Das einzige deutschsprachige Linux-Magazin

LINUX MAGAZIN 2/98

LINUX MAGAZIN

ISSN 1432 - 640 X DM 9,80 ÖS 81,- SFR 9,80

2/98

Linux News
Neues aus dem Internet

Testbericht
Red Hat 5.0/Intel

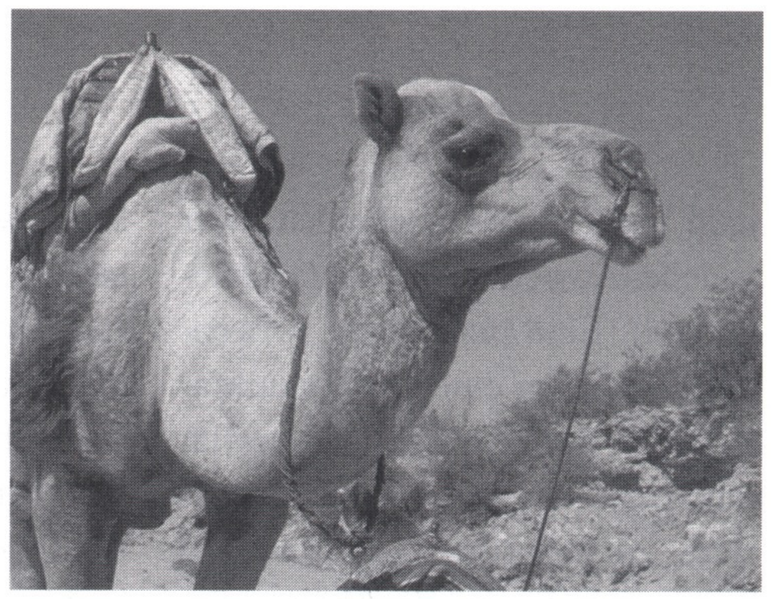
TV-Karten
Fernsehen mit BTTV

Tips und Tricks
Plattencrash – was nun?

Gast-freundlichkeit

WWW-Server mit Slip-Zugang

www.linux-magazin.de



Perl Snapshot

Wie von Geisterhand

von Michael Schilli

Erinnert sich noch jemand an *expect*? Zeichenketten zu senden, um auf Zeichenketten zu warten, um wiederum ... ideal, um über *telnet* Routinearbeiten auf einem Dutzend Rechner automatisch durchzuführen.

Expect basierte auf *Tcl*, und, wie soll ich es sagen, *Tcl* und ich, wir kamen uns nicht näher, etwas stand immer zwischen uns: Mal war's eine geschweifte Klammer in der falschen Zeile, mal ein *eval* zuviel oder zuwenig - es wollte nicht klappen mit uns zwei'n. Wie froh war ich daher, zu sehen, daß es das Perl-Modul *Net::Telnet* gibt und alles viel einfacher geht!

LISTING 1: CHECKLOAD.PL

```
1 #!/usr/bin/perl -w
2
3 use CGI qw/:standard :html/;
4 use Net::Telnet;
5
6     # Konfiguration
7 $userid = „watch“;
8 $passwd = „topsecret!“;
9
10     # Server-Namen und deren Prompts
11 @hosts = (['machine1.domain.com', 'machine1>'],
12            ['machine2.domain.com', '$'],
13            ['machine3.domain.com', '>'],
14            ['murkshost', '$'],
15           );
16 $logfilename = „/log/error_log“;
17
18     # Alle Rechner abklappern
19 for (@hosts) {
20     my ($host, $prompt) = @$_;
21
22     $prompt = sprintf('^%s\s*$', quotemeta($prompt));
23
24     $telnet = new Net::Telnet (Host     => $host,
25                                Timeout => 60,
26                                Prompt  => „/$prompt/m“);
27
```

Expect mit Perl

Heute stelle ich ein Skript vor, das sich der Reihe nach auf jedem Rechner eines Maschinenparks einloggt, dort die Auslastung und die Größe einer Log-Datei abfragt und die gesammelten Daten schließlich schön formatiert anzeigt. Und, zwecks Komfortsteigerung packen wir das Ganze in ein CGI-Skript, das einen handelsüblichen Web-Server dazu bewegt, auf Knopfdruck den Zustand unserer Rechner anzuzeigen. Abbildung 1 zeigt das Ergebnis des CGI-Skripts *checkload.pl* (Listing 1) im Browser.

Zeile 3 bindet *Lincoln Steins* praktisches *CGI*-Modul ein, mit den Tags *:standard* und *:html* exportiert es die weiter unten benötigten Funktionen ohne überflüssige Objekt-Huberei. *Cool*!

Zeile 4 zieht das Modul *Net::Telnet*, dessen Distribution auf dem CPAN unter

```
CPAN/modules/by-module/Net/Net-Telnet-
3.00.tar.gz
```

liegt und sich wie alle Perl-Module am schnellsten mit dem im Oktober vorgestellten *CPAN*-Saugrüssel installieren läßt. Die Benutzerkennung und das Paßwort aus den Zeilen 7 und 8 gewähren Zugang zu allen nachfolgend abgefragten Rechnern.

Jeder Eintrag der *@hosts*-Liste ab Zeile 11 enthält eine Referenz auf eine Liste, die als erstes Element den Host-Namen und als zweites den *Prompt* enthält, jene Zeichenkette, die der jeweilige Rechner zur Eingabeaufforderung anzeigt. So steht bei *machine2.domain.com* das für die Bourne-Shell typische *$*, *machine1.domain.com* bevorzugt hingegen *machine1>* - jedem das Seine.

Zeile 19 iteriert über diese *LoL* (*List of Lists*). Für jeden Schleifendurchgang liegt eine Referenz auf die aktuelle Unterliste in *$_*. Zeile 20 dereferenziert dies mittels *@$_* und kopiert die Werte für den Rechnernamen und den Prompt in die augenfreundlicheren Variablen *$host* und *$prompt*.
Damit die nachfolgende Telnet-Session auch wirklich nur auf den Prompt reagiert und nicht etwa auf eine aufgelistete Datei gleichen Namens, formt Zeile 21 aus der angegebenen Zeichenkette einen regulären Ausdruck, der festlegt, daß der Chatter auf eine Zeile als Antwort wartet, die mit dem Prompt beginnt und dann eventuell mit einer Reihe von Leerzeichen endet. Die *quotemeta*-Funktion aus der Perl-Standard-Bibliothek übernimmt dabei die Maskierung gefährlicher Zeichen wie *, denen in regulären Ausdrücken eine Sonderbedeutung zukommt.
Zeile 29 versucht dann, mittels der *login*-Methode des vorher erzeugten *Telnet*-Objekts Kontakt mit dem aktuellen Rechner aufzunehmen und den Login-Prozeß durchzuführen. Da *login* leider innerhalb des *Telnet*-Moduls mit einer *die*-Anweisung abbricht, falls der fremde Rechner sie zurückweist oder nicht erreichbar ist, fängt Zeile 29 diesen Fehlerfall mit Hilfe eines *eval*-Konstruktes ab. Ging innerhalb des *eval*-Blockes etwas schief, ist *$@* gesetzt, was Zeile 31 abprüft und gegebenenfalls die Ergebnis-Liste *@hostinfo* um einen Fehlereintrag bereichert.
Andernfalls führt Zeile 35 ein *uptime*-Kommando auf dem fremden Rechner aus. Die *cmd*-Methode aus dem Telnet-Modul setzt das angegebene Kommando ab, wartet auf den Prompt und liefert die empfangenen Zeilen als Liste zurück. Da *uptime* nur eine Zeile zurückliefert, enthält *$lines[0]* einen String a la

```
11:31am  up  2:28,  3 users,  load average:
0.07, 0.11, 0.15
```

Der reguläre Ausdruck aus Zeile 36 filtert daraus *0.07*, den Wert der Rechnerlast, gemittelt über die Zeitspanne einer Minute. Entsprechend startet Zeile 39 ein *ls*-Kommando, welches folgende Informationen über die Logdatei zurückliefert:

```
-rw-r--r-- 1 nobody nogroup 43896 Nov 22 13:39
/log/error_log
```

Zeile 40 trennt die ausgebenen Felder an den Zwischenräumen und filtert daraus das fünfte Element, die Größe der Datei, *43896* Bytes. In Zeile 41 kommt der bewährte Trick zum Einsatz, der große Zahlen durch Punkte in Dreier-Gruppen aufspaltet (Perl FAQ).

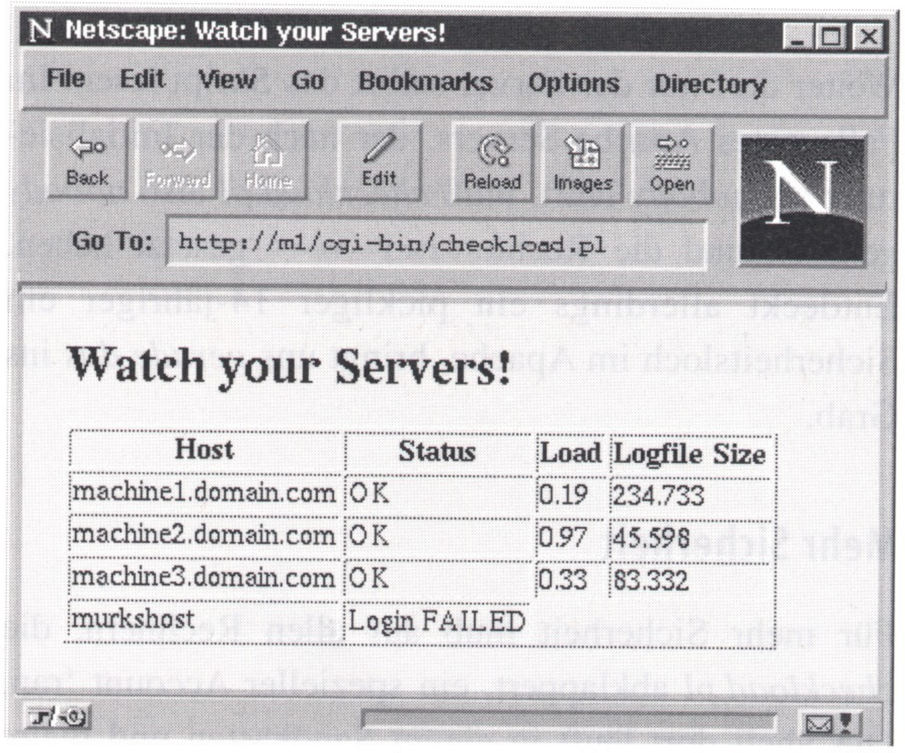


Abb.1: Alles läuft, einer schläft ...

Zeile 43 schiebt eine Referenz der Liste gewonnener Informationen ans Ende des Behälters (LoL) *@hostinfo*.
Statt *print* verwendet *checkload.pl*, wie z.B. in Zeile 50, *cgiprint*. Diese ab Zeile 70 definierte Funktion unterhält eine statische Variable, die ihr anzeigt, ob der vor allen anderen Ausgaben notwendige CGI-Header schon ausgegeben wurde. So muß sich niemand mehr um den lästigen Header kümmern. Egal welchen Weg das Programm im Fehler (oder guten) Fall nimmt - *cgiprint* wird's schon richten.
Zeile 50 gibt die HTML-Startsequenz aus und setzt dabei den Titel des angezeigten Dokuments. Nach der Ausgabe der Überschrift in Zeile 54 beginnt Zeile 56, den Inhalt einer HTML-Tabelle im String *$tablecontent* aufzubauen. Die *foreach*-Schleife zwischen den Zeilen 60 und 62 transformiert die LoL *@hostinfo* in eine HTML-Tabelle - für jeden Schleifendurchlauf erzeugt *TR()* eine neue Tabellenzeile, die wiederum mittels der *map*-Funktion in *<TD>*-Tags gepreßte Unterlisten-Elemente als Spalten enthält. *Uff*! Ohne Abitur geht's halt nicht.
Zeile 65 gibt das Monstrum schließlich aus, Zeile 66 schließt alle HTML-Ausgaben mit *</HTML>* ab.

Sicherheit

Noch ein ernstes Wort zum Thema Sicherheit: Natürlich kann ein Skript, das eine Benutzerkennung für mehrere Rechner samt gültigem Paßwort im Klartext enthält, ein gewaltiges Loch in ein sonst gut abgesichertes System reißen. Ohne weitere Sicherungsmaßnahmen sollte man das Skript deswegen nur innerhalb einer Firewall betreiben (Browser und Webserver!), die rigoros alle von außen kommenden *telnet*-Anfragen abblockt.

Weiter darf nur der Server selbst das Skript lesen. Im Falle eines Apache-Servers, der nach der Initialisierung als *nobody* laüft, muß *checkload.pl* also *nobody* gehören und die Rechte *-rwx------* gesetzt haben. Entdeckt allerdings ein pickliger 14-jähriger ein Sicherheitsloch im Apache, bringt uns *gerade das* ins Grab.

Mehr Sicherheit

Für mehr Sicherheit muß auf allen Rechnern, die *checkload.pl* abklappert, ein spezieller Account 'ran. Die Shell dort läuft in einem Sandkasten und bietet nur *ls* und *uptime* als Funktionalität. Keinen *vi*, kein *cat*, kein gar nichts! Das bringt zusätzlichen Schutz vor ungebetenen Gästen, ist aber ein wenig aufwendig:
So wie man beim anonymen FTP-Zugriff nicht in den Top-Verzeichnissen (z.B. */etc*) eines Servers herumwühlen darf, beschränkt ein speziell eingerichteter *watch*-Account den Zugriffsbereich des Benutzers mit *chroot* auf einen kleinen Unterbereich des Dateisystems. *chroot* legt ein Verzeichnis als neue Wurzel des Dateisystems fest und unterbindet rigoros Zugriffe oberhalb dieses *Käfigs*.
Hierzu richtet man in */etc/passwd* einen neuen Benutzer ein:

```
watch::9999:9999::/home/watch:/home/watch/bin/
cage
```

und erzeugt mit

```
mkdir /home/watch
```

dessen neues Verzeichnis. Das neue Paßwort wird als *root* mit *passwd watch* gesetzt. Die Käfig-Shell */home/watch/bin/cage* stricken wir mit dem folgenden Programm

```
#include <stdio.h>

main() {

    char *sandboxdir = „/home/watch";
    putenv(„LD_LIBRARY_PATH=/lib");
    putenv(„PATH=.:/bin");

    if(chdir(sandboxdir)) {
        perror(„Chdir failed");
        exit(1);
    }
    if(chroot(sandboxdir)) {
    perror(„Chroot failed");
    exit(1);
}

setuid(9999);

execl(„/bin/bash", „-cs", NULL);
    perror(„Shell did not start");
}
```

selber und kompilieren das Ergebnis nach */home/watch/bin/cage*. Das Executable *cage* muß dafür *root* gehören und das setuid-Bit (mit *chmod 4755 cage*) gesetzt haben. Die aus dem C-Programm aufgerufene *bash*-Shell liegt aber normalerweise in */bin*, ganz zu schweigen von den *shared libraries*, die sie benötigt - alles völlig außer Reichweite, sind wir einmal in */home/watch* gefangen, ohne Ausweg nach oben! Da hilft nur Kopieren: Die Shell, *ls* und *uptime*, den Linux-Loader und die Bibliotheken:

```
cd /home/watch
mkdir lib bin usr usr/lib dev etc log
cp /bin/ls bin
cp /usr/bin/uptime bin
cp /lib/ld-linux.so.1 lib
cp /etc/ld.so.cache etc
cp /lib/libtermcap.so.2 lib
cp /lib/libc.so.5 lib
cp /bin/bash bin
```

(Andere Linux-Versionen als 2.0.X erfordern unter Umständen andere Dateien, einfach *cage* als *root* aufrufen und eventuell ausgegebene Fehlermeldungen prüfen).
Und: *Oh Jammer oh Not*! Auch das zu untersuchende Logfile liegt außerhalb des Sicherheitsbereichs. Diese Schranke überwindet der *harte* Link:

```
ln /log/error_log log/error_log
```

im Verzeichnis */home/watch*. Damit greift man auch aus dem Käfig über */log/error_log* auf die Log-Datei zu. Ein symbolischer Link genügt übrigens nicht, der darf aus Sicherheitsgründen nicht über die gesetzte Grenze hinwegschauen.
Das *uptime*-Kommando benötigt weiter einen *mount* auf */proc*, was manuell mit

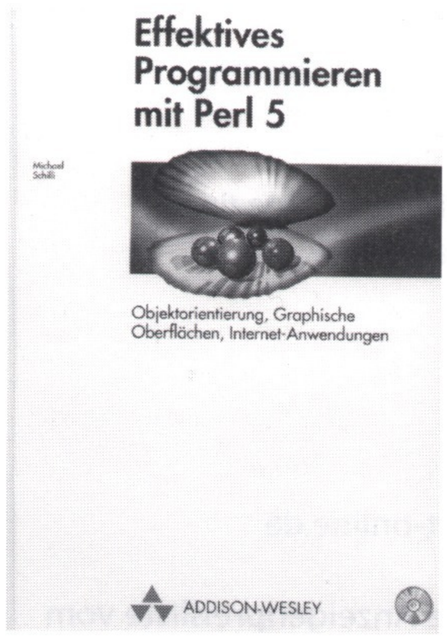
```
mount /proc /home/watch/proc -t proc
```

geht, für automatisches Mounten beim Startup sorgt folgender Eintrag in */etc/fstab*:

```
/proc    /home/watch/proc    proc    defaults
```

Außerdem nutzt *uptime* die Datei */var/run/utmp*, also spendieren wir auch dafür einen Link. In */home/watch* geht das mit

```
mkdir var var/run
ln /var/run/utmp var/run/utmp
```



Effektives Programmieren mit Perl 5

Was noch bleibt ...

Wer noch etwas Zeit und Muße hat: Das letzten Monat vorstellte *Chart*-Paket eignet sich hervorragend zur optisch ansprechenden Aufbereitung der Daten.
Sonst: Ab mit dem Skript ins *cgi-bin*-Verzeichnis des Webservers, den URL *http://my.server.com/cgi-bin/checkload.pl* in die Bookmarks/Favourites-Liste des Browsers aufgenommen und und bei Arbeitsbeginn einmal draufgeklickt. Alle Server laufen wie die Nähmaschinen? Na, da schmeckt der Morgenkaffee doch gleich viel besser. *Mmmmhh*. Scheint ein erfolgreicher Tag zu werden ...

DER AUTOR

Michael Schilli arbeitet als Web-Engineer für AOL Productions, San Mateo. Er ist Autor von „Effektives Programmieren mit Perl 5" und unter mschilli1@aol.com zu erreichen.
Seine Homepage: http://mission.base.com/mschilli.

▼ LISTING 1: CHECKLOAD.PL

```
28          # Einloggen, 'die' abfangen
29      eval { $telnet->login($userid, $passwd); };
30
31      if($@) {      # Fehler aufgetreten?
32          push(@hostinfo, [$host, „Login FAILED"]);
33      } else {
34                  # Last abfragen
35          @lines = $telnet->cmd(„uptime");
36          ($load) = ($lines[0] =~ /average:\s*([0-9.]+)/);
37
38                  # Logfile-Länge abfragen
39          @lines = $telnet->cmd(„ls -l $logfilename");
40          $logsize = (split(' ', $lines[0]))[4];
41          1 while ($logsize =~ s/^(\d+)(\d\d\d)/$1.$2/);
42
43          push(@hostinfo, [$host, „OK", $load, $logsize]);
44      }
45
46      $telnet->close;
47  }
48
49      # HTML ausgeben
50  cgiprint(start_html(-BGCOLOR => „bisque",
51                      „title"  => „Watch your Servers!"));
52
53      # Überschrift
54  cgiprint(h1(„Watch your Servers!"), „\n");
55
56  $tablecontent = TR( th(„Host"), th(„Status"),
57                      th(„Load"), th(„Logfile Size"), „\n" );
58
59      # @hostinfo LoL -> HTML Tabelle
60  foreach $lref (@hostinfo) {
61      $tablecontent .= TR( map { td($_) } @$lref ) . „\n";
62  }
63
64      # Tabelle ausgeben
65  cgiprint(table({border => 1}, $tablecontent));
66  cgiprint(end_html());
67
68
69  ############################################################
70  sub cgiprint {
71  ############################################################
72  # Text ausgeben, Header falls notwendig
73  ############################################################
74      print header() unless defined $header_printed;
75      print „@_";
76      $header_printed = 1;
77  }
```

Autorenliste

Marco Budde

Kalle Dalheimer

Hartmut Eilers

Robert Gehring

Wolfgang Hetzler

ThomaS Kniep

Bernhard Kuhn

Michael Lupp

Ralph Metzler

Thomas Riedl

Michael Schilli

Martin Schulze

Tom Schwaller

Norbert Walter

Service für DeLUG-Mitglieder

Aktuelle Software

Krypto-Info:

DFN-PCA Schlüssel: 2048/35DBF565 1997/04/16 DFN-PCA,
CERTIFICATION ONLY KEY (Low-Level: 1997-1998) <not-for-mail>
Key fingerprint = 09 7C 09 19 D3 C3 86 DC 7A 30 15 11 12 95 8D E3

IN-Root-CA Schlüssel: 2048/d8759c65
Key fingerprint = 1B E1 4F F7 EC DE 3B 09 6E 5A 9F 6F 13 CB AF C7
Hashwert des Ewigen Logfiles vom 02.01.1998 13:22:14
sha1=198e1e7519bd5c81d65b38150be9dab7a594136a
https://www.iks-jena.de/mitarb/lutz/logfile/

Inserentenverzeichnis

IMPRESSUM

Informationen, Berichte und Neuigkeiten aus der Linux-Welt

Herausgeber: Rudolf Strobl
Redaktion:
Email: redaktion@linux-magazin.de
Post: **Linux - Magazin Verlag**
Stefan-George-Ring 19
81929 München
Tel.: 089/99315-310
Fax: 089/99315-199
WWW: http://www.linux-magazin.de
ISSN: 1432 - 640 X
Chefredakteur: Tom Schwaller,
tom.schwaller@linux-magazin.de (ts)
Coverdesign: Tilo Wondollek, TW-Design
Layout : C.&D. Elsäßer
Mitarbeiter: siehe Autorenliste
Vertriebsmarketing: info@linux-magazin.de
Anzeigen: Agentur Neumann
Kalvarienbergstraße 31
Postfach 1352
87503 Immenstadt
Tel: 08323 / 98100
Fax: 08323 / 963226
Email: ThomasNeumann@t-online.de

Anzeigenpreise: Es gilt die Anzeigenpreisliste vom 01.12.1997

Das Linux - Magazin erscheint monatlich.
Einzelheft: **DM 9,80**
Preis des Jahresabonnements:
Incl. DeLUG-Mitgliedschaft **DM 180,-**
Ohne Mitgliedschaft **DM 103,-**
(Ausland: 115,-)

Für Schüler und Studenten 20% Ermäßigung (Vorlage Schüler- oder Studentenausweis)