Standard- und Programmiertools für Unix

Stephan Thesing & Henrik Theiling

4. `perl`: Practical Extraction and Report Language

Bereits gesehen:

- Grundlagen: Variablen, Ausdrücke, Pattern
- einige Befehle und Funktionen

Heute:

- Benutzerdefinierte Funktionen
- CGI-Skripte

Spezielle Variablen in `perl`:

`$_` Default-Wert für viele Operationen

@ARGV Kommandozeilenparameter

`$&` Kompletter Match

$1,...,$9 geklammerte Teilmatches

`$$` Prozeßnummer

`$!` Fehlermeldung

`$@` Array von Werten, z.B. Argumente einer Funktion

`$/` Feldtrenner beim Lesen (voreingestellt auf Newline)

`$[` Erster Array-Index (**Vorsicht**: lieber nicht verändern!)

Drei nützliche Arrays:

%ENV Umgebungsvariablen

@INC Suchpfade für `perl`-Module

%SIG Signal-Handler

```
$editor=$ENV{"EDITOR"};
@paths=split(/:/,$ENV{"PATH"});
unshift(@INC,"$ENV{'HOME'}/lib/perl");
$SIG{'ALARM'}="myalarm"; alarm 5;
sub myalarm { print "5 Sekunden vergangen"; }
```


Subroutinen

Via `sub` *Name Block*:

```
sub sortbynum { $a <=> $b; }
@sorted=sort sortbynum (4,3,3333,282);
```

Werterückgabe: letzter Ausdruck oder `return`

```
sub unsinn  { return 3.14159265; }
sub unsinn2 { 3.14159265; }
```

Achtung: Alle Variablen sind in `perl` zunächst global (auch die in Subroutinen).

Lokale Variablen, Parameter

Mit `my` kann man lokale Variablen deklarieren. Parameter werden an Subroutinen im Feld `@_` übergeben.

```
sub foo1 {
  my $c=$_[0];
  ...
}
```

```
sub foo2 {
  my $c=shift;
  ...
}
```

```
sub foo3 {
  my ($a,$b)= @_;
  ...
}
```


Lokal umdefinierte Variablen, Funktionsaufruf vor Definition

Mit `local` kann man globalen Variablen lokal einen anderen Wert zuweisen, der am Ende des Blocks wieder zurückgesetzt wird. (Unterschied zu `my` beachten!)

```
open (F, "test1.txt") || die $!;
{
   local $/= "\n::\n";
   @records= <F>;
}
close F;

print (sort { &nameof($a) cmp &nameof($b) } @records);

sub nameof {
   $_[0]=~  /\bname=(.*)\n/s;
   $1;
}
```

```
vorname=kurt
name=meier
alter=43
::

vorname=heinrich
name=schrell
strasse=kaiserstraße
::

vorname=henrik
name=theiling
wohnort=saarbrücken
::
```


Packungen: Weg zur Lokalisierung von Namen.

`package` *Name* verschiebt Scope aller folgenden Namen in das Package *Name*.

```
package foo;
$a=100;
$b=200;
package bar;
$a=23;
$b=9.888;
```

Default Package: `main`

Ansprechen von verdeckten Namen in einer anderen Package.

```
$foo="aber";
package a;
$foo=100;

package b;
$foo=0;
$bar=$a'foo+100;
print $main'foo;
```


Einbinden anderer Files mittels `require`

```
require "foo.pl";
unshift(@INC,"$ENV{'HOME'}/lib/perl");
require "mylocal.pl";
```

Neuere Syntax: `::` statt `'` und `use` Anweisung:

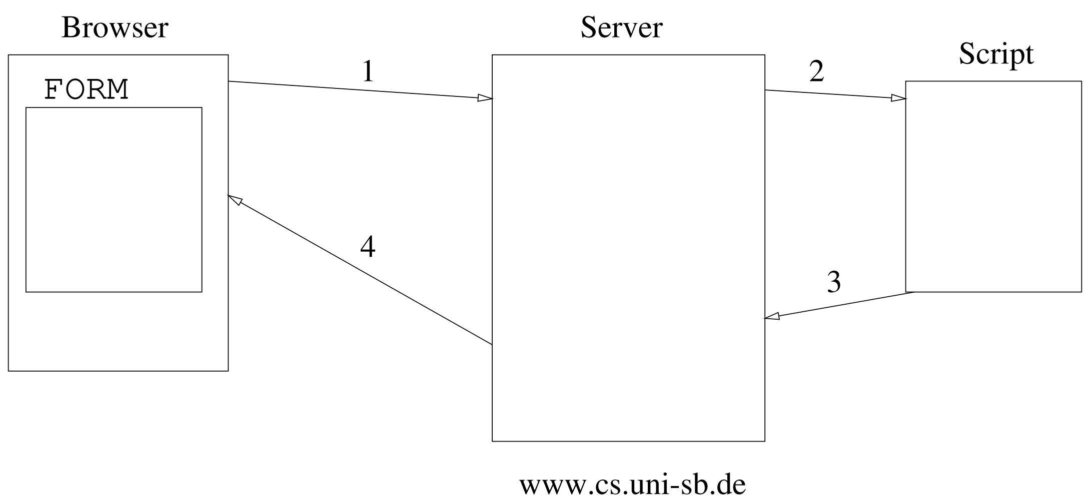
```
use foo $bar $foobar;
print $main::foo;
```

4.1. CGI

Vor Java einzige Methode der Interaktion in Web-Seiten

Prinzip:

- Aus Formularen werden Daten an den Server gesandt
- CGI-Skript (Programm) wird auf dem Server gestartet, bekommt Daten
- Skript produziert Ausgabe, die gültiges HTTP Objekt darstellt



1: Browser kodiert Daten und schickt sie mit der Skript-URL an den Server

2: Server startet Skript, setzt Umgebungsvariablen, übergibt Daten

3: Skript generiert HTTP konforme Ausgabe (Typ Ausgabe+Inhalt)

4: Server stellt einen Header vor die Ausgabe und schickt sie zurück

Aufbau eines CGI-Skripts

Einfacher Fall: in der HTML-Seite: `<A HREF="..../skript.cgi">`.

skript.cgi ist ein Perl-Programm:

```
#! /usr/local/bin/perl
print <<EOM;
Content-type: text/html

<HTML><HEAD><TITLE>Nix</TITLE></HEAD><BODY>
<H1>Hello World!</H1></BODY></HTML>
EOM
```

MIME-Typen: z.B. `text/plain`, `text/html`, `image/gif`, `image/jpeg`, `audio/basic`, ...

Skript-URL kann statt jeder anderen URL stehen. Ausgabe des Skriptes wird eingefügt. Z.B. Ein Bild, das sich bei jedem Aufruf der HTML ändern soll.

In HTML z.B:

`<IMG SRC="http://www.coli.uni-sb.de/~theiling/cgi/d2gif.cgi">`

Perl-Skript, das antwortet:

```
#! /coli/gnu/bin/perl
$HOME="/home/HU/theiling";
@ROBOT= ("bossd117",  "bossd1127", "bossd214", .... );
srand (time ^ ($$*3));
open (F, "$HOME/public_html/gif/$ROBOT[rand($#ROBOT+1)].gif");
undef $/;
print "Content-Type: image/gif\n\n";
print <F>;
close F;
```

Haupteinsatzgebiet: Formulare, zur Bearbeitung der eingegebenen Daten

Formulare:

```
<FORM METHOD="GET"
      ACTION="http://www.cs.uni-sb.de/cgi-bin/Kalender/foo">
Name: <INPUT TYPE=TEXT SIZE=20 NAME=ihr_name>
<P>
<INPUT TYPE=SUBMIT NAME=DOIT VALUE="Abschicken">
</FORM>
```


Viele Eingabetypen. Siehe Docs zu apache, oder W3 Konsortium HTML Docs

Formdaten werden als Name=Wert Paare an die URL des Skriptes angehängt:

```
http://www.cs.uni-sb.de/cgi-bin/Kalender/foo?ihr_name=Jim+Knopf
```

Paare werden durch `&` getrennt: `?a=b&c=d`

Daten müssen kodiert werden:

- Spaces zu `+`
- Alles, was kein Buchstabe ist: %xx (Hex ASCII Code des Zeichens)

Skript kann Datenstring (kodiert) auf zwei Arten bekommen:

1. In der Umgebungsvariablen `QUERY_STRING`

```
$query= $ENV{'QUERY_STRING'};
```

2. Via `STDIN`

```
{ local $/= undef; $query= <>; }
```

Im Formular wird die Methode angegeben: `METHOD=GET` oder `METHOD=POST`.

Umgebung des Skriptes:

Umgebungsvariablen

REQUEST_METHOD Entweder `GET` oder `POST`

QUERY_STRING Formulardaten, wenn `GET` Methode

REMOTE_HOST Rechner, auf dem der Browser läuft

Für Beispielausgabe siehe
`http://www.cs.uni-sb-de/cgi-bin/Kalender/foo?nix`

Files:

STDIN Formulardaten, wenn `POST` Methode

STDOUT Ausgabe, die zum Browser geschickt wird

STDERR Wandert in `error_log` auf dem Server

Es kann schwierig sein, eigene CGI-Skripte zu starten:

Permissions auf dem Server

`chroot` Umgebungen (eingeschränkte Programme zur Verfügung)

Debugging :-(


```
#! /usr/local/bin/perl
print "Content-type: text/html\n\n";
print "<HTML><HEAD><TITLE>Environment</TITLE></HEAD><BODY>\n"

foreach $v (%ENV) { print "$v: $ENV{'$v'}\n<p>\n"; }

if ($ENV{"REQUEST_METHOD"} eq "GET") {
 @pairs=split(/&/,$ENV{"QUERY_STRING"});
 for ($i=0;$i<=$#pairs; $i++) {
  ($k,$v)=split(/=/,$pairs[$i]);
  print "Key $k= $v\n<P>\n";
 }
}

print "</BODY></HTML>\n";
```

Unter `~thesing/lib/cgi-lib.pl` kann man eine Package finden, die beim Parsen hilft.

Z.B.:

```
unshift(@INC,"/home/stud/thesing/lib");
require "cgi-lib.pl";

.....

if (ReadParse(*inputs)) {
 # In %inputs: Key, Value
 foreach $v (keys %inputs) {
   print "$v=$inputs{'$v'}\n<p>\n";
 }
} else die;
```


Zusammenfassung CGI:

- Daten werden kodiert zum Server gesandt, der ein Skript aufruft
- Skript muß Typ und Ausgabe erzeugen
- Ausgabe wird durch URL angestossen
- `perl` nützlich, wg. assoziativer Arrays, Pattern-Matching

Wichtige Manual-Pages

`man perl`	Verweise auf die anderen Perl-Hilfeseiten
`man perlre`	Reguläre Ausdrücke, Pattern-Matching
`man perlfunc`	Vordefinierte Funktionen: `split`, `unlink`, …
`man perlop`	Operatoren: `<`, `lt`, `cmp`, `<=>`, `.=`, …
`man perlvar`	Alles Vordefinierte mit `$` am Anfang: `$/`, `$_`, …
`man perlipc`	„Inter-Process-Communication“: wie man mit anderen (Unter-)Prozessen redet. Und: wie man CGI-Skripten einbruchssicher macht.

`perl`:

- Gemisch von Features aus `awk`, `sed`, der Shell und **C**
- Interpretiert, daher „Rapid-Prototyping“
- Nützliche Pattern-Matching-Konstruktoren
- Manchmal schwer zu lesen (implizite Argumente, etc.)
- Schwach getypt
- Fehlerquellen: Fehlendes Typing, Scoping