zmtics and set noymtics set nozmtics commands work identically. See also the set format command.

xzeroaxis

set xzeroaxis draws the x-axis. By default, this option is on. **set noxzeroaxis** causes GNUPLOT to omit the x-axis.

Syntax:

```
        set xzeroaxis
        set noxzeroaxis
        show xzeroaxis
```

ylabel

The **set ylabel** command sets the y-axis label.   The position of this label depends on the terminal, and can be one of the following three positions (the position can be adjusted with optional parameters).

1. Horizontal text flushed left at the top left of the plot. Terminals that cannot rotate text will probably use this method.

2. Vertical text centered vertically at the left of the plot. Terminals that can rotate text will probably use this method.

3. Horizontal text centered vertically at the left of the plot. The EEPIC, LaTeX and TPIC drivers use this method. The user must insert line breaks using \\ to prevent the ylabel from overwriting the plot. To produce a vertical row of characters, add \\ between every printing character (but this is ugly).

Syntax:
```
        set ylabel {"<label>"} {<xoff>}{,<yoff>}
        show ylabel
```

With no parameters, the label is cleared. Specifying constants <xoff> or <yoff> as optional offsets for the label will move the label <xoff> or <yoff> character screen coordinates. For example,

```
        set ylabel -1
```

will change only the x offset of the ylabel, moving the label roughly one character width left of its default position. This is especially useful with the LaTeX driver.

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

yrange

The **set yrange** command sets the vertical range that will be displayed. This command turns y axis autoscaling off.

This range may also be specified on the plot command line.

Syntax:
```
        set yrange [{<ymin> : <ymax>}]
```

where <ymin> and <ymax> terms are constants or expressions.

Both the <ymin> and <ymax> terms are optional. Anything omitted will not be changed, so
```
        set yrange [:10]
```
changes ymax to 10 without affecting ymin.

ytics

The **set ytics** and **set noytics** commands are similar to the set xtics and set noxtics commands. Please see set xtics.

ydtics

The **set ydtics** and **set noydtics** commands are similar to the set xdtics and set noxdtics commands. Please see set xdtics.

ymtics

The **set ymtics** and **set noymtics** commands are similar to the set xmtics and set noxmtics commands. Please see set xmtics.

yzeroaxis

set yzeroaxis draws the y-axis. By default, this option is on. **set noyzeroaxis** causes GNUPLOT to omit the y-axis.

Syntax:
```
        set yzeroaxis
        set noyzeroaxis
        show yzeroaxis
```

zero

The **zero** value is the default threshold for values approaching 0.0. GNUPLOT will not plot a point if its imaginary part is greater in magnitude than the **zero** threshold. Axis ranges cannot be less than **zero**. The default **zero** value is 1e-8. This can be changed with the **set zero** command.

Syntax:
```
        set zero <expression>
        show zero
```

zeroaxis

set zeroaxis draws the x-axis and y-axis. By default, this option is on. **set nozeroaxis** causes GNUPLOT to omit the axes, and is equivalent to **set noxzeroaxis; set noyzeroaxis.**

Syntax:

```
        set zeroaxis
        set nozeroaxis
        show zeroaxis
```

See set xzeroaxis and set yzeroaxis.

zlabel

The **set zlabel** command sets the z-axis label that is centered along the z axis. Using the optional x,y screen offsets, the label can be placed anywhere on the plot. **set zlabel** with no parameters clears the label.

Syntax:

```
        set zlabel {"<label>"} {<xoff>}{,<yoff>}
        show zlabel
```

Specifying constants <xoff> or <yoff> as optional offsets for the label will move the label <xoff> or <yoff> character screen coordinates. For example,

```
        set zlabel ,1
```

will change only the y offset of the zlabel, moving the label roughly one character height up.

The zlabel will be drawn whenever surfaces or contours are plotted, in the space above the grid level.

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

zrange

The **set zrange** command sets the vertical range that will be displayed. This command turns z axis autoscaling off.   The zrange is used only by splot and is ignored by plot.

This range may also be specified on the splot command line.

Syntax:
```
        set zrange [{<zmin> : <zmax>}]
```

where <zmin> and <zmax> terms are constants or expressions.

Both the <zmin> and <zmax> terms are optional. Anything omitted will not be changed, so
```
        set zrange [2:]
```
changes zmin to 2 without affecting zmax.

ztics

The **set ztics** and **set noztics** commands are similar to the set xtics and set noxtics commands. Please see set xtics.

zdtics

The **set zdtics** and **set nozdtics** commands are similar to the set xdtics and set noxdtics commands. Please see set xdtics.

zmtics

The **set zmtics** and **set nozmtics** commands are similar to the set xmtics and set noxmtics commands. Please see set xmtics.

shell

The **shell** command spawns an interactive shell. To return to GNUPLOT, type **logout** if using VMS, exit or the END-OF-FILE character if using Unix, **endcli** if using AmigaDOS, or exit if using MS-DOS or OS/2.

A single shell command may be spawned by preceding it with the ! character ($ if using VMS) at the beginning of a command line. Control will return immediately to GNUPLOT after this command is executed. For example, in VMS, AmigaDOS, MS-DOS or OS/2,

```
      ! dir
```

prints a directory listing and then returns to GNUPLOT.

On an Atari, the **!** command first checks whether a shell is already loaded and uses it, if available. This is practical if GNUPLOT is run from **gulam**, for example.

splot

Three-dimensional surface and contour plotting is available in GNUPLOT with the splot command. See the plot command for features common to the plot command.

See also set contour, set cntrparam, and set surface.

Binary Data

Binary Data

Gnuplot will dynamically determine if a datafile is ASCII or binary.   ASCII data files are discussed in the plot section. For three dimensions, single precision floats are stored as follows:

```
      <ncols> <x0> <x1> <x2> ...
      <y0> <z0,0> <z0,1> <z0,2> ...
      <y1> <z1,0> <z1,1> <z1,2> ...
```

which is converted into triplet:

```
      <x0> <y0> <z0,0>
      <x0> <y1> <z0,1>
      <x0> <y2> <z0,2>


      <x1> <y0> <z1,0>
      <x1> <y1> <z1,1>
      <x1> <y2> <z1,2>
```

These triplets are then converted into gnuplot iso_curves and then uses gnuplot to do the rest of the plotting.

A collection of matrix and vector manipulation routines (in C) are provided in **gnubin.c**. The routine to write binary data is

```
 int fwrite_matrix(file,m,nrl,nrl,ncl,nch,row_title,column_title)
```

An example of using these routines is provided in the file **bf_test.c**. The corresponding demo file is **demo/binary.dem**.

start-up

When GNUPLOT is run, it looks for an initialization file to load. This file is called **.gnuplot** on Unix and AmigaDOS systems, and **GNUPLOT.INI** on other systems. If this file is not found in the current directory, the program will look for it in the home directory (under AmigaDOS, AtariTOS, MS-DOS and OS/2, the environment variable GNUPLOT should contain the name of this directory).   Note: if NOCWDRC is defined during the installation, GNUPLOT will not read from the current directory.

If this file is found, GNUPLOT executes the commands in this file. This is most useful for setting the terminal type and defining any functions or variables that are used often.

substitution

Command-line substitution is specified by a system command enclosed in backquotes. This command is spawned and the output it produces replaces the name of the command (and backquotes) on the command line.

Newlines in the output produced by the spawned command are replaced with blanks.

Command-line substitution can be used anywhere on the GNUPLOT command line.

Example:

This will run the program **leastsq** and replace **leastsq** (including backquotes) on the command line with its output:

```
      f(x) = `leastsq`
```

or, in VMS

```
      f(x) = `run leastsq`
```

user-defined

New user-defined variables and functions of one through five variables may be declared and used anywhere.

User-defined function syntax:

```
        <function-name> ( <dummy1> {,<dummy2> {, ...} } ) = <expression>
```

where <expression> is defined in terms of <dummy1> through <dummy5>.

User-defined variable syntax:

```
        <variable-name> = <constant-expression>
```

Examples:

```
        w = 2
        q = floor(tan(pi/2 - 0.1))
        f(x) = sin(w*x)
        sinc(x) = sin(pi*x)/(pi*x)
        delta(t) = (t == 0)
        ramp(t) = (t > 0) ? t : 0
        min(a,b) = (a < b) ? a : b
        comb(n,k) = n!/(k!*(n-k)!)
        len3d(x,y,z) = sqrt(x*x+y*y+z*z)
```

Note that the variable **pi** is already defined.

See show functions and show variables.

bugs

The bessel functions do not work for complex arguments.

The gamma function does not work for complex arguments.

There is a bug in the stdio library for old Sun operating systems (SunOS Sys4-3.2). The "%g" format for 'printf' sometimes incorrectly prints numbers (e.g., 200000.0 as "2"). Thus, tic mark labels may be incorrect on a Sun4 version of GNUPLOT. A work-around is to rescale the data or use the set format command to change the tic mark format to "%7.0f" or some other appropriate format. This appears to have been fixed in SunOS 4.0.

Another bug: On a Sun3 under SunOS 4.0, and on Sun4's under Sys4-3.2 and SunOS 4.0, the 'sscanf' routine incorrectly parses "00 12" with the format "%f %f" and reads 0 and 0 instead of 0 and 12. This affects data input. If the data file contains x coordinates that are zero but are specified like '00', '000', etc, then you will read the wrong y values. Check any data files or upgrade the SunOS. It appears to have been fixed in SunOS 4.1.1.

Microsoft C 5.1 has a nasty bug associated with the %g format for printf. When any of the formats "%.2g", "%.1g", "%.0g", "%.g" are used, printf will incorrectly print numbers in the range 1e-4 to 1e-1. Numbers that should be printed in the %e format are incorrectly printed in the %f format, with the wrong number of zeros after the decimal point.

To work around this problem, use the %e or %f formats explicitly.

GNUPLOT, when compiled with Microsoft C, did not work correctly on two VGA displays that were tested. The CGA, EGA and VGA drivers should probably be rewritten to use the Microsoft C graphics library. GNUPLOT compiled with Borland C++ uses the Turbo C graphics drivers and does work correctly with VGA displays.

VAX/VMS 4.7 C compiler release 2.4 also has a poorly implemented %g format for printf. The numbers are printed numerically correct, but may not be in the requested format. The K&R second edition says that for the %g format, %e is used if the exponent is less than -4 or greater than or equal to the precision. The VAX uses %e format if the exponent is less than -1. The VAX appears to take no notice of the precision when deciding whether to use %e or %f for numbers less than 1. To work around this problem, use the %e or %f formats explicitly. From the VAX C 2.4 release notes: e,E,f,F,g,G   Result will always contain a decimal   point. For g and G, trailing zeros will not be removed from the result.

VAX/VMS 5.2 C compiler release 3.0 has a slightly better implemented %g format than release 2.4, but not much. Trailing decimal points are now removed, but trailing zeros are still not removed from %g numbers in exponential format.

ULTRIX X11R3 has a bug that causes the X11 driver to display "every other" plot.   The bug seems to be fixed in DEC's release of X11R4 so newer releases of ULTRIX don't seem to have the problem.   Solutions for older sites include upgrading the X11 libraries (from DEC or direct from MIT) or defining ULTRIX_KLUDGE when compiling the x11.trm file.   Note that the kludge is not an ideal fix, however.

The constant HUGE was incorrectly defined in the NeXT OS 2.0 operating system.   HUGE should be set to 1e38 in plot.h. This error has been corrected in the 2.1 version of NeXT OS.

Some older models of HP plotters do not have a page eject command 'PG'. The current HPGL driver uses this command in HPGL_reset.   This may need to be removed for these plotters. The current PCL5 driver uses HPGL/2 for text as well as graphics.   This should be modified to use scalable PCL fonts.

On the Atari version, it is not possible to send output directly to the printer (using **/dev/lp** as output file), since CRs are added to LFs in binary output. As a workaround write the output to a file and copy it to the printer afterwards using a shell command.

Please report any bugs to bug-gnuplot@dartmouth.edu.