gnuplot Befehlsreferenz

Bazsó Ákos

Version 1.1
3. März 2010

Index

T

Vorbemerkungen

Diese Befehlsreferenz bezieht sich auf die aktuelle gnuplot Programmversion V. 4.2; ältere Versionen enthalten unter Umständen nicht alle aufgeführten Befehle, in neueren Versionen können neue Befehle hinzukommen und hier erklärte Befehle geändert werden.

gnuplot ist ein befehlsorientiertes interaktives Programm zur Darstellung von Daten und Funktionen[1]. Bei der Eingabe wird nach Groß- und Kleinschreibung unterschieden, d.h. `sin(x)` ist nicht gleich `Sin(x)`. Alle Befehlsnamen können abgekürzt werden (z.B. `splot` wird zu `sp`), solange die Abkürzung noch eindeutig ist. Mehrere Befehle dürfen hintereinander angegeben werden, solange sie durch ein Semikolon `";"` getrennt sind. (Ausnahmen bei bestimmten Befehlen werden dort erwähnt.) Zeichenketten (*strings*) werden in Anführungszeichen angegeben, entweder als `"Text"` oder `'Text'`.

Beim Aufruf von gnuplot können beliebig viele Dateinamen als Argumente angegeben werden, gnuplot wird dann versuchen diese Zeile für Zeile nach Befehlen zu durchsuchen und abzuarbeiten (batch mode). Nach dem letzten Befehl in der letzten Datei beendet sich das Programm von selbst. Falls keine Datei als Argument angegeben wird, startet ein interaktives Terminal und nimmt Befehle entgegen (interactive mode). Befehlseingaben dürfen sich auch über mehrere Zeilen erstrecken, solange das letzte Zeichen in einer Zeile ein Schrägstrich `"\"` ist.

Viele der internen Befehle haben mehrere Optionen, deren Reihenfolge normalerweise unwichtig ist. Sollten aber Fehlermeldungen darauf hinweisen, muss die exakte Reihenfolge der Befehle wie in der Hilfe erläutert eingehalten werden.

Diese Befehlsreferenz ist zum Teil nur eine subjektive Auswahl, sie beschreibt nicht alle Befehle und Optionen mit allen Details. Deshalb – und zur vollständigen Beschreibung aller Möglichkeiten – ist das in gnuplot eingebaute Hilfesystem unverzichtbar. Die Hilfe zu jedem Befehl steht über `help Befehl` zur Verfügung.

P.S.: Wer Fehler findet, darf sie behalten!

Referenzen

Die folgende Liste von Quellen wird ständig erweitert – falls dazu Zeit übrig bleibt ;-)

1. Thomas Williams, Colin Kelley, et al., gnuplot V. 4.2 Hilfesystem (`help copyright`)
2. gnuplot Homepage, http://www.gnuplot.info/
3. Demo scripts for gnuplot Version 4.2, http://gnuplot.sourceforge.net/demo/
4. The not so Frequently Asked Questions, http://t16web.lanl.gov/Kawano/gnuplot/index-e.html

[1] Quelle: gnuplot Hilfesystem – Thema: `help introduction`

Konventionen

In dieser gnuplot Referenz wird folgende Konvention zur Darstellung von Befehlen und Argumenten verwendet:

Stil für Eingaben Eine beliebige Eingabe wird so dargestellt: `plot sin(x)`

Stil für Befehle Grundsätzlich werden Befehlsoptionen durch zwei Arten von Klammern ([ ], { }) dargestellt:
`Befehl [notwendige Argumente] {optionale Argumente}`

Stil für Optionslisten Optionen werden mit den logischen Operatoren UND (&) bzw. ODER (|) verknüpft:
`Befehl [arg1 | arg2 | ...]` Auswahl von immer nur jeweils einem Argument – ENTWEDER arg1 ODER arg2 ODER ...
`Befehl [arg1 & arg2 & ...]` Auswahl von beliebig vielen Argumenten – arg1 UND arg2 UND ...

Stil für Befehlsbeschreibungen Befehle werden nach folgendem Schema beschrieben:

- Befehlsname
- Syntax und Optionen
- Beschreibung
- Beispiel(e)

Bemerkungen zu den verfügbaren Funktionen

Die von gnuplot bereitgestellten Funktionen[2] erwarten ganzzahlige ($x = 42$), reelle ($x = 3.14159$) oder komplexe Argumente ($x = \{Re, Im\}$).

Wichtig für alle Winkelfunktionen: Zur Auswahl des richtigen Winkel-Maßsystems die Einstellung von „angles“ = [degrees (Grad) | radians (Bogenmaß)] beachten (**siehe: set angles (S. 20)**).

Die Hilfe zu den folgenden in gnuplot verfügbaren Funktionen ist jeweils über den Befehl `help [Funktionsname]` erreichbar, z.B. `help sin`.

- `abs,arg,imag,invnorm,norm,real,sgn`
- `acos,acosh,asin,asinh,atan,atan2,atanh,cos,cosh,sin,sinh,tan,tanh`
- `besj0,besj1,besy0,besy1,erf,erfc,gamma,ibeta,inverf,igamma, lambertw,lgamma`
- `ceil,floor,int`
- `exp,log,log10,sqrt`
- `rand`

[2] Hilfethema: `help functions`. Unter Linux siehe auch `man math.h`.

Bemerkungen zu den verfügbaren Koordinatensystemen

Für Positionsangaben und die Wahl des Koordinatensystems[3] ist folgendes Schema zu beachten: `{system} [x], {system} [y], {system} [z]`
Möglichkeiten für `system`:

first Position in Einheiten der linken und unteren Achse (x-y-Achsen, Standard) **first**

second Position in Einheiten der rechten und oberen Achse (x2-y2-Achsen) **second**

graph Position innerhalb des Zeichenbereichs (zwischen x-x2-y-y2-Achsen), Wertebereich von $(0,0)$ = untere linke Ecke bis $(1,1)$ = obere rechte Ecke bzw. $(0,0,0)$ bis $(1,1,1)$ für splot **graph**

screen Position innerhalb des Bildbereichs (gesamter sichtbarer Bereich), Wertebereich von $(0,0)$ = untere linke Ecke bis $(1,1)$ = obere rechte Ecke **screen**

character wie die screen Koordinaten, nur in Einheiten der gewählten Schriftgröße (**siehe: set termoption (S. 46)**) **character**

Unterscheidung zwischen absoluten und relativen Koordinaten, mit `rto` werden Endkoord. relativ zu den Ausgangskoord. angegeben.
Beispiel:

1. `set arrow from 0,0 to 2,2` ... ein Pfeil vom Punkt $(x,y) = (0,0)$ (implizit: im System first) bis zum Punkt $(x,y) = (2,2)$ (ebenfalls System first)
2. `set arrow from screen 0,0 to graph 1,1` ... ein Mix von Koordinaten
3. `set arrow from screen 0.1,0.4 rto 0,0.4` ... ein Pfeil von $(x,y) = (0.1,0.4)$ bis $(x,y) = (0,0.4)$ relativ zu diesem Punkt, d.h. nach $(x,y) = (0.1+0,0.4+0.4) = (0.1,0.8)$

[3] Hilfethema: `help coordinates`

Befehle (alphabetisch)

Befehl: `cd` **cd**

`cd [pfad]`
wechselt das Arbeitsverzeichnis, der *Pfad* muss in Anführungszeichen angegeben werden.
Beispiel:

1. `cd "ziel"` bzw. `cd 'ziel'`, `cd "../data"`
2. `cd "d:\\data"` oder `cd 'd:\data'` ist richtig, aber `cd "d:\data"` ist falsch

Befehl: `call` **call**

`call [datei] {par1 & par2 & ...}`
wie `load` (**siehe: load (S. 10)**), mit dem Unterschied, dass innerhalb der *Datei* mit $0 für par1, $1 für par2, ... Variablen verwendet werden können (0 basierte Zählung!).
Beispiel:

1. `call ''myplot.plt'' 3.14 42` ... ruft „myplot.plt“ mit den Zahlen 3.14 und 42 als Parameter auf

Befehl: `clear` **clear**

`clear`
löscht das aktive Grafikfenster oder fügt eine leere Seite in die Ausgabedatei ein, wenn `set output` verwendet wird.

Befehl: `exit, quit` **exit, quit**

`exit | quit`
`gnuplot` beenden.

Befehl: `fit` **fit**

```
fit {xrange} {yrange} [funktion]
    [datei] {using auswahl}
    via [parameterdatei | [var1, var2, ...]]
```

Routine zur Anpassung einer *Funktion* an Daten aus *Datei* in den Variablen (x, y) oder (x, y, z) über Ermittlung der Parameter aus *Parameterdatei* bzw. nach den angegebenen Parametern (var1, var2, ...).

Eine Gewichtung der einzelnen Werte ist über eine zusätzliche Spalte in *Datei* möglich, sie sollte die Standardabweichung zum abhängigen Wert darstellen, also `using 1:2:3` für (x, y, σ_y) oder `using 1:2:3:4` für (x, y, z, σ_z).
Die Fehler der so bestimmten Parameter var_i können als eigene (temporäre) Variablen genutzt werden (**siehe: set fit errorvariables (S. 28)**); `save variables` (**siehe: save (S. 11)**) speichert die Variablen dauerhaft.
Beispiel:

1. `f(x)=a*x+b; fit f(x) ''messung.dat'' using 1:2 via a,b` ...
   bestimmt eine Ausgleichsgerade $f(x)$ mit Koeff. a, b nach Daten in Spalten 1 und 2 aus „messung.dat“
2. `fit f(x) ''messung.dat'' using 1:($2*0.5-3):3 via a,b` ...
   wie vorhin, diesmal mit Gewichtung nach dritter Spalte (immer gleiche Gewichtung mit `using 1:2:(1)`) und einer Transformation der Eingabedaten

Befehl: `help` **help**
`help [thema]`
der beste Freund des Benutzers ;-)
Beispiel:

1. `help` ohne Argument ruft die allgemeine Hilfe auf
2. `help plot` ruft Hilfe zum angegebenen Schlüsselwort *plot* auf

Befehl: `load` **load**
`load [datei]`
die Befehle in *Datei* Zeile für Zeile ausführen. Die Alternative direkt vom Terminal aus: `gnuplot ‘‘Datei‘‘`.
Beispiel:

1. `load ''myplot.plt''` bzw. `load 'myplot.plt'` ... startet eine neue gnuplot Sitzung und lädt die Datei „myplot.plt“, nach Ausführung aller Befehle darin beendet sich gnuplot automatisch.

Befehl: `pause` **pause**
`pause [Zeit] {''Text''}`
wartet für eine angegebene Anzahl von *Zeit* Sekunden auf Eingabe durch den Benutzer, wenn *Text* angegeben ist, wird er angezeigt. Wenn $Zeit = 0$, wird keine Pause gemacht, bei *Zeit* = -1 wird solange gewartet, bis Enter gedrückt wurde.
Beispiel:

1. `pause 3` ... wartet für 3 Sekunden
2. `pause 10 "Bitte warten - in 10 Sekunden geht es weiter!"` ... gnuplot wartet für 10 Sekunden und zeigt währenddessen den Text an

Befehl: `print` **print**

`print [expr1, expr2, ...]`
wie `pause 0 [expr]`, wobei der Ausdruck *expr* intern von gnuplot ausgewertet wird und das Ergebnis angezeigt wird. Beliebig viele Ausdrücke lassen sich hintereinander mit ',' getrennt aneinanderreihen
Beispiel:

1. `print 5/2` ... gibt den Wert „2" aus (ganzzahliger Wert der Division)
2. `print "Division 5/2 = ", 5/2` ... gibt den Text und das Ergebnis der Division aus
3. `print 5./2, ''\t*\t'', pi, ''\t=\t'', 5*pi/2` ... zeigt die von gnuplot ausgewerteten Ausdrücke an, "`\t`" erzeugt einen Tabulatorsprung

Befehl: `pwd` **pwd**

`pwd`
zeigt das aktuelle Arbeitsverzeichnis an.

Befehl: `reread` **reread**

`reread`
die letzte durch `load` geladene Datei von Beginn an neu einlesen, nützlich für Schleifen oder wenn die Datei extern verändert wurde.

Befehl: `reset` **reset**

`reset`
alle vorherigen `set` Befehle außer Kraft setzen, alles auf Standardwerte zurückstellen.

Befehl: `save` **save**

`save {functions, set, terminal, variables} [datei]`
sichert die aktuellen Einstellungen in *Datei*, mit `load` können sie später geladen werden. Ohne Angaben sichert `save` alle 4 Optionen, mit angegebener spezieller Option nur die jeweilige Einstellung.
Beispiel:

1. `save "work.plt"` ... sichert alle Umgebungseinstellungen

2. `save set ''mysettings.dat''` ... sichert nur die Einstellungen der „*set*“ Umgebung

3. `save term ''myterminal.dat''` ... sichert aktuelle Terminal Einstellungen

Befehl: `shell` **shell**

`shell`
öffnet eine Befehlszeile, beendet die Befehlszeile sofort nach Ausführung des Befehls, Alternative: `system “Befehl“` bzw. `! “Befehl“`

Befehl: `test` **test**

`test {terminal | palette}`
erstellt ein Testbild und zeigt die Möglichkeiten des eingestellten Terminals oder der Palette an.

Befehl: `plot` **plot**

```
plot {ranges} [funktion | datei {modifikator}]
     {axes [achsen]} {title "text" | notitle}
     {with [stil] {(linestyle | ls) &
                   (linetype  | lt) &
                   (linewidth | lw) &
                   (linecolor | lc) &
                   (pointtype | pt) &
                   (pointsize | ps) &
                   (fill      | fs) &
                   palette
                  }
```

erstellt eine 2D Grafik gemäß der angegebenen *Datei* oder *Funktion*; zulässige Funktionen sind interne Fkt. (**siehe: eingebaute Funktionen (S. 7)**), selbstdefinierte Fkt. z.B. mit `fit` (**siehe: fit (S. 9)**), und auch parametrisierte Funktionen $(t, f(t))$ (**siehe: set parametric (S. 39)**).
Die Angabe von *ranges* grenzt den Wertebereich auf die angegebenen Werte bzw. Intervalle ein (**siehe: set xrange (S. 50)**). Mögliche Wertebereiche sind: **plot ranges**

- **xrange, x2range:** Angaben für die untere (x) und obere (x2) Achse
- **yrange, y2range:** Angaben für die linke (y) und rechte (y2) Achse
- **trange:** Angabe für den Parameter t in parametric plots

- **rrange:** Angabe für die radiale Koord. r in polar plots

plot axes

Die Auswahl von *axes* legt fest, zu welcher Achse (Achsenskalierung) der aktuelle plot gehört (Standard: x1y1 Achse), d.h. unterschiedliche Achsenskalierungen in einem plot sind für verschiedene Achsen (siehe Beispiele) möglich.
Falls die Daten aus einer *Datei* gelesen werden, gibt es verschiedene *Modifikatoren*:

- *binary* ... liest *Datei* als Binärdatei ein; zwei Typen sind möglich: *binary matrix* (`binary`) ohne zusätzliche Angaben bei festem Dateiaufbau; sowie *binary general* (`binary <format>`) mit verschiedenen Arten von Dateiformaten[4] (**siehe: set datafile binary (S. 27)**).

- *matrix* ... liest *Datei* als eine Matrix ein, d.h. als Felder von Zahlen $z_{i,j}$, wobei die x-/y-Koordinaten implizit über die Zeilen-/Spaltennummer (beginnend mit Index 0) bestimmt werden:

$$\begin{array}{cccc} z_{0,0} & z_{0,1} & z_{0,2} & \cdots \\ z_{1,0} & z_{1,1} & z_{1,2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

  Beispiel:

  1. `plot ''mat.dat'' matrix using 1:3` ... die Zeilen der Matrix ausgeben
  2. `plot ''mat.dat'' matrix using 2:3` ... die Spalten der Matrix ausgeben
  3. `splot ''mat.dat'' matrix` ... die Matrix als Fläche in 3D ausgeben
  4. `splot ''mat.dat'' matrix using (1+$1):(1+$2*10):3` ...
     Matrix ausgeben und x- bzw. y-Achse skalieren

- *index* ... durch doppelte Leerzeilen getrennte Datengruppen bilden ein „Datenset“. `index i:j:k` wählt aus einer in Datensets unterteilten *Datei* sets aus.

  Beispiel:

  1. `plot ''datei.dat'' index 0` ... wählt nur das erste Datenset aus (Indexzählung beginnt bei 0!)
  2. `plot ''datei.dat'' index 4:7` ... wählt Datensets 5 bis 8 aus
  3. `plot ''datei.dat'' index 1:9:2` ... wählt Datensets von 1 bis 9 in Schritten von 2 aus, d.h. {1,3,5,7,9}
  4. ```
     set table ''data.dat''; plot x, x**2, x**3; unset table
     plot ''data.dat'' index 0, x    # index 0 = Graph f(x) = x
     plot ''data.dat'' index 1, x**2 # index 1 = Graph f(x) = x^2
     plot ''data.dat'' index 2, x**3 # index 2 = Graph f(x) = x^3
     ```

     eine Datei mit 3 Datensets für die Funktionen x, x^2, x^3 anlegen, danach die Datei einlesen und den gewählten Datenset (index) jeweils mit der entsprechenden Funktion vergleichen

[4] Hilfethemen: `help binary`; `help binary general`; `help binary examples`
     ```

- *every* ... zum periodischen Auswählen von Datenpunkten und -blöcken nach Format `every i:j:k:l:m:n`. Ein Datenpunkt ist eine einzelne Zeile in einer Datei, ein Datenblock sind mehrere durch Leerzeilen abgegrenzte Zeilen. Einzelne Zeilen werden von der k-ten Zeile bis zur m-ten Zeile in Abständen von i Zeilen ausgewählt, analog vom l-ten bis zum n-ten Block in Abständen von j Blöcken. Nicht angegebene Start-Werte (k, l) werden als 0, Schrittweiten (i, j) als 1 angenommen. **plot every**

  Beispiel:

  1. `every 2` ... nur jede zweite Zeile berücksichtigen
  2. `every :2` ... nur jeden zweiten Datenblock berücksichtigen
  3. `every ::3` ... jeweils erst ab der dritten Zeile jedes Datenblocks lesen
  4. `every ::20::75` ... nur Daten zwischen 20. und 75. Zeile berücksichtigen
  5. `every :::3::3` ... nur den 4-ten Block auswählen (Zählung ab 0!)
  6. `every :::::9` ... alle Blöcke von 0 bis 10
  7. `every 50::1::100` ... Voraussetzung: Datei hat 3 Spalten (x, y, z), jeweils nach 100 Zeilen mit gleichem x-Wert und versch. y-Werten folgt ein neuer x-Wert und y-Werte wiederholen sich (x-y-Gitter); alle Blöcke lesen, aber die erste Zeile überspringen und nur jede 50-ste Zeile lesen, d.h. nur jeweils die 2. und 51. Zeile lesen
  8.
```
plot 'ele.dat' using 1:2 every 3::0 title "Planet 1", \
             '' using 1:2 every 3::1 title "Planet 2", \
             '' using 1:2 every 3::2 title "Planet 3"
```

     für eine Datei mit 3 Planeten, bei der $(0, 1, 2)$-ten Zeile beginnen und jeweils 3 Zeilen überspringen

- *using* ... in *Datei* können verschiedene Spalten ausgewählt werden, dabei gilt das Format: `using x:y:z:...`. Falls die Spalten anders angeordnet sind, kann über $1, $2, usw. auf die passende Spalte zugegriffen werden, Beispiel: y und x Spalte vertauschen mit `using 2:1`. Die Spalte 0 steht für eine fortlaufende Nummer ($\propto$ Zeilennummer); eine leere using Angabe steht für den Standardwert, z.B. bei 4 Spalten ist `using :::4` identisch zu `using 1:2:3:4`. Mit using ist es auch möglich, mit Spaltenwerten zu „rechnen“, d.h. neue Werte basierend auf den eingelesenen zu erzeugen (siehe Beispiele). **plot using**

- *smooth* glättet eingelesene Daten nach verschiedenen Interpolationsmethoden.
  `smooth {unique | frequency | csplines | acsplines | bezier | sbezier}`

  – unique: mehrere Punkte mit gleichen x-Werten werden zu einem einzigen zusammengefasst mit den gemittelten y-Werten

  – frequency: mehrere Punkte mit gleichen x-Werten werden zu einem einzigen zusammengefasst mit den aufsummierten y-Werten

– csplines: verwendet cubic splines

– acsplines: verwendet cubic splines mit Gewichtung der Koeffizienten nach dritter Datenspalte aus gelesener Datei

Beispiel:

1. `plot ''datei.dat'' using 1:2:3 smooth acsplines` ...
   Gewichtung nach 3. Spalte aus *Datei*
2. `plot ''datei.dat'' using 1:2:(1.0) smooth acsplines` ...
   stets gleiche Gewichtung „1.0“ verwenden

– bezier: Daten mit einer Bezierkurve approximieren

– sbezier: wie bezier nach Anwendung von unique

Möglichkeiten für Stile (*with [Stil]*) bei `plot`: **plot Stile**

- *boxes* ... für Graphik mit Rechtecken, für Rechteckbreite **siehe: set boxwidth (S. 23)**
- *boxerrorbars, boxxyerrorbars* ... Rechteckstile wie mit *boxes* mit zusätzlichen Fehlerbalken für y-Variable bei *boxerrorbars* bzw. Fehlerbalken für x- und y-Variablen bei *boxxyerrorbars*. Achtung: zusätzliche Spalte für Fehler Δy ($y \pm \Delta y$), od. 2 zusätzliche Spalten für $\Delta y_-, \Delta y_+$ ($y^{+\Delta y_+}_{-\Delta y_-}$) bei *boxerrorbars*, bis zu 4 zusätzlichen Spalten als $(x, y, \Delta x, \Delta y)$ bzw. $(x, y, \Delta x_-, \Delta x_+, \Delta y_-, \Delta y_+)$ bei *boxxyerrorbars*.
- *candlesticks* ... für Statistikdaten mit Quartilen; 5 Spalten in der Eingabedatei notwendig, siehe: `load 'candlestick.dem'`
- *dots* ... jedes Punktepaar (x, y) wird durch einen Punkt dargestellt
- *errorbars, xerrorbars, yerrorbars, xyerrorbars, errorlines, xerrorlines, yerrorlines, xyerrorlines* ... Stile mit versch. Fehlerbalken (bei *errorbars*) für Punkte, bzw. mit verbundenen Punkten und Fehlerbalken (bei *errorlines*). Achtung: zusätzliche Spalten (1 bis 4 möglich) in Datei für $(x, y, \Delta x, \Delta y)$ oder $(x, y, \Delta x_-, \Delta x_+, \Delta y_-, \Delta y_+)$ notwendig.

  Beispiel:

  1. `plot ''daten.dat'' using 1:2:3 with xerrorbars` ...
     Daten mit x-Fehlerbalken $(x, y, \Delta x)$ zeichnen
  2. `plot ''daten.dat'' using 1:2:4 with yerrorbars` ...
     Daten mit y-Fehlerbalken $(x, y, \Delta y)$ zeichnen
  3. `plot ''daten.dat'' using 1:2:3:4 with xyerrorlines` ...
     Daten mit x- und y-Fehlerbalken $(x, y, \Delta x, \Delta y)$ zeichnen und Punkte durch Linien verbinden

- *filledcurves* ... für gefüllte Flächen in unterschiedlichen Formen, mögliche Optionen für Füllung sind:

- closed: Kurve als Polygon mit Füllung zwischen den Anfangs-/Endpunkten
- {above | below} {x1, x2, y1, y2} = [wert]: füllt die Kurve über bzw. unter einem bestimmten *Wert* der angegebenen Achse auf
- xy = [wert]: Füllung zwischen Anfangs-/Endpunkt der Kurve und dem gegebenen Punkt in x1,y1 Einheiten
- Füllung zwischen 2 Kurven aus Eingabedatei mit 3 Spalten (x, y_1, y_2), die Fläche zwischen den y-Werten wird gefüllt

  Beispiel:

  1. `plot sin(x) with filledcurves closed` ... Sinus-Fkt.
  2. `plot sin(x) with filledcurves above x1 = 0` ... teilweise gefüllte Sinus-Fkt.
  3. `plot sin(x) with filledcurves below y1 = -0.5` ... Füllung der Funktion zwischen -0.5 und -1 in y
  4. `plot sin(x) with filledcurves xy = 0,1` ... Extrapunkt $(x, y) = (0, 1)$ eingefügt
  5. `plot ''daten.dat'' using 1:2:3 with filledcurves` ... Achtung: unbedingt 3 Spalten einlesen mit `using`

Für weitere Optionen zur Füllung **siehe: set style fill (S. 42)**.

- *financebars* ... für Finanzdaten aus einer Datei mit 5 Spalten; für Beispiele siehe: `load 'finance.dem'`
- *histograms* ... für Histogramme von Daten aus mehreren Spalten $(x, y_1, y_2, y_3, \dots)$, für verschiedene Arten von Histogrammen **siehe: set style histogram (S. 42)** und `load 'histograms.dem'`
- *image, rgbimage* ... zur Ausgabe von Daten als Pixel-Bilder. Einlesen von (x, y, g) für plot oder (x, y, z, g) für splot mit Stil *image*, wobei „g“ einen Grauwert der aktuellen Palette darstellt. Mit Stil *rgbimage* werden Daten als (x, y, r, g, b) für plot oder (x, y, z, r, g, b) für splot eingelesen, wobei „r, g, b“ den Rot-, Grün- und Blau-Werten der jeweiligen (x, y) Koordinate entsprechen – siehe: `load 'image.dem'`.
- *impulses* ... stellt eine vertikale Linie von der x-Achse $(y = 0)$ bis zum Punkt (x, y) dar
- *labels* ... wenn bei Dateien mit mind. 3 Spalten die 3. Spalte Strings enthält, werden diese bei dem zugehörigen Punkt (x, y) automatisch als Textmarke (**siehe: set label (S. 32)**) gesetzt.
- *lines* ... aufeinander folgende Punkte durch Linien verbinden (**siehe: set style lines (S. 42)**)
- *linespoints* ... Koordinaten durch ein Symbol (z.B. Punkt) kennzeichnen und aufeinander folgende Punkte mit Linien verbinden

- *points* ... Koordinaten werden durch verschiedene Symbole gekennzeichnet abhängig von `pointtype` (Abkürzung `pt`). Für weitere Optionen **siehe: set pointsize (S. 40)**

  Beispiel:

  1. `plot sin(x) with points pt 5 ps 3` ... Symbolart und -größe einstellen

- *steps, fsteps, histeps* ... Darstellung als Stufenfunktionen, *steps* verbindet aufeinander folgende Punkte zuerst entlang der x-Richtung und dann y-Richtung, *fsteps* verbindet Punkte umgekehrt zu steps (y zuerst, dann x), *histeps* verbindet nach arithmetischem Mittelwert aus altem/neuem x-/y-Wert.

  Beispiel:

  1. Das Beispiel zeigt die Unterschiede der drei *steps* Stile. Im Intervall von 0 bis 10 (set xrange) wird die Funktion $f(x) = x$ in Schritten von 1 Einheit (set samples) dargestellt.

     ```
     set samples 11; set xrange [0:10]; set xtics 1;
     plot x with lines   title "f(x) = x", \
          x with steps   title "steps", \
          x with fsteps  title "fsteps", \
          x with histeps title "histeps"
     ```

- *vectors* ... Daten aus Datei als Vektorpfeile zeichnen. Für plot sind 4 Spalten (x, y, a, b), für splot sind dagegen 6 Spalten (x, y, z, a, b, c) erforderlich. Die Angabe (x, y, z, a, b, c) stellt den Vektor vom Ausgangspkt. (x, y, z) zum Endpkt. $(x+a, y+b, z+c)$ dar (für splot). Für weitere Formatierungen **siehe: set style arrow (S. 42)**.

Beispiel:

1. `plot sin(x)`

2. `plot f(x) = a*sin(b*x), a = 1.414, b = 0.707, f(x)`

3. `plot [0:2*pi] f(x)`

4. `plot [z=-pi:pi] [-1:1] sin(z)`

5. ```
   plot "daten.dat" using 1:($2 + 2) title "Funktion, verschoben",\
        "daten.dat" using 1:($3 - $2) notitle
   ```

6. ```
   set xrange [0:2*pi]; set x2range [-pi:pi]; set x2tics;
   plot sin(x) axes x1y1 title "Sinus x1y1",\
        sin(x) axes x2y1 title "Sinus x2y1"
   ```

zuerst die Wertebereiche für die x1 (untere) und x2 (obere) Achse festlegen, dann die Skalenstriche für x2 Achse einschalten, und schließlich zweimal die Sinus-Fkt. zeichnen und der jeweiligen Achse zuordnen.

Befehl: `splot` **splot**

```
splot {ranges} [funktion | datei {modifikator}]
      {title "text" | notitle}
      {with [stil]}
```

erstellt eine (2D Projektion einer) 3D Grafik der angegebenen *Funktion* oder *Datei*; zulässige Funktionen sind interne Fkt. (**siehe: eingebaute Funktionen (S. 7)**), selbstdefinierte Fkt. z.B. mit `fit`, und auch parametrisierte Funktionen $(u, v, f(u, v))$ (**siehe: set parametric (S. 39)**). Funktionen werden diskretisiert über einem Gitter aufgetragen, für Einstellungen zu diesem Gitter **siehe: set isosamples (S. 31)**. Die Sichtbarkeit von Flächen wird üblicherweise nicht berücksichtigt, d.h. es sind stets alle Punkte sichtbar, über die Option hidden3d (**siehe: set hidden3d (S. 30)**) lässt sich dies ändern.
Die Angabe von *ranges* grenzt Wertebereiche auf die angegebenen Werte bzw. Intervalle ein (**siehe: set xrange (S. 50)**). Mögliche Wertebereiche sind: **splot ranges**

xrange, x2range: Angaben für die untere (x) und obere (x2) Achse

yrange, y2range: Angaben für die linke (y) und rechte (y2) Achse

zrange: Angabe für die z-Achse, KEINE z2 Achse

urange, vrange: Angaben für die Parameter (u, v) in parametric plots

rrange: Angabe für die radiale Koord. r in polar plots

cbrange: Angabe für die colorbox

Falls die Daten aus einer *Datei* gelesen werden, gibt es verschiedene *Modifikatoren*:

- *binary* ... liest *Datei* als Binärdatei ein, wie bei `plot`
- *matrix* ... liest *Datei* als Matrix ein, d.h. nur z-Werte, x-/y-Werte ergeben sich implizit aus dem Zeilen-/Spaltenindex, wie bei `plot`
- *index* ... wählt in *Datei* bestimmte Datensets aus, wie bei `plot`
- *every* ... zum periodischen Auswählen von Datenpunkten und -blöcken, wie bei `plot`

- *using* ... in *Datei* können verschiedene Spalten ausgewählt werden, normalerweise sind 3 Spalten für splot notwendig: `using x:y:z`, zusätzliche Spalten bei manchen *Stilen* erforderlich.

Möglichkeiten für Stile (*with [Stil]*) bei `splot`: **splot Stile**

- *dots* ... jedes Wertetripel (x, y, z) wird durch einen Punkt dargestellt.
- *impulses* ... stellt eine vertikale Linie vom minimalen z-Wert (**siehe: set ticslevel (S. 46)** zur Anpassung) bis zum jeweiligen z-Wert dar.
- *lines* ... aufeinander folgende Punkte durch Linien verbinden (**siehe: set style line (S. 42)**).
- *linespoints* ... Koordinaten durch ein Symbol (z.B. Punkt) kennzeichnen und aufeinander folgende Punkte durch Linien verbinden.
- *points* ... Koordinaten werden durch versch. Symbole gekennzeichnet
- *pm3d* ... Daten durch eine Palette filtern (**siehe: set palette (S. 37)**) und als Farb- oder Graustufenebene bzw. -oberfläche darstellen (**siehe: set pm3d (S. 39)**)
- *vectors* ... Daten als Vektorpfeile darstellen, wie bei `plot`

Für Einstellungen zum Betrachtungswinkel **siehe: set view (S. 48)**; zur Einstellung der z-Höhe der xy-Ebene **siehe: set ticslevel (S. 46)**; zur Verbindung der Datenpunkte mittels Gitternetz **siehe: set dgrid3d (S. 28)**; zum Einzeichnen von Flächen **siehe: set surface (S. 45)**; zur Erstellung von Konturlinien **siehe: set contour (S. 26)**.
Beispiel:

1. `splot sin(x)*cos(y)` ... alle Punkte zeichnen
2. `set hidden3d; splot sin(x)*cos(y)` ... Sichtbarkeit beachten, vgl. zu vorher
3. `set isosamples 50; splot sin(x)*cos(y)` ... feineres Gitter einstellen, vgl. zu vorher
4. `set contour base; splot sin(x)*cos(y)` ... Konturen einzeichnen
5. `splot sin(x)*cos(y) with pm3d at surface` ... die z-Werte mittels einer Farbfunktion einfärben und auf die Oberfläche projizieren
6. `splot [0:1][][*:10] "daten.dat" using 2:1:($3*10)` ...
   Datei „daten.dat“ einlesen; gelesene x- und y-Spalte vertauschen (`using 2:1`); z-Koordinaten mit Faktor 10 skalieren; x-Wertebereich auf Intervall 0 bis 1 skalieren, y-Wertebereich von `gnuplot` automatisch festlegen lassen (bzw. vorherige Einstellung unverändert übernehmen); z-Wertebereich nach oben mit 10 begrenzen, minimalen Wert dynamisch ermitteln

Befehl: `replot` **replot**

`replot`
die letzte Zeichnung (mit `plot` oder `splot`) wiederholen. Wenn in der Zwischenzeit Parameter verändert wurden wird das berücksichtigt.

Befehle (set, show, unset)

Auf alle Optionen können die folgenden drei Befehle angewendet werden:

- `set [Option] {Wert1, Wert2, ...}` ... setzt die *Option* auf *Wert1*, *Wert2*, usw.
- `show [Option]` ... zeigt die aktuellen Einstellungen und Werte für *Option* an; `show all` zeigt ALLE aktuell gültigen Optionen an
- `unset [Option]` ... stellt die Standardeinstellung für *Option* wieder her

Befehl: `set angles` **set angles**

`set angles [degrees | radians]`
wählt den Wertebereich für trigonometr. Fkt., bei *degrees* (Grad) von $[0:360]$ bzw. bei *radians* (Bogenmaß) von $[0:2\pi]$, beeinflusst einige Funktionswerte (Winkelfkt.) und die Winkeldarstellung in Plots (parametric plots).

Befehl: `set arrow` **set arrow**

```
set arrow {Nummer} {from pos1 [to | rto] pos2}
          {head | heads | nohead | backhead}
          {size len, ang}
          {empty | filled | nofilled}
          {front | back}
          {arrowstyle &
           (linestyle | ls) &
           (linetype  | lt) &
           (linewidth | lw)
          }
```

erzeugt einen Pfeil von *Pos1* bis *Pos2*; wenn angegeben wird dieser Pfeil später mit Bezeichner *Nummer* angesprochen (nur Zahlen als Bezeichner möglich), ohne explizite Angabe von *Nummer* wird intern eine fortlaufende Nummer zugeordnet, sodass einmal definierte arrows nicht mit nächster Definition überschrieben werden.
Pfeile können mit Spitzen an beiden Enden = *heads*, mit nur einer Spitze = *head*, ohne Spitzen = *noheads* (eine einfache Gerade), oder mit umgekehrten Spitzen = *backheads* versehen sein.

Die Größe der Spitze wird durch *size* bestimmt, wobei *len* die Länge der Dreiecksseiten und *ang* den Winkel zur Pfeilrichtung vorgibt.
Die Spitzen werden als gefülltes Dreieck dargestellt mit *filled* bzw. als leeres Dreieck mit *empty* oder mit *nofilled* nur als einfache Spitze; für weitere Formatierungen: **siehe: set style arrow (S. 42)**, **siehe: set style line (S. 42)**.
Beispiel:

1. `set arrow to 1,2` ... Pfeil von $(0,0)$ (implizit angenommen) nach $(1,2)$
2. `set arrow 3 from 3, graph 0 to 3, graph 1 nohead` ... vertikale Linie in der Grafik (system first) bei x = 3 von unterem Rand (y = 0) zum oberen Rand (y = 1)
3. `set arrow from 10,5 rto 2,2` ... Pfeil von $(x,y) = (10,5)$ bis $(10+2, 5+2) = (12,7)$ in System „first“ wenn sonst nichts anderes angegeben
4. `set arrow from graph 0,0 to graph 1,1` ... Pfeil von linker unterer Ecke bis rechte obere Ecke
5. Eine Gruppe von Pfeilen zur Verdeutlichung der Unterschiede bei den Spitzen (jede Zeile ein separater Befehl):

```
set arrow 1 from 0,0 to -1,0 heads filled   # gefüllte Spitzen
set arrow 2 from 0,0 to  0,1 heads nofilled # einfache Spitzen
set arrow 3 from 0,0 to -1,0 head  empty    # leere Spitze
plot [-2:2] x
```

Befehl: `set autoscale`

set autoscale

```
set autoscale [{x  | xmin  | xmax}  |
               {x2 | x2min | x2max} |
               {y  | ymin  | ymax}  |
               {y2 | y2min | y2max} |
               {xy | xymin | xymax} |
               {z  | zmin  | zmax}  |
               {cb | cbmin | cbmax} |
               fix | keepfix
              ]
```

erlaubt die Autoskalierung für die angegebene Achse. Mit `unset autoscale {Achse}` wird die Autoskalierung für die angegebene Achse ausgeschaltet, mit angehängtem *min*

bzw. *max* wird nur der entsprechende Minimal- bzw. Maximalwert dieser Achse automatisch skaliert.
Beispiel:

1. `set autoscale x` ... skaliert x-Achse automatisch
2. `set autoscale xmax` ... skaliert Maximalwert der x-Achse automatisch, Einstellungen für xmin werden dadurch nicht verändert
3. `set autoscale` ... ermöglicht Autoskalierung für alle Achsen (Standard)

Befehl: `set bars` **set bars**
`set bars [small | large | fullwidth | size]`
zur Einstellung der Balkenbreite bei den plot-Stilen {errorbars, candlesticks, financebars, histograms}, manuelle Werte für *size* zwischen *small* = 0.0 und *large* = 1.0. *fullwidth* setzt die Breite der Fehlerbalken auf die Balkenbreite.

Befehl: `set bmargin` **set bmargin**
`set bmargin` ... **siehe: set margin (S. 33)**
kontrolliert den unteren Bildrandabstand

Befehl: `set border` **set border**

```
set border [Zahl]
           {front | back}
           {(linestyle | ls) &
            (linetype  | lt) &
            (linewidth | lw)
           }
```

legt fest, welche und wie die Grafikbegrenzungen gezeichnet werden; mit *front* als oberste Ebene, mit *back* als unterste Ebene (alle übrigen Grafikobjekte können sie überlagern). Für Zuordnungen der *Zahl* als Bitkombination zu den Achsen siehe Hilfethema: `help border`.
Beispiel:

1. `set border 31; plot sin(x)` ... zeichnet die rechte, linke, obere, untere Achse in plot (und die z-Achse in splot)
2. `set tics nomirror; set border 3; rep` ... nur x1 und y1 Achse zeichnen, vgl. zu vorher

3. `set border 4095` ... alle Begrenzungslinien in splot anzeigen

Befehl: `set boxwidth` **set boxwidth**
`set boxwidth [Breite] {absolute | relative}`
zur Einstellung der Rechteckbreite bei den plot-Stilen {boxes, boxerrorbars, candlesticks, histograms}.
Die *Breite* wird von gnuplot standardmäßig so berechnet, dass Boxen einander berühren (boxwidth = 1); mit *relative* und *Breite* < 1 bleibt Platz zw. Boxen, bei *Breite* > 1 überlappen die Boxen tlw.; mit *absolute* haben die Boxen die angegebene *Breite* von x-Einheiten.
Beispiel:

1. `plot sin(x) with boxes` ... Sinus-Fkt. mit boxes-Stil
2. `set boxwidth 0.5 relative; rep` ... Boxen nur halb so breit, d.h. berühren sich nicht, vgl. zu vorher
3. `set boxwidth 2.0 absolute; rep` ... Boxen sind 2 Einheiten der x-Achse breit, vgl. zu vorher

Befehl: `set cbdata` **set cbdata**
`set cbdata` ... **siehe: set xdata (S. 49)**
für Datenformat der colorbox (cb) Achse

Befehl: `set cbdtics` **set cbdtics**
`set cbdtics` ... **siehe: set xdtics (S. 49)**
für Tagesformat der colorbox (cb) Achse

Befehl: `set cblabel` **set cblabel**
`set cblabel` ... **siehe: set xlabel (S. 50)**
für Achsenbeschriftung der colorbox (cb) Achse

Befehl: `set cbmtics` **set cbmtics**
`set cbmtics` ... **siehe: set xmtics (S. 50)**
für Monatsformat der colorbox (cb) Achse

Befehl: `set cbrange` **set cbrange**
`set cbrange` ... **siehe: set xrange (S. 50)**
für Wertebereich der colorbox (cb) Achse

Befehl: `set cbtics` **set cbtics**
`set cbtics` ... **siehe: set xtics (S. 51)**
für Hauptskala (major tics) der colorbox (cb) Achse

Befehl: `set clabel` **set clabel**
`set clabel ''Format''`
verwendet das angegebene *Format* für die Zahlenwerte von contour plot Niveaulinien; für mögliche Formate **siehe: set format (S. 29)**
Beispiel:

1. `set clabel ''%.2e''` ... gibt Zahlen in Exponentialdarst. als $1.00e{+}01$, $1.50e{+}01$, $2.00e+01$ mit 2 Nachkommastellen aus
2. `set clabel ''%8.3g''` ... gibt Zahlen in Fließkommadarst. mit 3 Nachkommastellen aus (bis zu 8 Ziffern Gesamtlänge)

Befehl: `set clip` **set clip**
`set clip [points | one | two]`
kontrolliert die Art, wie gnuplot Datenpunkte, die außerhalb des sichtbaren (bzw. eingestellten) Bereichs liegen, behandelt.

- *points* ... setzt bei plots mit Stilen {points, lines, linespoints} das Zeichnen von Datenpunkten nahe den Begrenzungslinien außer Kraft
- *one* ... Verbindungen zwischen innen-/außenliegenden Punkten werden durch Liniensegment dargestellt
- *two* ... Verbindungen zwischen 2 außenliegenden Punkten, die durch Bildfeld gehen, werden dargestellt

Beispiel:

1. `unset clip; set pointsize 2; p [-3:3] sin(x) w lp pt 6` ...
   alle clipping Optionen abschalten, Punktgröße erhöhen (wg. Effekt), Randpunkte bei ± 3 werden eingezeichnet
2. `set clip points; rep` ... Datenpunkte zu nahe an Begrenzungen werden ignoriert, vgl. zu vorher

Befehl: `set cntrparam` **set cntrparam**

```
set cntrparam [ linear | bspline | cubicspline |
                points "pts" | order "ord" |
                levels { auto {"nbr"} | discrete [z1, z2, ...] |
                         incremental start, incr, end }
              ]
```

stellt in contour plots (**siehe: set contour (S. 26)**) fest, welche z-Werte als Konturlinien dargestellt werden, und bestimmt die Art wie gleiche z-Werte (isolevel) verbunden werden.

- *linear* ... gleiche z-Werte durch lineare Polygonzüge (Geradeb) verbunden
- *bspline* ... eine glatte Kurve der Ordnung *ord* verbindet näherungsweise Punkte mit gleichem z-Wert
- *cubicspline* ... stückweise lineare Verbindungen zw. echten und interpolierten gleichen z-Werten
- *points* ... Anzahl der Stützwerte *pts* für stückweise lineare Verbindungen zwischen den z-Werten
- *order* ... Ordnung der bspline Kurve, wenn keine automatische Wahl (*bspline*) gewünscht ist. Werte für *ord* zwischen 2 (linear) und 10.
- *levels [Option]* wählt die Anzahl der Konturlinien nach der *Option*:
  - auto: automatisch (Standard), *nbr* wählt optional die Anzahl der Konturen, die tatsächliche Anzahl kann aber abweichen (laut gnuplot aus zeichentaktischen Gründen—der User soll nicht immer bekommen was er will ;)
  - discrete: Konturlinien nur für die angegebenen diskreten z-Werte z1, z2, ...
  - increment: Konturlinien ab z-Wert *start* in Schritten von *incr* bis z-Wert *end*

Standardwerte: `cntrparam linear points 5 order 4 auto 5`
Beispiel:

1. `set contour; set cntrparam linear; splot x*y with lines` ... die Konturen einzeichnen und gleiche z-Werte linear verbinden
2. `set cntrparam order 8` ... bsplines der Ordnung 8 verwenden, um gleiche z-Werte zu verbinden
3. `set cntrparam levels auto 10` ... 10 Konturlinien wenn es sich mit zmax - zmin in Schritten von 10-er Potenzen ausgeht
4. `set cntrparam levels discrete 20,40,60,90,120,150,180` ... Konturlinien bei den angegebenen diskreten z-Werten (in splot)

5. `set cntparam levels incremental 0,1,4` ... von $z = 0$ bis $z = 4$ mit Schritten von $\Delta z = 1$

Befehl: `set colorbox`

set colorbox

```
set colorbox [horizontal | vertical]
             {default | user [{origin x,y} & {size x,y}]}
             {border [linestyle] | bdefault | noborder]}
```

zeigt den Farbverlauf in den aktuellen pm3d Palettenfarben an (**siehe: set palette (S. 37)**); *horizontal* oder *vertical* (Standard) geben die Richtung an; die genaue Position wird entweder durch *default* (automatisch, Standard) bestimmt oder mit *user* und den Angaben für *origin* und *size* angegeben (**siehe: Koordinatensysteme (S. 8)** – Standard ist das screen Koord.sys.).
Die Rahmenart für die colorbox wird bestimmt durch:

- *bdefault* ... (Standard) zeigt einen Standardrahmen an
- *noborder* ... schaltet den Rahmen aus
- *border* ... zeigt den Rahmen auch an zusammen mit einem gewählten Linienstil (**siehe: set style line (S. 42)**)

Für Einstellungen zur colorbox Achsenskala **siehe: set cbtics (S. 23)**, der Beschriftung **siehe: set cblabel (S. 23)**, oder dem Wertebereich **siehe: set cbrange (S. 23)**.
Beispiel:

1. `set colorbox horizontal user origin 0.1,0.02 size 0.8,0.04` ...
   eine horizontale colorbox unterhalb der x-Achse
2. `set style line 123 linewidth 0.2 linecolor 3;`
   `set colorbox border 123` ... verwendet den zuvor definierten Linienstil „123“ für die colorbox Begrenzung

Befehl: `set contour`

set contour

```
set contour [base | surface | both]
```

zur Verwendung von Konturlinien bei splot (mit diskreten Datenpunkten, **siehe: set dgrid3d (S. 28)**). Bei der Option *base* (Standard) werden Konturlinien nur auf die Gitterebene gezeichnet (entspr. xy-Ebene, **siehe: set ticslevel (S. 46)**), bei *surface* nur auf die Oberfläche selbst (bei $z = 0$), bei *both* auf alle beiden Arten.

Befehl: `set data style`

set data style

`set data style` ... **siehe: set style data (S. 42)**
veraltete Syntax, in neuen gnuplot Versionen unter `set style data`

Befehl: `set datafile` **set datafile**

```
set datafile [ binary | commentschars "z" |
               fortran | missing "z" |
               separator ["z" | whitespace]
             ]
```

legt fest, wie Dateien für (s)plot ausgelesen werden und wie die Spalten organisiert sind

- *binary* ... Datei enthält binäre Daten (Alternative zu `(s)plot ''datei'' binary`, siehe: `help binary`)
- *commentschars "Zeichen"* ... gibt das/die Kommentarzeichen bekannt, wenn das führende Zeichen in einer Zeile eines der angegeben Zeichen ist, wird diese Zeile ignoriert.

  Beispiel:

  1. `set datafile commentschars ''#''` ... wie bei Bash Kommentaren, Zeile "# ein Kommentar" wird ebenso ignoriert wie "# 0 1 2 3", aber nicht "0 1 #2 3"
  2. `set datafile commentschars ''#;!C''` ... alle Zeilen mit einem der angegebenen führenden Zeichen werden ignoriert, z.B. Bash '#', DOS ';', Fortran 'C' und '!' Kommentare

- *fortran* ... Sonderüberprüfung auf 'D' oder 'Q' Zeichen in Wertangaben, sonst können nur 'e' oder 'E' Angaben umgewandelt werden, für gnuplot ist $1.0e+5$ OK, aber $1.0D+5$ ein Fehler, mit `set datafile fortran` sind beide Arten OK
- *missing "Zeichen"* ... fehlende Datensätze werden durch *Zeichen* angezeigt, z.B. "NaN" oder "?"

  Beispiel:

  1. `set datafile commentschars '#'; set datafile missing '#'` ... übergeht auch Zeilen wie "0 1 # 2 3" als Kommentare bzw. als Zeilen mit fehlenden Daten

- *separator* ... Standard *whitespace*, d.h. Spalten sind durch Leerzeichen getrennt, in CSV Dateien besser: `set datafile separator ','` oder `';'`, in Tabulator getrennten Dateien `"\t"`

Befehl: `set decimalsign` **set decimalsign**
`set decimalsign 'z'`
bestimmt das Dezimalzeichen, also `'.'` für Darstellung 1.23 bzw. mit `','` für die übliche Darstellung $1,23$.
Beispiel:

1. `set decimalsign ','; p sin(x)`

Befehl: `set dgrid3d` **set dgrid3d**
`set dgrid3d [Zeilen, Spalten, Gewichtung]`
konvertiert kontinuierliche Daten in Gitterdatenformat für contour plot
Beispiel:

1. ```
   set dgrid3d 100, 100, 1;   # 3D Gitter einstellen
   set contour base;          # Konturlinien
   splot sin(x) * sin(y) w l  # kontinuierliche Funktion zeichnen
   ```

Befehl: `set dummy` **set dummy**
`set dummy [var1, {var2}]`
benennt die dummy Variablen in plot, splot, parametric plot um, d.h. statt $\{x, t, \dots\}$ jetzt neue Bezeichner $\{a, b, \dots\}$ möglich
Beispiel:

1. `set dummy d; plot sin(d)` ... Variable d statt x verwenden
2. `set parametric; set dummy m,n; splot m, n**2, m*n` ...
   parametischer Modus mit Variablen (m, n) statt (u, v)

Befehl: `set encoding` **set encoding**
`set encoding [codepage]`
stellt die Zeichendarstellung nach Standard *codepage* ein, z.B. "default", "cp850", wichtig für Sonderzeichen, für eine Liste von codepages siehe: `help set encoding`

Befehl: `set fit` **set fit**
`set fit {logfile "Datei"} {errorvariables | noerrorvariables}`
kontrolliert das Verhalten der `fit` Funktion (**siehe: fit (S. 9)**), statt Standard *fit.log* Datei kann z.B. `set fit logfile '/tmp/muell.log'` angegeben werden. Mit Option `set fit errorvariables` werden die Fehler der Fitparameter unter ihrem Namen plus `'_err'` gespeichert, und können z.B. in plots benutzt werden.
Beispiel:
   ```

1. `set fit errorvariables`
   `fit f(x) "daten.dat" u 1:2 via a,b`
   `print "Fehler von Parameter a = ", a_err`
   `print "Fehler von Parameter b = ", b_err`

Befehl: `set fontpath` **set fontpath**
`set fontpath [''pfad1'' & ''pfad2'' & ...]`
zusätzliche Schriftarten an den Orten „pfad1“, „pfad2“ usw. einbinden

Befehl: `set format` **set format**
`set format [Achse] ''Format''`
setzt ein manuelles Zahlenformat für die gewählte *Achse*; Standardformat ist immer „%g“; mögliche Achsen sind {x, x2, y, y2, xy, z, cb}, wird keine Achse angegeben werden alle Achsen umformatiert. Das *Format* wird als String mit `""` oder `''` zusammengesetzt aus "%x.y{e,f,g,x,P}" (siehe: `help format specifiers`), wobei x = Anzahl der Stellen, y = Anzahl der Nachkommastellen, e = Exponentialdarstellung, f = Fließkommadarst., g = die kürzere Darst. von e oder f, x = Hexadezimaldarst., P = Vielfache von Pi
Beispiel:

1. `set format x ''%.3P''; plot [-2*pi:2*pi] sin(x)` ... Skalierung der x-Achse mit 3 Nachkommastellen in Einheiten von π
2. `set format x ''%.3P''; set xtics 0.5*pi; rep` ... auch Skalenstriche in Einheiten von π darstellen, vgl. zu vorher
3. `set format x ''%f %%''; rep` ... Zahlenformat der x-Achse mit Prozentangaben (doppeltes %% weil spezielles Zeichen)

Befehl: `set function style` **set function style**
`set function style` ... **siehe: set style function (S. 42)**
veraltete Syntax, in neuen gnuplot Versionen unter `set style function`

Befehl: `show functions` **show functions**
`show functions`
alle selbstdefinierten Funktionen anzeigen, **siehe: save functions (S. 11)** zum Speichern von selbstdefinierten Funktionen

Befehl: `set grid` **set grid**

```
set grid {{no}{m}{x | x2 | y | y2 | z | cb}tics}
         {polar "Winkel"}
         {layerdefault | front | back}
         {(linestyle | ls) &
          (linetype  | lt) &
          (linewidth | lw) ["Hauptskala" {, "Nebenskala"}]
         }
```

zeichnet das Gitternetz für die Haupt-/Nebenskala. Es können {xtics, x2tics, ytics, y2tics, ztics, cbtics} für die Hauptskala (major tics), und {mxtics, mx2tics, mytics, my2tics, mztics, mcbtics} für die Nebenskala (minor tics) separat eingestellt werden, bzw. über no{x} für Achse {x} verworfen werden.
Im polar Modus (**siehe: set polar (S. 40)**) kann über *polar "winkel"* ein Gitter in Polarkoordinaten für Segmente von jeweils *Winkel* Einheiten angezeigt werden (Standard: 30 Grad Polarwinkel bzw. der analoge Wert in Radians) (**siehe: set angles (S. 20)**).
Über *front* bzw. *back* wird das Gitter in einer Ebene über bzw. unter der Zeichenebene dargestellt (in 2D Standard = back, in 3D andere Methode), sodass andere Bildteile nicht überlagert werden. Achsenstile können mit linestyle (und linewidth, linetype, linecolor, **siehe: set style line (S. 42)**) für major und minor tics eingestellt werden.
Beispiel:

1. `set grid; p sin(x)` ... zeigt x, y Hauptskala an
2. `set xtics 1; set mxtics 2; set grid mxtics; rep` ... zeigt Haupt- und Nebengitter (**siehe: set mxtics (S. 34)**) auf x-Achse an
3. `set angles degrees; set polar; set grid polar; plot sqrt(t)` ... Polardarstellung der Fkt. $\sqrt{t}$ mit polarem Gitter
4. `set grid polar 10; rep` ... polares Gitter mit Zonen alle 10 Grad, vgl. zu vorher

Befehl: `set hidden3d` **set hidden3d**
`set hidden3d [defaults | Optionen]`
kontrolliert den Algorithmus zur Erkennung von Überschneidungen versch. Flächen bzw. Linien in splot; für weitere Optionen siehe Hilfethema: `help hidden3d`.
Beispiel:

1. `splot sin(x) * (y*y) with lines` ... Funktion darstellen, keine Überprüfung auf verdeckte Flächenteile
2. `set hidden3d; rep` ... vgl. zu vorher

Befehl: `set historysize` **set historysize**

`set historysize ''wert''`
den gnuplot Befehlsspeicher auf „Wert“ Zeilen begrenzen.

Befehl: `set isosamples` **set isosamples**
`set isosamples [xval, yval]`
kontrolliert die Gitterfeinheit bei splot für kontinuierliche Funktionen; Standard sind 10 Stützstellen pro Achse (10 x 10 insgesamt). Über *xval* und *yval* kann das Gitter modifiziert werden. Wenn kein *yval* angegeben wurde wird der *xval* Wert auch für y verwendet.
Beispiel:

1. `splot x**2 * sin(y)` ... Standard 10x10 isosamples
2. `set isosamples 100; rep` ... höhere Gitterauflösung mit 100 Stützstellen für x-Achse

Achtung: zu hohe Werte von isosamples in Verbindung mit hidden3d bringen eine sehr lange Kaffeepause.

Befehl: `set key` **set key**

```
set key [on | off | default]
        {horizontal | vertical}
        {inside | outside | at "pos"}
        {left | center | right} {top | center | bottom}
        {title "Text"} {enhanced | noenhanced}
        {box [linestyle & linetype & linewidth] | nobox}
```

kontrolliert die Darstellung der Legende, also die Beschreibung zu den mit (s)plot dargestellten Funktionen bzw. Daten (über den angegebenen title "...").
Die Positionierung erfolgt mit *horizontal* oder *vertical*, automatisch innerhalb des Zeichengebiets mit *inside* oder außerhalb der graph Koord. mit *outside*; manuelle Position durch *at x,y* Koord. (**siehe: Koordinaten (S. 8)**). Genauere Angaben über die Kombination von horizontal/vertikal mit Ausrichtung, z.B. *top right* (Standard) oder *bottom center*.
Der Legendentext kann linksbündig (*Left*) oder rechtsbündig (*Right*, Standard) eingestellt werden, *box* zeichnet ein umgebendes Rechteck um die Legende (für Formatierungen **siehe: set style line (S. 42)**); die Legende kann einen eigenen Titel über *title ”Text”* erhalten (siehe Hilfethema: `help enhanced`), für weitere Optionen siehe Hilfethema `help set key`.
Beispiel:

1. `set key inside top center box Left title "Legende";`
   `p x**2 * sin(x)` ...
   einige Optionen einstellen

Befehl: `set label` **set label**

```
set label "Nummer" ["Text" {at "Pos"}]
          {left | center | right}
          {rotate [by "Winkel"] | norotate} {enhanced | noenhanced}
          {front | back}
          {offset "Wert"}
          {textcolor "Farbe" & font "Name, Grösse"}
```

Felder für Beschriftungen anlegen. Über den Bezeichner *Nummer* (eine positive ganze Zahl) kann das Beschriftungsfeld nachträglich verändert werden – wird keine Nummer angegeben, dann verwendet gnuplot intern eine automatisch vergebene Nummer, sodass einmal definierte Felder nicht überschrieben werden.
Der *Text* kann über *at xpos, ypos(, zpos)* beliebig in den versch. Koord.sys. positioniert werden (**siehe: Koordinatensysteme (S. 8)**). Mit *left*, *center*, *right* wird die Textrichtung rel. zur Position bestimmt, *rotation* und *enhanced* erlauben mehr Textformatierungen, ebenso *textcolor* (**siehe: set palette (S. 37)**) und *font*. *front* setzt die Beschriftung als oberste Ebene, damit sie nicht verdeckt werden kann, mit *back* (Standard) bleibt die Beschriftung ev. unter dem Bild verdeckt.

Befehl: `set lmargin` **set lmargin**
`set lmargin` ... **siehe: set margin (S. 33)**
kontrolliert den linken Bildrandabstand

Befehl: `set loadpath` **set loadpath**
`set loadpath ''pfad1'', ''pfad2'', ...]`
sucht Dateien für `call`, `load`, `(s)plot` zusätzlich an den durch „pfad1“, „pfad2“, ... angegebenen Orten

Befehl: `set locale` **set locale**
`set locale ''Typ''`
Umgebungseinstellung z.B. zu Sprache oder Datumsformat

Befehl: `set logscale` **set logscale**
`set logscale [Achse] [Basis]`
stellt die logarithmische Skalierung für die angegebene *Achse* {x, x2, y, y2, z, cb} ein, verwendet den Logarithmus zur angegebenen *Basis* (Standard: 10 = dekadischer Log.)
Beispiel:

1. `set logscale z` ... logarithmische Skalierung für z-Achse

2. `set logscale zcb` ... logarithmische Skalierung für z- und colorbox-Achse

3. `set logscale y 2; plot exp(x)` ... logarithmische Skalierung zur Basis 2 für y-Achse

Befehl: `set mapping` **set mapping**

`set mapping [cartesian | spherical | cylindrical]`

setzt das Koordinatensystem für die Eingabedaten für splot, Standard ist *cartesian* = kartesisches Koord.sys. (x, y, z). Wenn die Eingabedaten in einem speziellen Koord.sys. vorliegen, kann hier auf sphärische (*spherical*) oder zylindrische (*cylindrical*) Koord. umgeschaltet werden.

Befehl: `set margin` **set margin**

`set margin [Rand]`

gilt auch für: `bmargin`, `lmargin`, `rmargin`, `tmargin`

stellt die Bildrandabstände von automatisch berechneten auf manuelle Werte um, z.B. um mehr Platz zu haben oder den Platz besser auszunutzen.

Beispiel:

1. `plot sin(x); set bmargin 10; rep` ... vgl. vorher und nachher für den unteren Bildrand

2. ```
   set multiplot layout 2,1  # Bilder in 2 Zeilen und 1 Spalte
   set bmargin 0             # unterer Rand 0
   set format x ""           # keine x-Werte anzeigen
   plot sin(x)
   unset bmargin             # unterer Rand automatisch
   set tmargin 0             # oberer Rand 0
   unset format              # x-Werte anzeigen
   plot cos(x)
   ```

   2 Bilder auf Abstand 0 zusammenschieben, sodass sie wie ein Bild erscheinen (**siehe: set multiplot (S. 33)**, **siehe: set format (S. 29)**)

Befehl: `set mcbtics` **set mcbtics**

`set mcbtics` ... **siehe: set mxtics (S. 34)**

für Nebenskala (minor tics) der colorbox (cb) Achse

Befehl: `set multiplot` **set multiplot**

`set multiplot [layout zeilen, spalten]`
   ```

```
{rowsfirst | columnsfirst}
{downwards | upwards}
{scale x, y}
{offset x, y}
{title "Text"}
```

mehrere Bilder auf einer Seite darstellen. Die grobe Anordnung wird mit *layout Zeilen, Spalten* festgelegt, mit *rowsfirst* werden zuerst die Zeilen gefüllt, *columnsfirst* füllt also zuerst die Spalten. Die Reihenfolge der Bilder wird über *downswards* (von oben nach unten) bzw. *upwards* (von unten nach oben) kontrolliert (Standard: rowsfirst, downwards). *scale* skaliert das gesamte Zeichenfeld, *offset* verschiebt das Zeichenfeld, und *title "Text"* betitelt das multiplot Arrangement, **siehe: set origin (S. 36)** und **siehe: set size (S. 41)** für die manuelle Positionierung der Teilbilder.
Beispiel:

1. `set multiplot layout 3,2 columnsfirst title "Sinus-Funktionen"`
   `p sin(x);  p sin(2*x); p sin(3*x); \`
   `p sin(4*x); p sin(6*x); p sin(10*x)`
   `unset multiplot`

Befehl: `set mx2tics` **set mx2tics**
`set mx2tics` ... **siehe: set mxtics (S. 34)**
für Nebenskala (minor tics) der x2 Achse

Befehl: `set mxtics` **set mxtics**
`set mxtics [default | sub]`
gilt auch für: `mcbtics`, `mx2tics`, `mytics`, `my2tics`, `mztics`
für Nebenskala (minor tics) der x Achse; *sub* kontrolliert wie viele Subintervalle zwischen je zwei major tics eingefügt werden, *default* stellt den Standardwert wieder her.
Beispiel:

1. `set mxtics 10; set xtics 10; p [0:50] sin(x)` ... Fkt. im Intervall 0 bis 50 mit Hauptskala in Schritten von 10 Einheiten und Nebenskala in Schritten von $10/10 = 1$ Einheiten

Befehl: `set my2tics` **set my2tics**
`set my2tics` ... **siehe: set mxtics (S. 34)**
für Nebenskala (minor tics) der y2 Achse

Befehl: `set mytics` **set mytics**

`set mytics` ... **siehe: set mxtics (S. 34)**
für Nebenskala (minor tics) der y Achse

Befehl: `set mztics` **set mztics**
`set mztics` ... **siehe: set mxtics (S. 34)**
für Nebenskala (minor tics) der z Achse

Befehl: `set object` **set object**

```
set object {"Nummer"} [rectangle]
           {{from Pos1 to|rto Pos2} | {{at | center} Pos1 size x,y}}
           {front | back | behind}
           {{default} | {(fillcolor | fc) "Farbe" &
                         (fillstyle | fs) "Stil" &
                         (linewidth | lw) "Breite" & ...
                        }
           }
```

definiert ein Objekt mit Bezeichner *Nummer* (nur ganzzahlige Werte oder leer) für alle folgenden 2D Zeichnungen. In gnuplot Version 4.2 sind nur Objekte *rectangle* möglich, in höheren Versionen kommen viell. Objekte circle, ellipse, polygon dazu.
Ein Objekt von Typ rectangle wird durch Angabe von Punktepaar `from x1,y1 to x2,y2` für diagonale Ecken (**siehe: Koordinaten (S. 8)**) bzw. durch Angabe von `center xc,yc size b,h` (*at* ist gleich zu *center*) für das Zentrum (xc, yc) und die Breite b, Höhe h festgelegt.
Die Zeichenebene des Objekts wird durch *front* (Vordergrund), *back* (Untergrund) oder *behind* (Hintergrund – z.B. für farbige Zeichenfläche) bestimmt.
Die Option *default* übernimmt voreingestellte Eigenschaften wie Linienbreite und -farbe (**siehe: set style rectangle (S. 42)**), alternativ dazu sind eigene Einstellungen für fc, fs, lw, etc. möglich.
Beispiel:

1. `set object 1 rectangle from screen 0,screen 0 to screen 1,screen 1\`
   `behind fillcolor rgb "yellow"` ... ein komplett gelber Hintergrund
2. `set object 2 rect from graph 0,graph 0 to graph 1,graph 1 behind\`
   `fc rgb "cyan"` ... ein farbiger Hintergrund nur für den Zeichenbereich
3. `set object 3 rect from 0,0 to 5,5 default` ... ein Quadrat mit Seitenlänge 1 in x-Einheiten
4. `set obj 4 rect at 0,0 size 2,2 fc rgb "red"lw 3;`
   `set obj 5 rect at 0,0 size 4,4 fc rgb ”blue” behind; p x, -x` ...

zwei farbige Rechtecke von versch. Größe in versch. Ebenen, Objekt 5 verdeckt Obj. 4 nicht, weil mit Option behind hinter Obj. 4 verschoben

Befehl: `set offsets` **set offsets**

`set offsets [links, rechts, oben, unten]`
erweitert den Wertebereich um den Graphen (nur in plot) um die angegebenen Werte (statt autoscaling), d.h. xrange wird um *links* bei negativen Werten vergrößert, um *rechts* bei positiven Werten, analog wird yrange um *oben* und *unten* erweitert.
Bei positiven Offset-Werten wird der Wertebereich absolut immer grösser, negative offsets vermindern ihn.
Beispiel:

1. `set offsets 0,0,2,1; plot sin(x)` ... Graph im y-Bereich von $[-2:3]$ statt von $[-1:1]$ mit autoscaling darstellen
2. `set offsets 0,0,2,-1; rep` ... jetzt wird der Graph auf $[0:2]$ begrenzt (negative offset Werte verkleinern yrange)

Befehl: `set origin` **set origin**

`set origin [x,y]`
setzt den Ursprungspunkt für die Zeichenebene in screen Koordinaten (**siehe: Koordinaten (S. 8)**, **siehe: set size (S. 41)**).
Beispiel:

1. manuelle Positions- und Größenangaben

```
set multiplot
set origin 0.0,0.2; set size 1.0,0.8; p sin(x)
set origin 0.7,0.0; set size 0.3,0.3; p cos(x)
unset multiplot
```

Befehl: `set output` **set output**

`set output ''Datei''`
gibt eine Datei an, in welche die Anzeige umgeleitet wird. Wichtig im Zusammenhang mit `set terminal` (**siehe: set terminal (S. 45)**).
Beispiel:

1. `set output 'fig1.eps'`
2. `set output 'graph2.png'`

3. `set output -` ... ohne Angabe von *Datei* oder mit `"-"` wird an die Standardausgabe des aktiven Terminals gesendet

Befehl: `set palette` **set palette**

```
set palette [color | gray {gamma "Wert"}]
            { rgbformulae r,g,b |
              defined [ (w1 f1 {, w2 f2, ... }) ] |
              file "Datei" |
              functions r,g,b
            }
            {model { RGB | HSV | CMY | YIQ | XYZ }}
            {positive | negative }
            {maxcolors "Zahl"}
```

enthält Einstellungen für die Umwandlung von kontinuierlichen Werten in Farben oder Graustufen. Eine Liste von palette-kompatiblen Terminaltypen gibt das Hilfethema `help set pm3d`.
Grundsätzlich kann eine *color* oder *gray* Palette benutzt werden; nur für gray gibt es eine Gammakorrektur mit *gamma*, der *Wert* (Standard = 1.5) verändert die Lage des mittleren Grauwertes (g = 0.5) zwischen Weiß (> 1) und Schwarz (< 1).
Die Umwandlung von Grauwerten in Farbwerte geschieht mittels analytischer Formeln (rgbformulae, functions) oder diskreter (und interpolierter) Wertetabellen (defined, file).

- *rgbformulae r,g,b* ... wählt drei aus 37 versch. Funktionen aus, wie die (r = Rot, g = Grün, b = Blau)-Werte aus dem Intervall $[0:1]$ berechnet werden (Standard = 7,5,15, siehe `show palette rgbformulae` für die expliziten Formeln).

- *functions r,g,b* gibt drei selbstdefinierte Funktionen an, nach denen Rot, Grün und Blau aus der Variable „gray“ (aus Intervall $[0:1]$) berechnet werden.

- *defined (Wert Farbe)* ... Palette wird aus der Liste von eingegebenem *Wert* und passender *Farbe* interpoliert. Die Farbe kann auf mehrere Arten angegeben werden: als drei „r g b“ Werte aus $[0:1]$, als Farbname (siehe `show palette colornames`), oder im HTML-Farbformat „#rrggbb“ mit Hexadezimalen Werten.

- *file Datei* ... wie `set palette defined`, liest aber die einzelnen Farben aus *Datei*, diese enthält 3 Spalten für (R,G,B) oder 4 Spalten für (Grau,R,G,B).

Über *model Typ* stehen unterschiedliche Farbräume zur Verfügung. Der Farbraum ist normalerweise RGB, andere Farbräume wie CMY (cyan, magenta, yellow), HSV (hue, saturation, value) etc. sind manchmal nützlich. Alle Farbangaben der Art „r g b“ funktionieren genauso mit anderen Farbräumen, nur dass dann Werte für den entsprechenden Buchstaben festgelegt werden.

Die Anzahl der verschiedenen Farben in einer Palette wird über *maxcolors nbr* geregelt. *positive* ist die natürliche Palette, *negative* kehrt die Farbrichtung um.
Beispiel:

Beispiele mit `test palette` ausprobieren

1. `set palette gray gamma 2.2` ... eine Graustufenpalette mit einem gestauchten Schwarzbereich
2. `set palette rgbformulae 1,2,3` ... Palette durch Funktionen 1,2,3 bestimmt
3. `set palette functions gray,gray**2,gray**3` ... Palette nach Potenzen von gray
4. `set palette model HSV functions gray,1,1 negative` ... dem sichtbaren Spektrum nachempfundene Palette
5. eine „Ampelfarben-Palette“

```
set palette define ( 0.0 'green', 0.2 'white', \
                     0.4 'yellow', 0.6 'orange', \
                     0.8 'light-red', 1.0 'dark-red' )
```

6. `set palette defined (0 1 0 0, 0.5 'yellow', 1 ''#00ff0a'')` ... ein Mix von Farbangaben (0 = rot, 0.5 = gelb, 1 = grün mit etwas blau)
7. `set palette file 'mycolors.pal' using 1:2:3` ... Palette aus Datei „mycolors.pal“ einlesen
8. `set palette gray negative maxcolors 16` ... Palette mit 16 versch. Grauwerten in „negativer“ Farbrichtung = von hell (1) nach dunkel (0)

Befehl: `show palette`

show palette

```
show palette { colornames | gradient |
               fit2rgbformulae | rgbformulae |
               palette
             }
```

Siehe Beispiele für die Auswirkungen der Befehle.
Beispiel:

1. `set palette` ... Standardeinstellungen wiederherstellen

2. `show palette` ... listet aktuelle Einstellungen auf
3. `show palette colornames` ... listet von gnuplot benannte Farben auf
4. `show palette gradient` ... zeigt den Farb-/Graustufengradienten an
5. `show palette fit2rgbformulae` ... versucht zu einer nicht-rgbformulae – z.B. zu einer durch defined oder file gegebenen – Palette die passenden rgbformulae zu finden; gibt die Nummern der Funktionen zurück.
6. `show palette rgbformulae` ... zeigt die Transformationsformeln Graustufen $\mapsto$ Farbe an.
7. `show palette palette [nbr] {float | int}` ... zeigt die Palette für *nbr* verschiedene Farben an, für $n = \{0, 1\}$ werden alle Farbdefinitionen angezeigt, $n = 2$ wäre eine zweifarbige (monochrome schwarz-weiß) Palette. Wahlweise Darstellung der Palette mit *float* od. *int*eger Werten für die RGB Paare.
8. `test palette` ... zeigt ein Testbild für die aktuelle Palette an.

Befehl: `set parametric` **set parametric**

`set parametric`

mit dieser Option sind parametrische Darstellungen mit plot und splot möglich, d.h. die x, y Koordinaten werden als Funktionen eines Parameters t (für plot) bzw. als Funktionen zweier Parameter u, v (für splot) angegeben. Statt t bzw. u, v können auch andere Parameternamen verwendet werden, **siehe: set dummy (S. 28)**.
`plot [x-func(t), y-func(t)]` in Variablen $(t, f(t))$
`splot [x-func(u,v), y-func(u,v), z-func(u,v)]` in Variablen $(u, v, f(u, v))$
Voraussetzung für Beispiele: vorher `set parametric` eingeben. **Beispiel:**

1. `plot sin(t),cos(t)` ... zeichnet einen Kreis
2. `plot sin(t),t**2` ... zeichnet eine Lissajous-artige Kurve
3. `splot cos(u)*cos(v), sin(u)*cos(v), sin(v)` ... zeichnet eine Kugel

Befehl: `show plot` **show plot**

`show plot`

zeigt den letzten `plot` Befehl an

Befehl: `set pm3d` **set pm3d**

```
set pm3d {at [(bottom | b) | (surface | s) | (top | t)]}
         {interpolate val1, val2}
         {map}
         {scansautomatic | scansforward | scansbackward | depthorder}
         {flush [begin | center | end]}
         {explicit | implicit}
```

pm3d Modus für Farbkarten von 3D/4D Daten; funktioniert mit Terminals x11 (Unix), windows (Windows), aquaterm (OS X), postscript (inkl. eps), (e)pslatex, png, jpeg, pdf, svg (siehe Hilfethema: `help pm3d`).
pm3d Farbfläche bei Option *implicit* (oder `set style {data|function} pm3d`) immer mit Ursprungsdaten kombiniert, bei *explicit* (Standard) zeichnet splot nur dann die Farbfläche, wenn das über `with pm3d` angefordert wird (nützlich bei mehreren Datenquellen in einem Bild).
Die Farbfläche kann über *b,s,t* oder Kombinationen davon positioniert werden, bei *bottom* unter die z-Werte der Daten, mit *surface* auf die z-Werte, durch *top* über die z-Werte der Daten, *bt* (= bottom and top) kombiniert Farbflächen unter und über den Daten.
interpolate steps in scan, steps between scans (?) erhöht die Anzahl der Gitterpunkte durch Interpolation, *map* projiziert alle Werte auf eine Ebene und zeigt keine explizite z-Koordinate an, implizit über Grau-/Farbwerte (siehe: set view), scansX legt die Art fest, wie Datenpunkte eingelesen werden, dadurch kann bei sich schneidenden Flächen die Darstellung verbessern (**siehe: set hidden3d (S. 30)**).

Befehl: `set pointsize` **set pointsize**
`set pointsize "Wert"`
skaliert die Symbolgröße in plot, *Wert* multipliziert die Symbolgröße.
Beispiel:

1. `set pointsize 2.0` ... doppelte Größe
2. `set pointsize 0.5` ... halbierte Größe

Befehl: `set polar` **set polar**
`set polar`
wechselt von kartesischen Koord. zu Polarkoord. (**siehe: set grid polar (S. 29)** um Achsenskala ebenfalls in Polarkoord. darzustellen). Standardwerte für polar sind Winkel-Variable *t* und kein Gitter (bzw. kartesisches Gitter).
Beispiel:

1. `set polar; set grid polar; plot sqrt(t), -sqrt(t)` ...
   Polarkoordinaten und polares Gitter für die angegebenen Funktionen

Befehl: `set print` **set print**
`set print ''Datei''`
gibt die Ausgabe eines `print` Kommandos in *Datei* aus

Befehl: `set rmargin` **set rmargin**
`set rmargin` ... **siehe: set margin (S. 33)**
kontrolliert den rechten Bildrandabstand

Befehl: `set rrange` **set rrange**
`set rrange` ... **siehe: set xrange (S. 50)**
für Wertebereich der radialen Koord. r in polar mode (**siehe: set polar (S. 40)**)

Befehl: `set samples` **set samples**
`set samples [val1 {, val2}]`
Anzahl der Stützpunkte bei der Auswertung von Funktionen (Standard = 100), für plot ist nur der Wert *val1* wichtig, in splot können beide Werte eingestellt werden
Beispiel:

1. `show samples; plot sin(x) w lp` ... Standardsampling (100 Punkte), Symbole zeigen die Lage der Samplingpunkte an
2. `set samples 500; rep` ... Sampling auf 500 Punkte erhöht, vgl. zu vorher
3. `set samples 5; rep` ... Sampling besteht jetzt aus 5 Punkten, vgl. zu vorher

Befehl: `set size` **set size**
`set size [ratio ''r'' | noratio | square | xs,ys]`
stellt die Größe der gesamten Bildebene (inkl. Achsenbeschriftungen, Titel, Graphen) ein, unabhängig von `set terminal size tx,ty`, mit `set size 1,1` wird die vollständige von *tx,ty* vorgegebene Fläche benutzt, für $xs, ys < 1$ wird nur ein Teil des verfügbaren Bereichs ausgenutzt, mit $xs, ys > 1$ wird nur ein Teil der Graphik dargestellt.
ratio ist die Angabe für das *ys/xs* Verhältnis, es hat keine Auswirkung auf 3D splots. Mit ratio < 1 wird das Bild breiter, mit ratio > 1 wird das Bild höher; negative ratio Werte betreffen die Achseneinheiten, d.h. ratio = -1 stellt gleiche Einheiten auf der x und y Achse her.
Beispiel:

1. `set size ratio 1` ... quadratisches Bild, synonym zu *square*.
2. `set size square` = `set size 1,1` ... quadratisches Bild
3. `set size 0.5,1` ... halbe Breite bei voller Höhe

4. `set size ratio 2` ... Bild doppelt so hoch wie breit

Befehl: `set style` **set style**

```
set style [ arrow | data | fill | function |
            histogram | increment | line | rectangle
          ]
```

erstellt oder ändert bereits vorgegebene Stile. Ein zentraler Punkt, um verschiedene Stile zu verwalten, viele gnuplot Funktionen und Optionen greifen auf diese Einstellungen zu.

- *arrow* ... ändert den Stil für den Pfeil mit Bezeichner *nr*; für alle Einstellungen **siehe: set arrow (S. 20)**. **style arrow**

  ```
  arrow "nr" [default | head | heads | nohead]
             {size len, ang}
             {empty | filled | nofilled}
             {front | back}
             {(linestyle | ls) &
              (linetype  | lt) &
              (linewidth | lw)
             }
  ```

- *data [Stil]* ... setzt global für aus Dateien gelesene Daten den *Stil* fest, für mögliche Stile **siehe: plot with (S. 15)**, **siehe: splot with (S. 19)**. **style data**

- *fill* ... ändert den globalen Füllstil für versch. Objekte, z.B. boxes, histograms, filledcurves, rectangles etc. **style fill**

  ```
  fill {empty | solid "dichte" | pattern {"nr"}}
       {border "typ" | noborder}
  ```

  `fill` füllt eine Fläche mit Farbe, Standard = *empty*. Mit *solid* wird die Fläche mit einer festen Farbe mit einer „Farb-Dichte“ zwischen 0 und 1 gefüllt, 0 = empty, 1 = voll, Zwischenwerte je nach Terminaltyp. *pattern* verwendet ein Muster (bzw. das Muster mit Nummer *nr*, falls angegeben) je nach Terminaltyp. Zusatz von *border* rahmt die Box mit dem aktuellen Linientyp ein bzw. mit *Typ* wie extra angegeben (siehe: set style line).

  Beispiel:

1. `set style fill solid 0.5 noborder` ... halbtransparente Füllung (Wert 0.5)
2. `set style fill pattern 1 fc rgb "red"` ... Füllung mit Muster Nummer 1 mit roter Farbe

- *function [Stil]* ... setzt global für Funktionen den *Stil* fest, für mögliche Stile **siehe: plot with (S. 15)**, **siehe: splot with (S. 19)**. **style function**

- *histogram* ... Einstellungen für den Histogramms Stil **style histogram**

```
histogram { clustered {gap "wert"} |
            errorbars {gap "wert" & linewidth}
          }
          {columnstacked | rowstacked}
```

abh. von der Anzahl eingelesener Spalten erstellt *clustered* eine Gruppe von Blöcken um $x = 1$ (für die erste Zeile), durch einen Abstand getrennt folgen danach die Blöcke um $x = 2$ (zweite Zeile) usw. Mit *gap* wird ein anderer Abstand eingestellt; Standardwert gap = 2 bedeutet, der Abstand entspricht 2 Blockbreiten.

errorbars benötigt jeweils 2 Spalten für jede Histogrammbox, die erste Spalte ist der y-Wert, die zweite Spalte der Fehler Δy. *gap* und *linewidth* kontrollieren das Aussehen.

Histogramme können auch gestapelt (stacked) sein, die beiden Optionen sind: *columnstacked* zur Auswahl einer Spalte, alle Zeilen werden für diese Spalte aufeinandergestapelt; *rowstacked* stapelt alle ausgewählten Spalten einer Zeile aufeinander.

Beispiel:

1. `set style data histogram; set style histogram clustered gap 3; p ''hist.dat'' u 2, '' u 3, '' u 4` ... Spalten 2, 3 und 4 für Histogramm auswählen, Abstand auf 3 Blockbreiten erhöhen
2. `set style data histogram; set style histogram errorbars; p ''hist-err.dat'' u 1:2, '' u 3:4` ... Fehlerbalken für Histrogramm
3. `set style data histogram; set style histogram columnstacked; p ''hist.dat'' u 2` ... für Spalte 2 alle Zeilen in einem Histogramm bei $x = 1$ aufstapeln, d.h. ein Rechteck wird von $y = 0$ bis zur Höhe des y-Wertes y_1 der ersten Zeile aufgetragen, darauf folgt von $y = y_1$ bis $y = y_2$ ein weiteres Rechteck mit der Höhe des y-Wertes y_2 der zweiten Zeile usw.

- *increment [default | user]* ... ermöglicht bei mehreren plots in einem Bild mehrere Linientypen zu verwenden. Bei *default* (Standard) werden die vordefinierten Typen automatisch verwendet, bei *user* verwendet gnuplot stattdessen bevorzugt selbstdefinierte Typen (falls sie vorher mit `set style line ''nr''` auch definiert wurden). **style increment**

Beispiel:

1. `set style line 1 lw 2 lc rgb "black";`
   `set style line 2 lw 2 lc rgb 'blue';`
   `set style increment user;`
   `p sin(x), cos(x), 1-sin(2*x)`

   ... definiere „eigene" Linienstile 1 und 2 anstatt Standardstilen (diese werden aber nicht überschrieben) und verwende diese in plot, dritte Funktion wird wieder mit Standardstil 3 gezeichnet

- *line* ... ändert den Linienstil, d.h. legt Linienbreite, -farbe und Linienart (dashed, dotted, solid) sowie Symbolart (Dreieck, Kreis, Rechteck, etc.) und -größe fest **style line**

```
line "index" [ default |
               { (linetype  | lt) "typ" &
                 (linecolor | lc) "farbe" &
                 (linewidth | lw) "breite" &
                 (pointtype | pt) "typ" &
                 (pointsize | ps) "groesse"
               } |
               palette
             ]
```

erstellt (oder ändert den zu Nummer *index* gehörenden) Linienstil, *default* stellt den Standard wieder her; *pointsize* hier ist unabhängig von `set pointsize`! Option *palette* gilt für splot und ändert die Linienfarbe gemäß der gewählten Farbpalette.

Beispiel:

1. `set style line 1 linetype -1` ... durchgehende schwarze Linie
2. `set style line 30 linetype 3 linewidth 2` ... anderer Name für Standardtyp 3 mit doppelter Linienbreite
3. `splot x*y with lines palette` ... Farbverlauf auf Linien
4. `set style line 123 lw 2 lc rgb 'gold' pt 7 ps 3;`
   `p sin(x) w lp lt 123` ... Linienstil 123 erstellen und in plot anwenden

- *rectangle* ... globale Einstellungen für mit `set object rectangle` erstellte Objekte **style rectangle**

```
rectangle [front | back]
          { (fillcolor | fc) "farbe" &
            (fillstyle | fs) "stil"
          }
```

Es können später individuell pro Objekt andere Werte festgelegt werden. Mit *front* steht das Objekt im Vordergrund, mit *back* hingegen im Hintergrund. *fillcolor* ist die Füllfarbe, *fillstyle* der anzuwendende Füllstil (siehe: set style fill). Standardwerte sind border (schwarz), solid (Füllung mit Hintergrundfarbe), nopattern.

Beispiel:

1. `set style rectangle back fc rgb "red"fs solid 1.0 border 2` ...
   neue Objekte werden deckend rot gefüllt, erhalten einen Rahmen doppelter Breite und bleiben im Hintergrund

Befehl: `set surface` **set surface**

`set surface`
zeichnet in splot die Oberfläche mit dem durch splot [data, function] with [style] gegebenen Stil, mit unset surface werden keine Punkte oder Linien für die Oberflächen gezeichnet - Konturen aber schon

Befehl: `set table` **set table**

`set table ''datei''`
statt mit plot oder splot eine Graphik kann gnuplot eine Datei ausgeben, welche die Ausgabe der x,y(,z) Koordinaten des (s)plot Befehls enthält
Beispiel:

1. `set table "plot.dat"; plot sin(x); unset table`
   `!more ''plot.dat''` ...
   Datei „plot.dat“ enthält jene Punkte, die der plot Befehl angezeigt hätte

Befehl: `set terminal` **set terminal**

`set terminal [Typ] {enhanced} {Optionen}`
Auswahl eines Zeichenterminals für `gnuplot` .
Option *enhanced* bietet weitere Möglichkeiten für die Textformatierung (hoch-, tiefgestellt, rotiert), mögliche Typen (Auswahl, für weitere Infos siehe Hilfethema: `help terminal Typ`):

- dumb (witzig)
- epslatex
- latex
- postscript
- png

- svg
- windows
- x11

Befehl: `set termoption` **set termoption**

`set termoption {enhanced | noenhanced} {font ''fontname, fontsize''}`
stellt nachträglich zu `set terminal` die Optionen ein.

Befehl: `set tics` **set tics**

`set tics`
gilt auch für: `xtics, x2tics, ytics, y2tics, ztics, cbtics`
kontrolliert die Skalen (major tics) aller Hauptachsen auf einmal, d.h. Einstellungen für (x, x2, y, y2, z, cb) Achse werden hier zentral getroffen. Über `set xtics` usw. individuell noch Feineinstellungen möglich, **siehe: set xtics (S. 51)** für Optionen die auch hier gelten.

Befehl: `set ticslevel` **set ticslevel**

`set ticslevel` ... **siehe: set xyplane (S. 53)**
veraltete Syntax, soll in neuen `gnuplot` Versionen durch `set xyplane` ersetzt werden.

Befehl: `set timestamp` **set timestamp**

```
set timestamp {"format"} {top | bottom}
              {rotate | norotate} {offset x,y}
              {font "fontname"}
```

setzt einen Zeitstempel ins Bild, durch *format* (siehe: `set timefmt`) wird das Datumsformat eingestellt, *top* oder *bottom* (Standard) positionieren den Stempel am linken Bildrand, *rotate* setzt vertikalen Text, *offset* verschiebt den Stempel um x,y Einheiten, *font* wählt eine Schriftart aus.
Beispiel:

1. `set timestamp ''%d/%m/%y'' top offset 10,0` ... setzt den Stempel im Format Tag/Monat/Jahr oben links um 10 Einheiten in positiver x-Richtung verschoben (0 in y-Richtung)

Befehl: `set timefmt` **set timefmt**

`set timefmt ''format''`
zur Einstellung von Zeit-/Datumsformaten und in `set timestamp`.
Mögliche Formatangaben:

- $\%d$ Tag 1 - 31
- $\%m$ Monat 1 - 12
- $\%y$ Jahr zweistellig 0 - 99
- $\%Y$ Jahr vierstellig
- $\%j$ Tag im Jahr 1 - 365
- $\%H$ Stunde 0 - 24
- $\%M$ Minute 0 - 60
- $\%S$ Sekunde 0 - 60
- $\%B$ Monatsname

Befehl: `set title` **set title**

```
set title "text" {enhanced | noenhanced}
                 {offset x,y &
                  font "fontname, fontsize" &
                  (textcolor | tc) "farbe" &
                  (linetype  | lt) "typ"
                 }
```

setzt *text* als einen zentrierten Titel oben ins Bild. Optionen sind *enhanced* für erweiterte Formatierung (besondere Symbole od. Zeichen in *text*), *textcolor* wählt eine *Farbe*, *linetype* einen Linientyp nach definiertem *Typ* (Zahl). Für die manuelle Positionierung einen *offset x,y* angeben, Verschiebung um die angegebenen x-/y-Einheiten
Beispiel:

1. `set title "Das hier ist der Titelöffset 0,-2 font Ärial, 20"` ...
   den Titeltext in Schriftart Arial und Größe 20pt um -2 Einheiten in y-Richtung verschieben (nach unten Richtung Bild)

Befehl: `set tmargin` **set tmargin**
`set tmargin` ... **siehe: set margin (S. 33)**

kontrolliert den oberen Bildrandabstand

Befehl: `set trange` **set trange**
`set trange` ... **siehe: set xrange (S. 50)**
für Wertebereich in parametrischer Koord. t in parametric mode (**siehe: set parametric (S. 39)**).

Befehl: `set urange` **set urange**
`set urange` ... **siehe: set xrange (S. 50)**
für Wertebereich in parametrischer Koord. u in parametric mode (**siehe: set parametric (S. 39)**).

Befehl: `show variables` **show variables**
`show variables {all}`
gibt eine Liste von selbst-definierten Variablen aus, mit Option *all* auch von gnuplot internen Variablen (Prefix `GPVAL`).

Befehl: `show version` **show version**
`show version {long}`
gibt eine Übersicht über die aktuelle gnuplot Version.

Befehl: `set view` **set view**
`set view [xrot, zrot {, scale, zscale} | map]`
bestimmt die Betrachtungswinkel in splot. Standardwinkel sind: xrot = 60° (zwischen [0 : 180°], zrot = 30° (zwischen [0 : 360°]). Option *map* für eine 2D Abbildung (**siehe: set pm3d map (S. 39)**), *scale* skaliert den gesamten Graph um diesen Wert, *zscale* nur die z-Achse.
Beispiel:

1. `set view 30,60, 1.5` ... Rotation 30° um x-Achse, 60° um z-Achse und Skalierung auf 1.5-fach
2. `set view ,180,,0.5` ... Rotation nur um z-Achse, x-y-Skalierung Standard (1-fach), z-Skalierung halbiert

Befehl: `set vrange` **set vrange**
`set vrange` ... **siehe: set xrange (S. 50)**
für Wertebereich in parametrischer Koord. v in parametric mode (**siehe: set parametric (S. 39)**).

Befehl: `set x2data` **set x2data**
`set x2data` ... **siehe: set xdata (S. 49)**
für Datenformat der x2 Achse

Befehl: `set x2dtics` **set x2dtics**
`set x2dtics` ... **siehe: set xdtics (S. 49)**
für Tagesformat der x2 Achse

Befehl: `set x2label` **set x2label**
`set x2label` ... **siehe: set xlabel (S. 50)**
für Achsenbeschriftung der x2 Achse

Befehl: `set x2mtics` **set x2mtics**
`set x2mtics` ... **siehe: set xmtics (S. 50)**
für Monatsformat der x2 Achse

Befehl: `set x2range` **set x2range**
`set x2range` ... **siehe: set xrange (S. 50)**
für Wertebereich der x2 Achse

Befehl: `set x2tics` **set x2tics**
`set x2tics` ... **siehe: set xtics (S. 51)**
für Hauptskala (major tics) der x2 Achse

Befehl: `set x2zeroaxis` **set x2zeroaxis**
`set x2zeroaxis` ... **siehe: set zeroaxis (S. 55)**
für die Anzeige der Parallelen zu der x2 Achse durch $y = 0$

Befehl: `set xdata` **set xdata**
`set xdata {time}`
gilt auch für: `set x2data`, `ydata`, `y2data`, `zdata`, `cbdata`
setzt für die jeweilige Achse (x, x2, y, y2, z, cb) ein normales Datenformat (Standard) bzw. mit *time* das Datumsformat, damit werden die Werte als Zeitangaben aufgefasst und nach Tag/Monat/Jahr konvertiert.
Beispiel:

1. `set xdata time; plot sin(x)` ... Sinus über einer Zeitachse

Befehl: `set xdtics` **set xdtics**

`set xdtics`
gilt auch für: `set x2dtics`, `ydtics`, `y2dtics`, `zdtics`, `cbdtics`
die jeweilige Achsenskala wird in Einheiten von Tagen angezeigt (0 = Sun ... 6 = Sat, Modulo 7)
Beispiel:

1. `set xdtics; plot sin(x)` ... Sinus über einer Tagesachse

Befehl: `set xlabel` **set xlabel**

```
set xlabel "Text"
           {offset "Wert"}
           {font "fontname, fontsize"}
           {(textcolor | tc) "farbe" & (linetype | lt) "typ"}
           {enhanced | noenhanced}
           {rotate by "Winkel"}
```

gilt auch für: `set x2label`, `ylabel`, `y2label`, `zlabel`, `cblabel`
Beschriftung *Text* in Anführungszeichen ("Text" bzw. 'Text') als Achsentitel der jeweiligen Achse. *offset* gibt eine Verschiebung von der Standardposition $(0, 0)$ um diesen (x, y) *Wert* in best. Koordinaten (first, second, graph, screen) an; *font* wählt eine alternative Schriftart und Größe, *enhanced* erlaubt Hoch-/Tiefstellung von Text und spezielle Symbole; *rotation* um *Winkel* dreht die Beschriftung.
Beispiel:

1. `set xlabel ''x-Achse'' offset 0,-1 font ''Courier, 15''` ... Titel lautet „x-Achse“ und wird um -1 y-Einheiten tiefer gesetzt mit Schriftart „Courier“ in Größe 15

Befehl: `set xmtics` **set xmtics**
`set xmtics`
gilt auch für: `set x2mtics`, `ymtics`, `y2mtics`, `zmtics`, `cbmtics`
die jeweilige Achsenskala wird in Einheiten von Monaten angezeigt (1 = Jan ... 12 = Dec, Modulo 12)
Beispiel:

1. `set xmtics; plot sin(x)` ... Sinus über einer Monatsachse

Befehl: `set xrange` **set xrange**

```
set xrange [{min | *}:{max | *}]
           {reverse | noreverse}
           {restore}
```

gilt auch für: `set x2range`, `yrange`, `y2range`, `zrange`, `cbrange`, `rrange`, `trange`, `urange`, `vrange`
Einstellungen für den Wertebereich der jeweiligen Achse. Die Werte werden als Intervall [xmin:xmax] angegeben, für eine teilweise Autoskalierung sind Platzhalter `*` erlaubt. *reverse* vertauscht xmin und xmax, d.h. Wertebereich [xmin:xmax] $\longrightarrow$ [xmax:xmin]; *restore* stellt die letzten gespeicherten Werte wieder her.
Beispiel:

1. `set xrange [xmin:xmax]` ... setzt sowohl xmin als auch xmax auf festen Wert
2. `set xrange [:xmax]` ... lässt xmin unverändert (letzter fester oder automatischer Wert) und setzt xmax fest
3. `set xrange [*:xmax]` ... xmin automatisch skalieren und setzt xmax fest

Befehl: `set xtics` **set xtics**

```
set xtics {axis | border}
          {mirror | nomirror}
          {in | out}
          {scale [default | major, minor]}
          {rotate by "Winkel" | norotate}
          {offset "Wert" | nooffset}
          {font "fontname, fontsize"}
          {textcolor "farbe"}
          { autofreq |
            incr |
            start, incr, end |
            add ("text" "wert") |
            {("text1" "pos1" typ, "text2" "pos2" typ , ...)}
          }
```

gilt auch für: `tics`, `x2tics`, `ytics`, `y2tics`, `ztics`, `cbtics`
legt das Aussehen der Hauptmarkierungen (major tics) der jeweiligen Achse fest, `set xtics` (Standard) zeigt tics an, `unset xtics` lässt sie weg.
Zu den einzelnen Optionen:

- *axis* zeigt die Skala bei 0-Wert der Achse selbst, *border* (Standard) zeigt die Skala nahe am Bildrand.

- *in, out* zeichnet die Skalenstriche innen oder außen
- *(no)mirror* Skalenstriche auf gegenüberliegender Achse nach gleichen Optionen zeichnen bzw. sie weglassen
- *scale* skaliert die Länge der Skalenstriche, Standard = 1.0 für Hauptskala (major tics), 0.5*major für Nebenskala (minor tics), über *major,minor* beide Skalierungsfaktoren getrennt einstellbar
- *rotate* dreht die Beschriftung um 90° (Standard), oder um den mit *rotate by Winkel* angegebenen Wert (falls durch den Terminaltyp unterstützt)
- *offset* versetzt die Beschriftung um $x, y(, z)$ Einheiten
- *autofreq* die Häufigkeit der Markierungen automatisch festlegen (kurz: auto)
- *start, incr, end* tics von Wert *start* bis *end*-Wert in Abständen von *incr*, oder nur *incr* angeben für automatische Wahl von *start,end*
- *add* zusätzliche Markierung bei *Wert* mit Beschriftung *Text* einfügen
- weitere manuelle Skalenwerte durch Format `(text wert typ)` für Beschriftung *Text* an der Position (Koordinatenangabe) *Wert* mit *Typ* = 0 für major oder 1 für minor tics

Anwendungsbeispiele zu den oben genannten Optionen:
Beispiel:

1. `set xtics border; plot sin(x)` ... Standardverhalten
2. `set xtics axis; rep` ... vgl. zu vorher
3. `set xtics out nomirror; set ytics in nomirror; plot sin(x)` ... Skalenstriche innen für y-Achse bzw. außen für x-Achse und nicht gespiegelt
4. `set xtics scale 1.25,0.3` ... längere major und kürzere minor tics
5. `set xtics scale 1,1` ... gleich große major und minor tics
6. `set xtics 2; p sin(x)` ... Sinus im Intervall -10 bis +10 (Standard) mit Markierungen alle 2 Einheiten (incr = 2)
7. `set xtics -5,0.5,5; p sin(x)` ... Markierungen nur im Intervall $x \in [-5:5]$ in Schritten von 0.5
8. `set xtics ("Pi"3.14159)` ... bzw. `('Pi' pi)` mit interner Konstante „pi“
9. `set xtics ('klein' 1, 'mittel' 10, 'riesig' 100);`
   `p [0:100] sin(x)` ... eigene Textbeschriftung statt Standardskala

Befehl: `set xyplane` **set xyplane**
`set xyplane {at zpos | frac}`
setzt die Position der x-y-Ebene in splots fest. Die Option *at z* setzt die absolute Position auf den Wert *zpos*, *frac* gilt für relative Angaben – Abstand der x-y-Ebene zu zmin ist $(zmax - zmin) * frac$.
Beispiel:

1. `set xyplane at 0` ... x-y-Ebene liegt bei Höhe $z = 0$
2. `set xyplane 0.1` ... Abstand 1/10 des gesamten zrange
3. `set xyplane 1.5` ... vergrößert Abstand auf 1,5-fach

Befehl: `set xzeroaxis` **set xzeroaxis**
`set xzeroaxis` ... **siehe: set zeroaxis (S. 55)**
für die Anzeige der Parallelen zu der x Achse durch $y = 0$

Befehl: `set y2data` **set y2data**
`set y2data` ... **siehe: set xdata (S. 49)**
für Datenformat der y2 Achse

Befehl: `set y2dtics` **set y2dtics**
`set y2dtics` ... **siehe: set xdtics (S. 49)**
für Tagesformat der y2 Achse

Befehl: `set y2label` **set y2label**
`set y2label` ... **siehe: set xlabel (S. 50)**
für Achsenbeschriftung der y2 Achse

Befehl: `set y2mtics` **set y2mtics**
`set y2mtics` ... **siehe: set xmtics (S. 50)**
für Monatsformat der y2 Achse

Befehl: `set y2range` **set y2range**
`set y2range` ... **siehe: set xrange (S. 50)**
für Wertebereich der y2 Achse

Befehl: `set y2tics` **set y2tics**
`set y2tics` ... **siehe: set xtics (S. 51)**
für Hauptskala (major tics) der y2 Achse

Befehl: `set y2zeroaxis` **set y2zeroaxis**
`set y2zeroaxis` ... **siehe: set zeroaxis (S. 55)**
für die Anzeige der Parallelen zu der y2 Achse durch $x = 0$

Befehl: `set ydata` **set ydata**
`set ydata` ... **siehe: set xdata (S. 49)**
für Datenformat der y Achse

Befehl: `set ydtics` **set ydtics**
`set ydtics` ... **siehe: set xdtics (S. 49)**
für Tagesformat der y Achse

Befehl: `set ymtics` **set ymtics**
`set ymtics` ... **siehe: set xmtics (S. 50)**
für Monatsformat der y Achse

Befehl: `set ylabel` **set ylabel**
`set ylabel` ... **siehe: set xlabel (S. 50)**
für Achsenbeschriftung der y Achse

Befehl: `set yrange` **set yrange**
`set yrange` ... **siehe: set xrange (S. 50)**
für Wertebereich der y Achse

Befehl: `set ytics` **set ytics**
`set ytics` ... **siehe: set xtics (S. 51)**
für Hauptskala (major tics) der y Achse

Befehl: `set yzeroaxis` **set yzeroaxis**
`set yzeroaxis` ... **siehe: set zeroaxis (S. 55)**
für die Anzeige der Parallelen zu der y Achse durch $x = 0$

Befehl: `set zdata` **set zdata**
`set zdata` ... **siehe: set xdata (S. 49)**
für Datenformat der z Achse

Befehl: `set zdtics` **set zdtics**
`set zdtics` ... **siehe: set xdtics (S. 49)**

für Tagesformat der z Achse

Befehl: `set zero` **set zero**
`set zero ''wert''`
ignoriert Werte mit Absolutbetrag des Imaginärteils größer als *Wert*

Befehl: `set zeroaxis` **set zeroaxis**

```
set zeroaxis { (linestyle | ls) "stil" &
               (linetype  | lt) "typ" &
               (linewidth | lw) "wert"
             }
```

gilt auch für: `xzeroaxis`, `x2zeroaxis`, `yzeroaxis`, `y2zeroaxis`, `zzeroaxis`
zeichnet die Parallele zu der jeweiligen Achse – falls sie im ausgewählten (s)plot Bereich sichtbar ist, d.h. `xzeroaxis` zeichnet bei $y = 0$ eine Achse, `zzeroaxis` zeichnet bei $x = y = 0$ eine Achse
Beispiel:

1. `set xzeroaxis; p sin(x)` ... zeichnet die $y = 0$ Gerade im Bild ein
2. `set yzeroaxis; rep` ... zeichnet bei $x = 0$ die Gerade parallel zur y-Achse ein
3. `set zzeroaxis; set xyplane at -100; sp x*y` ... bei splot die Parallele zur z-Achse durch $(x, y) = (0, 0)$ anzeigen

Befehl: `set zlabel` **set zlabel**
`set zlabel` ... **siehe: set xlabel (S. 50)**
für Achsenbeschriftung der z Achse

Befehl: `set zmtics` **set zmtics**
`set zmtics` ... **siehe: set xmtics (S. 50)**
für Monatsformat der z Achse

Befehl: `set zrange` **set zrange**
`set zrange` ... **siehe: set xrange (S. 50)**
für Wertebereich der z Achse

Befehl: `set ztics` **set ztics**

`set ztics` ... **siehe: set xtics (S. 51)**
für Hauptskala (major tics) der z Achse

Befehl: `set zzeroaxis` **set zzeroa-**
`set zzeroaxis` ... **siehe: set zeroaxis (S. 55)** **xis**
für die Anzeige der Parallelen zu der z Achse durch $(x, y) = (0, 0)$