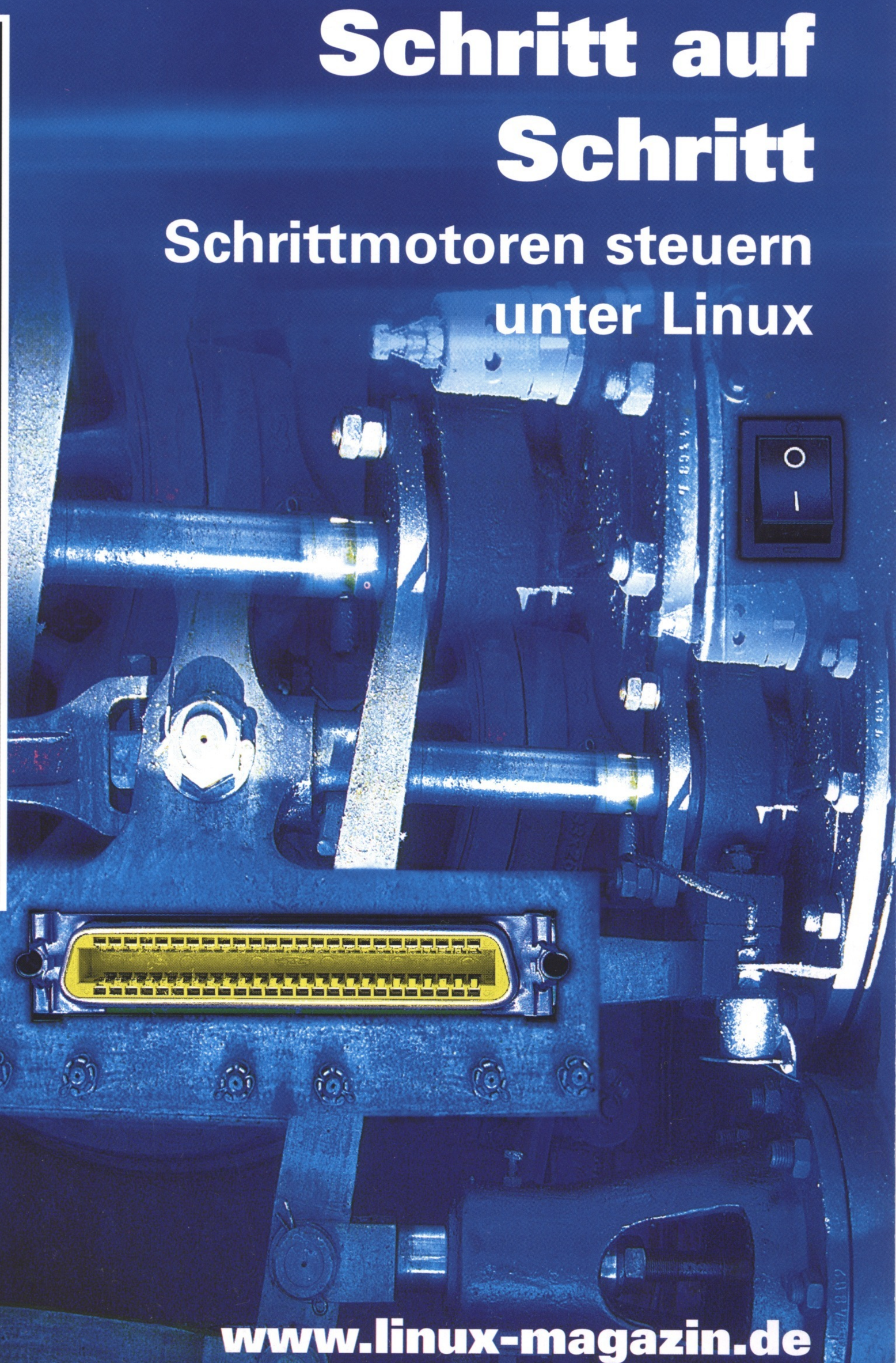
Das einzige deutschsprachige Linux-Magazin

LINUX MAGAZIN 3/98

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

Schritt auf Schritt

Schrittmotoren steuern unter Linux



Linux News

Neues aus dem Internet

Bericht

Linux goes Titanic

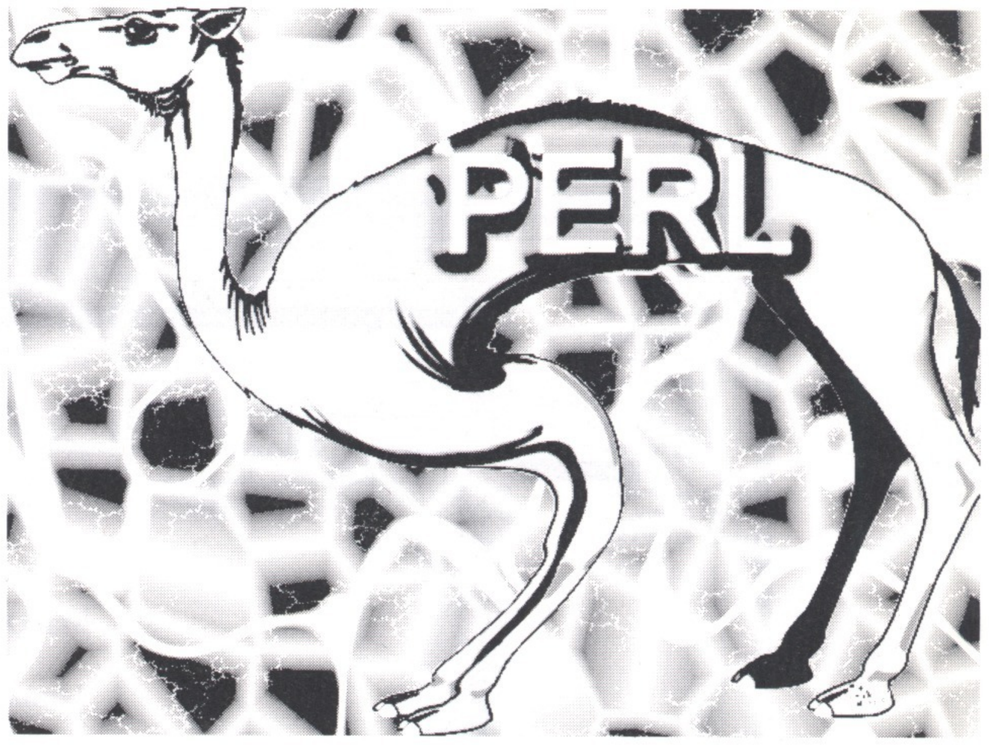
Echtzeiterweiterung

RT-Linux

Für Einsteiger

Debian m68k-Linux

www.linux-magazin.de



Perl Snapshot

Aus der CGI-Trickkiste

von Michael Schilli

Wer CGI-Skripts mit „use CGI" anfängt, hat es leichter, denn CGI.pm, Lincoln Steins praktisches Modul, holt Query-Parameter ab, verkürzt HTML-Ausgaben, hilft beim Debuggen - ein Tausendsassa in Modulform!

CGI.pm liegt der aktuellen Perl-Distribution *perl-5.004_04* bei - ein weiterer Grund, aufzurüsten! Sonst liegt es auf dem CPAN unter

```
CPAN/modules/by-module/CGI/CGI.pm-2.37b12.tar.gz
```

CGI.pm arbeitet eigentlich mit einem CGI-*Objekt*:

```
use CGI;
$q = new CGI;
```

Über die Objektreferenz *$q* stehen anschließend die Dienste als Methoden zur Verfügung. So gibt

```
print $q->header();
print $q->h1(„Überschrift");
```

den Server-Header und eine *H1*-Überschrift etwa als

```
Content-type: text/html

<H1>Überschrift</H1>
```

aus. Im harten Alltagsgeschäft verkompliziert das Objekt *$q* aber nur unnötig die HTML-Ausgaben. Mit einer Tag-Liste hinter dem *use*-Befehl läßt *CGI.pm* fünf gerade sein und importiert die Methoden als Funktionen in den Namensraum des Skripts:

```
use CGI qw/:standard/;
print header, h1(„Überschrift");
```

Für CGI-Parameter, Formulare und Basis-HTML reicht *:standard*, kommen Tabellen mit ins Spiel, muß auch noch *:html3* 'ran. Doch noch einmal zurück zum Anfang: Richtiges HTML mit Titel, Body und allem, was dazugehört, schafft

```
print start_html('-title'    => 'Titel',
                 '-BGCOLOR' => 'white'),
      h1(„Überschrift"),
      end_html;
```

Das setzt den Titel des zurückgelieferten HTML-Dokuments auf *Titel* und die Hintergrundfarbe auf ein neutrales Weiß. Zwar kommt ab Perl 5.003 der vordere Teil einer *Key/Value*-Kombination wie z.B. *-title => 'abc'* auch ohne sichernde Anführungszeichen aus, da aber *CGI.pm* eine Unmenge von Funktionen exportiert, kann es zu Verwirrungen kommen: *-title* könnte der Perl-Interpreter auch als negativen Rückgabewert der von *CGI.pm* exportierten Funktion *title()* verstehen. Bei angeschalteter *-w*-Option mault *perl* entsprechend; *'-title'* schafft hier Klarheit und Ruhe.
So gibt's für jeden HTML-Tag eine Funktion aus *CGI.pm*. Listing *basehtml.pl* zeigt ein CGI-Skript, das die wichtigsten Tags aus dem HTML-Standard-Fundus verwendet: Listen, verschiedene Schriftarten, Hyperlinks.

Ins *cgi-bin*-Verzeichnis des Webservers verfrachtet, bringt ein Aufruf von *http://server/cgi-bin/basehtml.pl* im Browser die Ausgabe aus Abbildung 1 zum Vorschein.

Die Funktionen aus CGI.pm, die Texte HTML-formatiert zurückgeben, heißen wie die zugehörigen HTML-Tags: Aus <P> wird p(), aus <UL> (Bullet-Liste) wird ul(), mit dem Unterschied, daß HTML einen Ausdruck mit einem öffnenden und einem schließenden Tag umrahmt, während <CGI.pm> ver-

Bullet-Liste

- *kursiv*
- **fett**
- getippt

Glossar-Liste

Hyperlink
Hier klicken!
Hyperlink als Image
Image!

Abb.1: HTML-Tags aus dem Standard-Fundus

schachtelte Funktionenaufrufe erfordert, um die gewünschte Struktur zu erzeugen. So liefert dl(ul('a'), ul('b')) etwa

```
<DL><UL>a</UL><UL>b</UL></DL>
```

Um Attribute für ein HTML-Tag zu setzen, also z. B. mit dem *SRC*-Attribut die Quelldatei für ein Image-Tag festzulegen, erhält die entsprechende *CGI.pm*-Funktion einfach einen anonymen Hash als ersten Parameter, der Wertepaare der Form Attributname/Wert enthält, für einen Hyperlink also beispielsweise

```
a({href => 'http://anywhere.com'},
   „Hier klicken!");
```

was

```
<A HREF=http://anywhere.com>Hier klicken!</A>
```

zurückliefert.

Tabellen

Tabellen enstehen mit den Funktionen *table*, *TR*, *th* und *td*, die den HTML-Tags *TABLE*, *TR*, *TH* und *TD* entprechen (daß die *TR*-Funktion großgeschrieben wird, liegt nur daran, daß es schon eine Perl-Funktion namens *tr* gibt).

```
use CGI qw/:standard :html3/;

print table(
      {-border => 1, -bgcolor => 'beige'},   „\n",
      TR( th(„Spalte1"), th(„Spalte2") ),    „\n",
      TR( td(„Feld 1/1"), td(„Feld 1/2") ), „\n",
      TR( td(„Feld 2/1"), td(„Feld 2/2") ), „\n",
 );
```

Die Newlines wären zwar aus HTML-Sicht nicht notwendig, erleichtern aber das Lesen der Ausgabe:

```
<TABLE BORDER="1" BGCOLOR="beige">
<TR><TH>Spalte1</TH> <TH>Spalte2</TH></TR>
<TR><TD>Feld 1/1</TD> <TD>Feld 1/2</TD></TR>
<TR><TD>Feld 2/1</TD> <TD>Feld 2/2</TD></TR>
</TABLE>
```

Stellt Perl-Code den Tabelleninhalt dynamisch zusammen, ergibt sich öfter das Problem, daß man eigentlich alle Tabellendaten innerhalb eines

LISTING 1: basehtml.pl

```
1  #!/usr/bin/perl -w
2
3  use CGI qw/:standard/;
4
5  print header,
6        start_html('-title'    => 'HTML-Tags',
7                   '-BGCOLOR' => 'white'),
8
9        h2(„Bullet-Liste"),
10       ul(
11          li( i(„kursiv") ),
12          li( b(„fett") ),
13          li( tt(„getippt") )
14        ),
15
16       hr,    # Querstrich
17       p,     # Absatz
18
19       h2(„Glossar-Liste"),
20       dl(
21          dt(„Hyperlink"),
22          dd(
23             a( {href => 'http://www.com'}, „Hier klicken!")
24          ),
25          dt(„Hyperlink als Image"),
26          dd(
27             a( {href => 'http://www.com'},
28                img({src => „/pic.gif"}))
29          )
30       ),
31
32       end_html;
```

table(...)-Aufrufs sammeln möchte, aber es geht nicht, weil deren Erzeugung so kompliziert ist! Teilen wir die Arbeit doch auf:

```
use CGI qw/:standard :html3/;

# Tabellen-Inhalt erzeugen

foreach $row (1..2) {
   $rowcontent = „";
   foreach $col (1..2) {
      $rowcontent .= td(„Feld $row/$col");
   }
   $tablecontent .= TR($rowcontent) . „\n";
}
```

```
# Tabellen ausgeben

print table(
      {-border => 1, -bgcolor => 'orange'}, „\n",
      TR( th(„Spalte1"), th(„Spalte2") ),    „\n",
       $tablecontent
 );
```

Perls *map*-Funktion bewältigt schleifen-typische Aufgaben auch ohne Schleife. Das nachfolgende Snippet macht aus einer Liste von Spaltenüberschriften (*@head*) und einer LoL (List of Lists, enthält die Tabellenzeilen als Unter-Listen) ohne viel Federlesens eine Tabelle:

```
use CGI qw/:standard :html3/;

@head = („1. Spalte", „2. Spalte",
         „3. Spalte");
@lol  = ([1,2,3], [4,5,6], [7,8,9]);

print table(
   TR(map { th($_) } @head), „\n",
   map { TR(map { td($_) } @$_) . „\n" } @lol
);
```

Die erste Zeile im *table()*-Aufruf umschließt jeden Eintrag in *@head* mit *<TH>...</TH>* und rahmt das Ganze mit *<TR>...</TR>* ein - fertig ist der Tabellenkopf. Der *map*-Befehl liefert ja bekanntlich eine neue Liste zurück, indem er jedes Element der ihm überreichten Liste durch den in geschweiften Klammern entstehenden Ausdruck ersetzt.
Bei der Zeile mit den zwei *map*-Befehlen, die direkt darunter steht, wäre mir fast der Hirnkasten zersprungen, so angestrengt mußte ich nachdenken. Der Trick ist der: Der äußere *map*-Befehl ackert *@lol* durch, läßt die gefundenen Unter-Listen (*@$_*) vom inneren *map*-Befehl verarbeiten, schreibt sein *<TR>...</TR>* drumherum und gibt's zurück. Der innere *map*-Befehl rahmt jeden Unter-Listen-Eintrag mit *<TD>...</TD>* ein. Alles klar?

Weitere Tags

CGI.pm beherrscht alle HTML-Tags, auch wenn sie nicht unbedingt in der mitgelieferten Dokumentation (erscheint mittels *perldoc CGI*) stehen. Der HTML-Tag in Kleinbuchstaben ergibt meist die entsprechende Funktion aus *CGI.pm*. Attribute (wie z.B. das *SRC*-Attribut im *IMG*-Tag) setzt ein anonymer Hash als erstes Argument. Da diese Funktionen meist nicht mit dem *:standard*-Tag exportiert werden, hilft ein vorangestellter Modul-Bestimmer wie *CGI::*. So ändert

```
print CGI::font({size => '+1', color =>
'red'}, „Achtung, Rot!");
```

die Fontgröße und -farbe in HTML mit

```
<FONT SIZE="+1" COLOR="red">Achtung,
Rot!</FONT>

Fehler
```

Tritt bei einem CGI-Skript ein schwerwiegender Fehler auf, so schwer, daß man am liebsten alles hinwerfen würde und das Skript abbrechen, stellt sich das Problem, daß vor der eigentlichen Fehlermeldung ein CGI-Header kommen muß, sonst zeigt der Browser ein unschönes *Internal Server Error* an und das heißt: *Amateur am Werk!*
Steht die Header-Ausgabe am Anfang und der kritische Teil des Skripts inmitten eines *eval*-Konstrukts, kann nichts schiefgehen: Läuft das Skript auf eine *die*-Anweisung, springt es aus dem *eval*-Block heraus und *hinein!* in die nachfolgende *if*-Bedingung, denn *$@* führt in diesem Fall den Wortlaut der Fehlermeldung:

```
use CGI qw/:standard/;

print header();

eval {
   #...
   print p(„Noch geht's gut ...\n");
   #...
   die „Schwerer Fehler!";
};

if ($@) {
   print h1(„Fehler: $@");
}
```

Um den *eval*-Block zu sparen, kann man auch die Pseudo-Signale *__DIE__* und *__WARN__* abfangen, und einen Handler für den Fehlerfall definieren:

```
use CGI qw/:standard/;

$SIG{__DIE__} = $SIG{__WARN__} =
   sub { cgiprint(h1(„@_")); exit 0 };

#...
```

```
cgiprint(p(„Noch geht's gut ..."));
#...
die „Schwerer Fehler!";

sub cgiprint {
##############################################
####
# Text ausgeben, Header falls notwendig
##############################################
####
    print header()
          unless defined $header_printed;
    print „@_";
    $header_printed = 1;
}
```

Alle Ausgaben schleust dieser Code durch *cgiprint*, einer *print*-Funktion, die lediglich beim ersten Aufruf der Ausgabe den Header voranstellt. *cgiprint* 'merkt' sich diesen Zustand in der globalen Variablen *header_printed*.

Debugging

Bevor ein CGI-Skript im *cgi-bin*-Verzeichnis irgendwelchen Unsinn treibt, sollte es zumindest einmal fehlerfrei auf dem Trockenen laufen: Ein Perl-Skript mit eingebundenem *CGI.pm* 'weiß', ob es von der Kommandozeile oder vom Server aufgerufen wurde. Falls es im ersten Fall irgendwo CGI-Eingabe-Parameter verarbeitet, stoppt es vorher und nimmt Wertepaare der Form *Key=Value* entgegen:

```
(offline mode:
enter name=value pairs on standard input)
a=b
c=d
CTRL-D
```

Anschließend fährt es normal mit der Bearbeitung fort, nur daß die Query-Parameter *a* und *c* die Werte *b* bzw. *d* enthalten.
Um in *cgi-bin* schnell mal auszuprobieren, welche Parameter ein Skript zugespielt bekommt, eignet sich das Skript

```
use CGI qw/:standard/;

print header,
 start_html,
 h2(„Parameter:"), CGI::as_string(),
 h2(„Environment:"),
 (map { p(„$_ => $ENV{$_}") } sort keys %ENV),
end_html;
```

Die *as_string()*-Funktion gibt alle empfangenen Query-Parameter als Glossar-Liste in HTML aus, egal ob sie per GET- oder POST-Methode angelangten. Kritische Zeichen, die der Browser zur eindeutigen Übermittlung im Format *%xx* maskierte, entpackt *CGI.pm* dabei wieder vorschriftsmäßig.
Die Zeile mit dem *map*-Befehl schreibt noch sämtliche Umgebungsvariablen aus dem *ENV*-Hash schön formatiert in die Ausgabe. Der Server nutzt die Environment-Variablen als Schnittstelle, um dem Skript bestimmte Variablen zu übermitteln (z.B. den Original-Querystring, die IP-Adresse des Absenders, eventuell vorhandene Request-Header etc.).
Ins *cgi-bin*-Verzeichnis des Web-Servers gestellt und mit

```
http://localhost/cgi-bin/dump.pl?a=b&c=d
```

aufgerufen, zeigt *dump.pl* im Browser etwa ein Bild nach Abbildung 2.

Wie gelangen die Eingabedaten normalerweise zum CGI-Skript? *Formulare, Formulare, Formulare* heißt die Antwort. Wie man sie (natürlich mit *CGI.pm*) erstellt und Benutzer-Eingaben auswertet (natürlich auch mit *CGI.pm*), zeige ich in der nächste Folge. Habe die Ehre, Freunde!

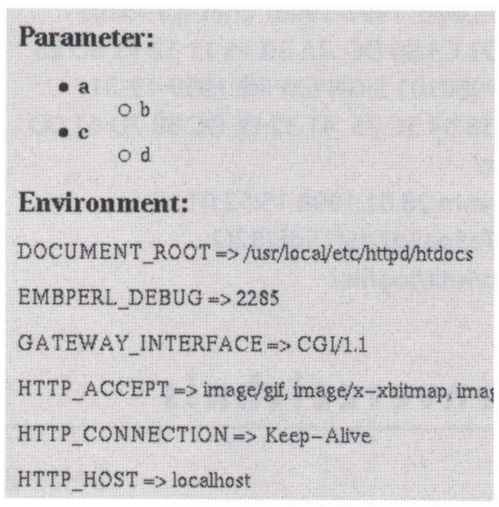


Abb.2: Ausgabe von dump.pl

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 oder http://mission.base.com/mschilli zu erreichen.

Autorenliste

Marco Budde

Kalle Dalheimer

Wolfgang Hetzler

Bernhard Kuhn

Michael Schilli

Tom Schwaller

Till Christian Sierung

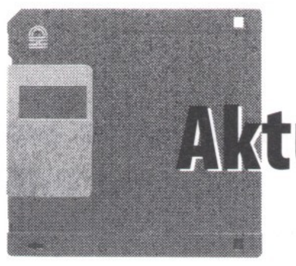
Martin Steigerwald

Daryll Strauss

Norbert Walter

Andreas Zickner

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: 2048/19980101 SIGN EXPIRE:1999-12-31
Key fingerprint = B3 06 9A 8D 38 04 3C 75 41 32 EE DC 8B 7D 61 0D
http://www.in-ca.individual.net/
Hashwert des Ewigen Logfiles vom 28.01.1998 15:52:01
sha1=89e4ce278e09f75a65d37e5ea2d7d5f15df282f2
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)