LINUX MAGAZIN 12/1997 ISSN 1432 - 640 X

Informationen, Berichte und Neuigkeiten aus der LINUX-Welt

LINUX MAGAZIN

12/1997

DM 9,- ÖS 75,- SFR 9,-

Titel story

Eintauchen in Traumwelten

Virtual Reality mit VLE

Linux News
Neues aus dem Internet

3D-Modellierer
GIG3DGO von ElectroGIG

CAD-Systeme
ARCAD und Microstation95

Linux kommerziell
Grundig TV und Raima

Graphik
Entstehung des 3-D-Pinguins

Reihen
System-, KDE-, und Xlib-Programmierung

Perl
Das Chart-Paket

Wissen
Posix- und Java-Threads

Tips und Tricks
Texten mit mpage

Erstes & einziges deutschsprachiges LINUX MAGAZIN

LINUX MAGAZIN 12/1997 ISSN 1432 - 640 X
Informationen, Berichte und Neuigkeiten aus der LINUX-Welt
www.linux-magazin.de

Das Chart-Paket

Balken und Kuchen

von Michael Schilli

Wieviel Plattenplatz hat *Michael Schilli* zuhause noch auf seinem Rechner frei? Diese Frage interessiert definitiv *niemanden*. Doch heute soll es darum gehen, Daten mit dem *Chart*-Package von *David Bonner* augenfreundlich anzuzeigen - und das gleich auf vier verschiedene Arten!

Und da kommt die Ausgabe des *df*-Kommandos, das unter Linux die Auslastung der konfigurierten Platten-Partitionen auflistet, gerade recht:

```
Filesystem 1024-blocks  Used Available Capacity Mounted on
/dev/hda2      839001  597769    197888      71%    /
/dev/hda1      806144  457376    348768      57%
/mnt1
/dev/hdc5     1249696 1213472     36224      97%
/mnt2
```

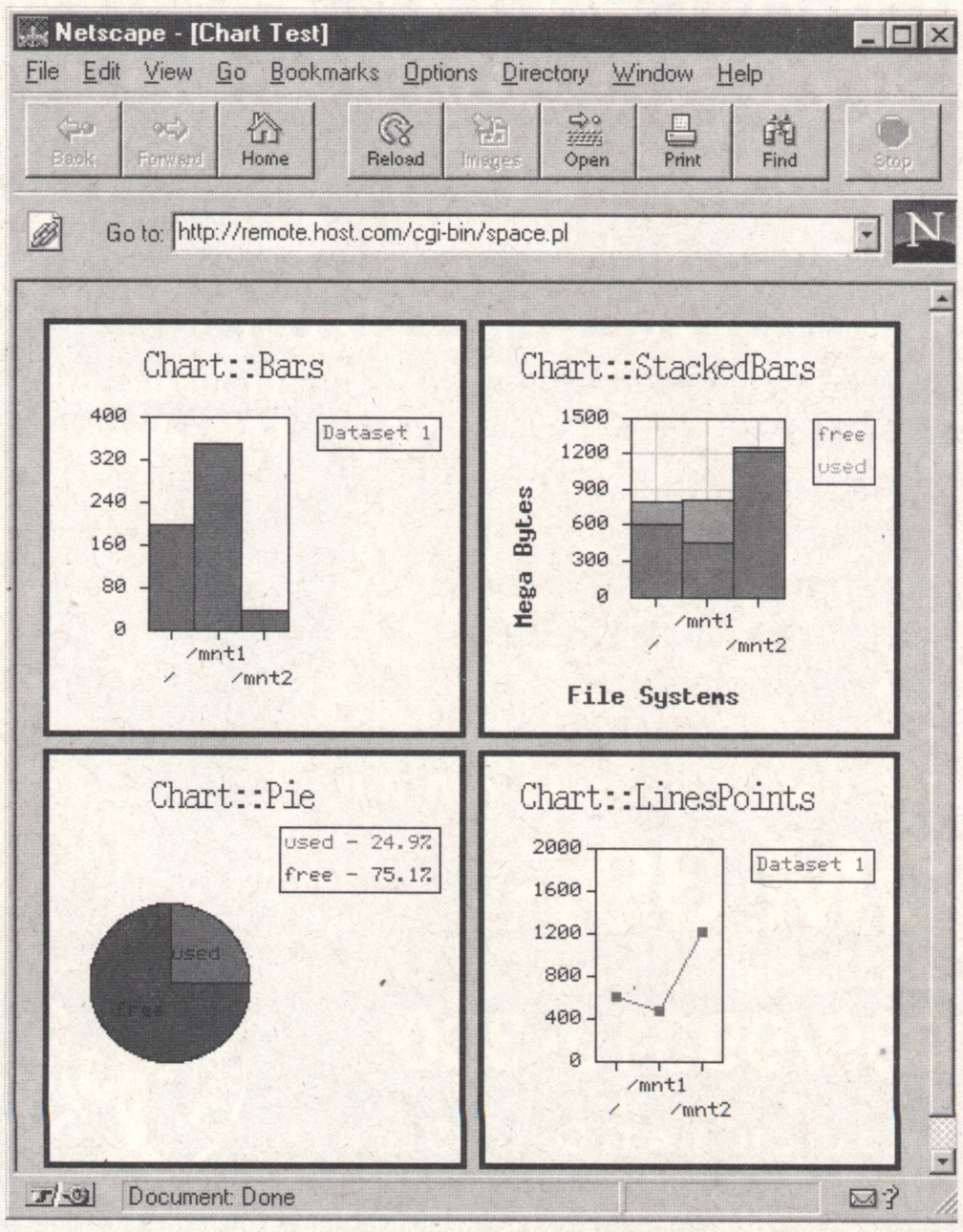

Abb.1: Plattenplatzauslastung per CGI

Abbildung 1 zeigt, was das unten vorgestellte CGI-Skript *space.pl* mit dem *Chart*-Modul (siehe Kasten Installation) daraus produziert:

So funktioniert's: Die Zeilen 5 und 6 ziehen *Lincoln Steins* praktische CGI-Library und erzeugen ein CGI-Objekt *$q*, das den CGI-Kleinkram wie die Ausgabe des Headers, das Lesen der Eingabeparameter oder auch die Ausgabe einzelner Tags vereinfacht. Der in Zeile 9 definierte (Pseudo-)-Signal-Handler läßt *space.pl*, falls es fehlerbedingt auf eine *die*-Anweisung läuft, nicht einfach abbrechen und den Web-Server ein unschönes *Internal Server Error yada yada yada* anzeigen, sondern gibt die detaillierte Fehlermeldung formatiert als HTML-Seite aus.

Wenn *space.pl* keinen CGI-Parameter mit dem Namen *graph* mitbekommt (was beim ersten Aufruf der Fall ist), verzweigt Zeile 23 zu den Anweisungen ab Zeile 25, die zunächst den HTTP-Header und die HTML-Startsequenz ausgeben, und dann, über die *for*-Schleife von Zeile 26, in etwa folgendes:

```
<IMG SRC="/cgi-bin/space.pl?graph=bars" ... >
<IMG SRC="/cgi-bin/space.pl?graph=stackedbars" ... >
<BR>
<IMG SRC="/cgi-bin/space.pl?graph=pie" ... >
<IMG SRC="/cgi-bin/space.pl?graph=linespoints" ... >
```

space.pl gibt also eine HTML-Seite aus, die <*IMG*>-Tags enthält. Diese fordern wiederum GIF-Bilder an, die dynamisch erzeugt werden - und zwar von niemand anderem als *space.pl* selbst: Die in den Tags verpackten *space.pl*-Requests übergeben - nach der GET-Methode - Werte für den Parameter *graph*.

Falls *graph* z.B. den Wert *„bars“* führt, springt *space.pl* von Zeile 35 nach 37, holt das *Chart::Bars*-Paket aus der *Chart*-Sammlung herein, malt die entsprechende Graphik, gibt sie mitsamt einem passenden HTTP-Header aus und - verabschiedet sich.

Das *df*-Kommando aus Zeile 12 liefert die eingangs vorgestellten Rohdaten für die Darstellung. Die Zeilen 14 bis 18 modeln sie in folgende Arrays um:

```
@filesystems = („/“, „/mnt1“, „/mnt1“);
@freespace   = (197.888, 348.768, 36.224);
@usedspace   = (597.769, 457.376, 1213.472);
```

Der Array *@filesystems* enthält also die (Verzeichnis-)Namen der verfügbaren Dateisysteme. In der gleichen Reihenfolge stehen in *@freespace* die Anzahl der in jedem dieser Dateisysteme verfügbaren Bytes, in *@usedspace* die der verbrauchten.

Nun geht's an die Darstellung, abhängig vom verlangten Graphiktyp.

Einfache Balken

Für eine einfache Graphik, die, wie in Abbildung 1 links oben, den verfübaren Plattenplatz pro Dateisystem als Säule ausgibt, erzeugt Zeile 38 ein *Chart::Bars*-Objekt, mit den Pixel-Ausmaßen *200 x 200*. Die nachfolgend aufgerufenen Methoden vervollständigen den Graphen nach und nach, bis ihn schließlich eine letzte als GIF-Bild ausspuckt. Zeile 39 setzt die Überschrift, die hier - zu Demonstrationszwecken - *Chart::Bars* heissen soll. Die *add_dataset*-Methode in Zeile 40 legt die X-Werte der Graphik fest: die Namen der Dateisysteme. Zeile 41 schließlich gibt die zugehörigen Y-Werte an: die Höhe der einzelnen Balken.

Ist nicht mehr angegeben, normiert *Chart::Bars* selbständig den Graphen, zeichnet die Achsen und alles was dazugehört. Der Datensatz (die Menge aller X- und Y-Werte) erhält in der 'Legende' rechts im Bild automatisch den Namen ``Dataset 1'' zugewiesen. Die *cgi_gif()*-Methode in Zeile 42 gibt das GIF samt passendem CGI-Header aus - *Ende Banane*!

Gestapelte Balken

Die *Chart::StackedBars*-Graphik in Abb. 1 rechts oben zeigt die etwas mehr aufgemotzte Darstellung zweier Datensätze, nicht, wie *Chart::Bars* es tun würde, neben-, sondern übereinander. So kommt neben den Teilbeträgen (freie und belegte Bytes) auch die Summe (Plattenkapazität) der Dateisysteme zum Vorschein. Die erste *add_dataset*-Methode fügt, wie gehabt, die X-Werte (Dateisysteme) ein, die zweite die verbrauchten Bytes und die dritte die freien.

Zur Verschönerung setzt *space.pl* in den Zeilen 48 bis 54 noch zusätzliche Parameter wie *x_label*/*y_label* (Beschriftung für die X bzw. Y-Achse), *legend_labels* (eine Referenz auf ein Array mit sinnvollen Bezeichnungen für die Datensätze statt des sonst automatisch gewählten *„Dataset N“*) und *max_val* (den in Y-Richtung maximal dargestellten Wert). Ist *grid_lines* auf *„True“* gesetzt, zeichnet *Chart* ein Gitter in den Graphenbereich, das die Zuordnung der Y-Werte vereinfacht. Die Farben für die Datensätze bestimmt *colors*, eine Referenz auf einen Array von Arrayreferenzen (hier zeigt sich, wer seine Hausaufgaben gemacht hat) mit den RGB-Werten der entsprechenden Farben. *[255,0,0]* ist hierbei schlichtes Rot, *[0,255,0]* reines Grün.

Kuchen

Die freien und die belegten Bytes auf der *„/“*-Partition ergeben (wer hätte das gedacht?) zusammen 100% - ein Anwendungsfall für eine simple Kuchengraphik mit zwei Anteilen! Zeile 64 erzeugt das notwendige *Chart::Pie*-Objekt, Zeile 66 setzt die Beschriftung der Anteile und Zeile 67 deren Werte. Der Graph links unten in Abbildung 1 zeigt die relative Auslastung der ersten Partition.

Linien mit Markierungspunkten

Für die Plattenplatz-Anzeige etwas absurd, aber für andere Anwendungsfälle ganz praktisch ist *Chart::LinesPoints*, das Linien mit Stützpunkten zeichnet. Analog hierzu arbeiten *Chart::Lines* bzw. *Chart::Points*, die entweder *nur* Linien oder *nur* Punkte malen. In jedem Fall zeichnet der erste Datensatz (Zeile 77) für die diskreten Werte auf der X-Achse und der zweite (Zeile 78) für die Höhe der Stützpunkte verantwortlich. Der Graph rechts unten in Abbildung 1 zeigt das Ergebnis.

Insgesamt ist *Chart* ein ausgesprochen praktisches Modul und *David Bonner* hat versprochen, bald 3D-Effekte einzuführen, sodaß die Anzeige noch (!) ansprechender zu werden verspricht - *cool*!

Installation

Die *Chart*-Distribution von *David Bonner* benutzt die Routinen der *GD*-Library von *Lincoln D. Stein*. Deswegen sind zwei Distributionen zu installieren: *chart-0.94.tar.gz* und *GD-1.14.tar.gz*. Am einfachsten geht das natürlich mit dem letztens vorgestellten CPAN-Saugrüssel *CPAN.pm*. Als *root* führt

```
perl -MCPAN -e shell
cpan> install LDS/GD-1.14.tar.gz
...
cpan> install DBONNER/chart-0.94.tar.gz
...
```

den gesamten Installationsprozeß automatisch durch. Manuell müssen *LDS/GD-1.14.tar.gz* und *DBONNER/chart-0.94.tar.gz* aus z. B.

ftp://ftp.gmd.de/packages/CPAN/modules/by-authors/

geholt, mit *tar zxfv* entpackt und mit *perl Makefile.PL* und *make install* (als *root*) installiert werden. Und los geht's!

LISTING SPACE.PL

```
1  #!/usr/bin/perl
2
3  $df_command = „/bin/df“;     # df-Kommando listet File-Systeme
4
5  use CGI;
6  $q = CGI->new();              # CGI-Objekt
7
8                                # ‘die’-Anweisungen enden hier
9  $SIG{__DIE__} = sub { print $q->header; print $q->h1(@_); exit 0; };
10
11 # Plattenbelegung über df-Kommando ermitteln
12 open(PIPE, „$df_command|“) || die „$df_command: command not found“;
13 while(<PIPE>) {
14     $count++ || next;         # Erste Zeile ignorieren
15     my ($drive, $total, $used, $free, $perc, $mount) = split(‘ ‘, $_);
16     push(@mounts, $mount);
17     push(@used, $used/1000);
18     push(@free, $free/1000);
19 }
20 close(PIPE) || die „$df_command failed“;
21
22 # Ohne Parameter aufgerufen - HTML-Seite ausgeben
23 if(!defined $q->param(„graph“)) {
24
25     print $q->header, $q->start_html(-title => ‘Chart Test’);
26     for (qw/bars stackedbars pie linespoints/) {
27         print $q->img({src => „$ENV{SCRIPT_NAME}?graph=$_“,
28                        border => 3,
29                        hspace => 3,
30                        vspace => 3}), „\n“;
31         print $q->br if $gcount++ % 2;
32     }
33     print $q->end_html;
34
35 } elsif($q->param(„graph“) eq „bars“) {    # Balkengraphik
36
37     use Chart::Bars;
38     my $g = Chart::Bars->new(200,200);     # Objekt erzeugen
39     $g->set(‘title’ => ‘Chart::Bars’);     # Titel setzen
40     $g->add_dataset(@mounts);              # Werte auf X-Achse
41     $g->add_dataset(@free);                # Balkenhöhe
42     $g->cgi_gif();                         # Gif ausgeben
43
44 } elsif($q->param(„graph“) eq „stackedbars“) {
45
46     use Chart::StackedBars;
47     my $g = Chart::StackedBars->new(200,200);
48     $g->set (‘title’ => ‘Chart::StackedBars’);
49     $g->set(‘x_label’ => „File Systems“);
50     $g->set(‘y_label’ => „Mega Bytes“);
51     $g->set(‘legend_labels’ => [„free“, „used“]);
52     $g->set(‘grid_lines’ => „true“);
53     $g->set(‘max_val’ => 1500);
54     $g->set(‘colors’ => [[255,0,0], [0,255,0]]);
55     $g->add_dataset(@mounts);
56     $g->add_dataset(@used);
57     $g->add_dataset(@free);
58     $g->set(‘legend_labels’ => [„free“, „used“]);
59     $g->cgi_gif();
60
61 } elsif($q->param(„graph“) eq „pie“) {     # Kuchengraphik
62
63     use Chart::Pie;
64     my $g = Chart::Pie->new(200,200);      # Objekt erzeugen
65     $g->set (‘title’ => ‘Chart::Pie’);     # Titel setzen
66     $g->add_dataset(„used“,“free“);        # Anteil-Beschreibung
67     $g->add_dataset($free[0],$used[0]);    # Werte-Satz
68     $g->cgi_gif();                         # Gif ausgeben
69
70                                            # Linien mit
71                                            # Stützpunkten
72 } elsif($q->param(„graph“) eq „linespoints“) {
73
74     use Chart::LinesPoints;
75     my $g = Chart::LinesPoints->new(200,200);
76     $g->set (‘title’ => ‘Chart::LinesPoints’);
77     $g->add_dataset(@mounts);              # X-Achsen-Werte
78     $g->add_dataset(@used);                # Höhe
79     $g->cgi_gif();                         # Gif ausgeben
80
81                                            # Gestaffelte Balken
82 }
```

DER AUTOR

Michael Schilli arbeitet als Web-Engineer bei AOL Productions, San Mateo. Er ist Autor des bei Addison-Wesley erschienenen Buches *„Effektives Programmieren mit Perl 5“* und unter *mschilli1@aol.com* oder *http://mission.base.com/mschilli* zu erreichen.