

Perl Historie

- Entstanden: 1987
- **P**ractical **E**xtraction and **R**eport **L**anguage
- Syntax stark an C angelehnt
- viele Ähnlichkeiten zu Shell-Scripting
- Perl 5.0 seit 1994 — seitdem Objektorientierung
- Plattformen: Unix, Mac, Windows, ...
- Perl ist einfach, universell, effizient und vollständig
- Starker Einsatz von „Sonderzeichen“

Funktion versus Esthetik

So kann ein Perl Programm aussehen:

```
#
# mod perl Handler
#
sub handler($$) {
    my($class, $r) = @_;

    my $logmsg = '';
    my $docroot = $r->document_root;
    my $uri      = $r->uri;

    #
    # Pfad-Segmente ermitteln
    #
    my $pathdata = $r->pnotes('dirmagic');
    $logmsg      .= Data::Dumper->Dump([ $pathdata ], [ 'pathdata' ]) . "\n";

    ...
}
```

So leider auch:

```
' /\_/\ ' .print[split??"."(($/=q|Cms)+-03467:;<=|)=~tr!C-z -B! -z!)x
'( o.o )' .$/]->[hex]foreach split qr<>,qq+1ecd039ad65b025b8063475b+||
' > ^ < ' .q<!-- Wolfgang Kinkeldei - mailto:wolfgang@kinkeldei.de -->
```

Beispiele zur Syntax

```
#!/usr/bin/perl
#
# Programm zur Sortierung der Eingabe
#

```

oder kürzer:

```
#!/usr/bin/perl

print sort <>;
```

oder als 1-Zeiler:

```
perl -e 'print sort <>'
```

Variablen

- Skalare (=Singularitäten)

```
$alter = 88;  
$name = "Wolfgang";  
$formel = '1 + 2';  
$dateien = `ls -al *.txt`;   
$satz = "Mein Name ist $name, ich werde $alter.";
```

- Arrays

```
@mein_auto = ('VW', 'Audi', 'BMW');  
@unsinnige_liste = (12, 0xef, 'rot');  
$mein_auto[2] = "Smart";           # ersetzt 'BMW'
```

- Hashes

```
%gehalt_von = (kohl => 250_000, merkel => 100000.00);  
%rgbwert_von = (rot => "#ff0000", gruen => "#00ff00");  
$rgbwert_von{rot} = "#ff1111";  
$gehalt_von{billgates} = 1_000_000;
```

```
$person = "billgates";  
print "$person verdient: $gehalt_von{$person}";
```

Funktionen

- Definition einer Funktion:

```
sub meine_funktion {  
    # ..._befehle  
}
```

- Aufruf einer Funktion:

```
meine_funktion( ... argumente ... );  
$variable = meine_funktion( ... argumente ... );  
$variable = meine_funktion;  
$variable = meine_funktion arg1, arg2, ...;
```

- Eingebaute Funktionen:

- Arithmetik
- Stringmanipulation, Regular expressions
- Array- und Hash Manipulation
- Datei-Operationen
- div. Betriebssystemaufrufe, auch Low-Level

Reguläre Ausdrücke

- Beschreibungssprache für Textmustervergleiche

- **beliebiges Zeichen**
- () **Gruppierung**
- [] **eines dieser Zeichen bzw. [^...] keines davon**
- ? **vorangegangenes Symbol nicht oder einmal**
- + **vorangegangenes Symbol mindestens einmal**
- * **vorangegangenes Symbol beliebig oft oder nicht**
- ^ **Zeilenanfang**
- \$ **Zeilenende**
- \ **Escaping und Sonder-Funktionen**

- Beispiele:

[0-9]	genau eine Ziffer
[0-9]+	mindestens eine Ziffer
[0-9]+(,[0-9]+)?	Zahl mit wahlweise Nachkomma-Anteil
^[]*#	Diese Zeile ist ein Perl-Kommentar
</?.*?>	HTML-Tag -- .*? ist Sonderform von *
[a-z_]+\.[a-z]+	„ordentlicher“ Dateiname

Anwenden von Regulären Ausdrücken

- Pattern-Matching

```
# Test auf „nur Ziffern“  
if (m/^[0-9]+$/) { ...
```

```
# Test auf Endung „.jpg“ oder „.gif“  
if ($datei =~ /\.(jpg|gif)$/) { ...
```

```
# Resultat = Bestandteile eines Datums z.B. 2005-10-05  
($y,$m,$d) = m{^([0-9]+)-([0-9]+)-([0-9])$};
```

- Suchen und Ersetzen

```
s/,/./g; # alle Komma durch „.“ ersetzen
```

```
s{^.*//}{}; # Pfad vor Dateiname entfernen  
s{^.*//([^/]+)$}{$1};
```

```
s/([0-9]+)/$sum = $sum + $1/eg;
```

Datei-Operationen

- Lesen von Dateien

```
$zeile = <STDIN>;      # liest vom tty
$zeile = <>;           # list tty oder Pipe oder Argumente
@zeilen = <>;         # liest alle Zeilen auf einmal
```

- Schreiben von Dateien

```
open(OUTFILE, ">dateiname.txt");
print OUTFILE "Inhalt für die Datei\n");
close(OUTFILE);
```

- Datei-Operationen

```
if (-f "/home/user/test.dat") { ... }
$status = stat("/home/user/test.dat");
```

- Directory-Routinen, I/O Funktionen, etc.
z.B. chmod, mkdir, ioctl, ...

Perl 1-Zeiler (1)

- Typischer Aufruf: `perl -e 'Befehlskette'`

```
perl -e 'print crypt("passwort","salt");'  
perl -e 'print sqrt(2);'  
perl -e 'print reverse <>'
```

- Module zur Unterstützung einsetzen: `-M`

Internet-Seite lesen und ausgeben

```
perl -MLWP::Simple -e 'getprint "http://www.heise.de";'
```

Alle Links anzeigen

```
perl -MLWP::Simple -e '$_=get "http://www.heise.de"; \  
s{(<a\s+href.*?</a>)}{print "$1\n"}ige';
```

Alle HTML-Tags rausfiltern

```
perl -MLWP::Simple -e '$_=get "http://www.heise.de"; \  
s{</?[^\>]*>}{}g; print;'
```

Perl 1-Zeiler (2)

- Eingabezeilen verändern: **-p**

eine Mail lesen und "=xx" durch Zeichencode ersetzen

```
perl -p -e 's/=(..)/chr(hex($1))/ge'
```

Buchstaben um 13 Zeichen verschieben (rot13)

```
perl -p -e 'tr/a-z/n-za-m/'
```

Alle Buchstaben GROSS machen

```
perl -p -e '$_ = uc'
```

```
perl -p -e '$_ = uc($_)'
```

```
perl -p -e 'tr/a-z/A-Z/'
```

Perl 1-Zeiler (3)

- Filtern von Eingaben: `-n`

Nur die Zeilen ausgeben, die „mehr“ enthalten:

```
perl -n -e '/mehr/ && print'
```

Doppelte Zeilen herausfiltern:

```
perl -n -e 'print if (!$seen{$_}++)'
```

Doppelte Zeilen angeben:

```
perl -n -e 'print if ($seen{$_}++==1)'
```

Datei von...bis lesen

```
perl -ne 'print if /^START$/ .. /^END$/'
```

Mail-Header abspalten

```
perl -ne 'print if m/^\r*$/ ... 1'
```

Perl 1-Zeiler (4)

- Aufteilung der Eingabe in Spalten: -a

nur 2. Spalte einer Datei ausgeben

```
perl -a -n -F' /\s+/' -e 'print "$F[1]\n"
```

Login und Home-Dir ausgeben:

```
perl -a -n -F: -e 'print "$F[0] - $F[5]\n"' /etc/passwd
```

Verrücktheiten

- Web-Server in einer Zeile

```
perl -MIO::All -e 'io(":8080")->fork->accept->(sub { $_[0] <
io(-x $1 ? "./$1 |" : $1) if /^GET \/(.*) / })'
```

- Was macht dieses Programm?

```
#!/usr/bin/perl
eval{ require Win32::Console::ANSI;
};sub {::printf @_ } sub {
rand 39/999} sub 0123 {::
"" ."\e[%d" .";%dH%s" ,@_
};; sub 1234 {0123 $|=
,1, "\e[J"} while(1) {
$c or do {::"\e[1"
"" .";%dm", 30+ rand
7 ; 1234;$u=&sub ;$v =
&sub ;$ c= 3999 ;$b=qw
/19 9 1/[int rand 3 ]};0123
11*sin( $v* $c)+ 13,39*cos ($u*$c)+41,
qw+J A P H+[$c%4] ; ($a++%$b)or$c--;}
-w -w -w -w -w
```

Praktische Beispiele

Plain Text umbrechen

```
#!/usr/bin/perl
```

```
use Text::Wrap;  
$Text::Wrap::columns = 60;  
print wrap(" ", " ", "<>");
```

oder:

```
#!/usr/bin/perl
```

```
use Text::Autoformat;  
$text = join(" ", <>);  
print autoformat($text, {left=>4,right=>70,all=>1});
```

Praktische Beispiele

XML Datei auslesen

```
#!/usr/bin/perl

use XML::Twig;

$xml = XML::Twig->new();
$xml->parsefile($ARGV[0]);

foreach my $title ($xml->get_xpath("item/title")) {
    print $title->text() . "\n";
}
```

Praktische Beispiele

Debugging made easy :-)

```
#!/usr/bin/perl

use Smart::Comments;

$counter = 20;

### counter: $counter

for $i (1..$counter) { ### Rechne [===|           ] % fertig
    sleep(1);
}
```

Ende

Danke fürs Zuhören

Real programmers can write assembly code in any language.
Larry Wall