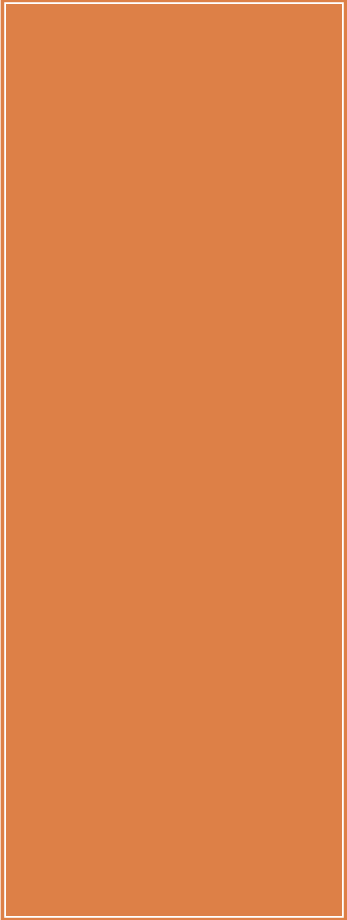


# PERL

*“the Swiss Army Chainsaw of Programming Languages.”*

# Inhalt

- 
- Perl
  - Syntax
  - Operatoren
  - Variablen
  - Bedingungen und Schleifen
  - Funktionen
  - Referenzen

# Inhalt

---

- 
- Dateizugriff
  - Module
  - CGI
  
  - Fragen

# Perl

- Aktuelle Version: 5.8.8
- Erschienen: 1987
- Name stammt aus dem Mathäus-Evangelium (Pearl), weitere nachträglich gewählte Acronyme:
  - ▣ *Practical Extraction and Report Language*  
(zweckmäßige Extraktions- und Berichtssprache)
  - ▣ *Pathologically Eclectic Rubbish Lister*  
(krankhaft vielseitiger Blödsinnsauflister)

# Perl

- Plattformunabhängig
- Timtowtdi („Tim today“): There is more than one way to do it
- Scriptsprache, die aber vor Ausführung vollständig kompiliert wird.
- Für Windows: activeperl ([www.activestate.com](http://www.activestate.com))
- Lizenz: Wahlweise GPL oder Artistic License

# Larry Wall



“It is easier to port a shell than a shell script.”

“Only perl can parse Perl.”

“It won't be covered in the book. The source code has to be useful for something, after all... :-)”

# Anwendungsbereiche von Perl

- Umgang mit Daten aller Art (z.B. logfiles)
  - ▣ Extrahieren und Auswerten
  - ▣ Erweiterung von bash, sed, awk, grep, sort, etc.
- Common Gateway Interface (CGI)
- Automatisierung häufiger Aufgaben
- Schnell zu schreibende Vorschau auf Software

# Schätzung



Philipp Burch  
philippb@activevb.de  
Arbeitsbereich:  
Tutorials  
Besondere Kenntnisse:  
Hardwarenahe Entwicklung

Konrad Doblander  
Konrad@ActiveVB.de  
Arbeitsbereich:  
Foren-Betreuung

Bernhard Döbler  
Bard@ActiveVB.de  
Arbeitsbereich:  
Foren-Betreuung  
Besondere Kenntnisse:  
Datenbanken, Software Engineering (Softwaredesign), Windows Internals

Klaus Langbein  
Klaus@ActiveVB.de  
Arbeitsbereich:  
Administration, Foren-Betreuung  
Besondere Kenntnisse:  
API, Peek 'n Poke, Sound

Achim Neubauer  
Achim@ActiveVB.de  
Arbeitsbereich:  
Foren-Betreuung, Scripte

Dietrich Nohl  
dnohl@dnohl.ch  
Arbeitsbereich:  
Foren-Betreuung

Daniel Noll  
Daniel@ActiveVB.de  
Arbeitsbereich:  
VB .NET-Rubrik und Tipps

Text: philippb@activevb.de:  
Mail: philippb@activevb.de

Text: Konrad@ActiveVB.de:  
Mail: Konrad@ActiveVB.de

Text: Bard@ActiveVB.de:  
Mail: Bard@ActiveVB.de

Text: Klaus@ActiveVB.de:  
Mail: Klaus@ActiveVB.de

Text: Achim@ActiveVB.de:  
Mail: Achim@ActiveVB.de

Frage:

Wie viele Zeilen/Zeichen Code sind nötig, um aus einem Quelltext einer Webseite alle E-Mail-Adressen zu extrahieren?



# Schätzung

```
#!/usr/bin/perl -w

while(<>) {
    print "Text: $2:\nMail: $1\n\n"
    while(/<a[^"]+href="mailto:([^"]+)"[^>]*>([^<]+)<\a>/gi)
    }
}
```

Frage:

Wie viele Zeilen/Zeichen Code sind nötig, um aus einem Quelltext einer Webseite alle E-Mail-Adressen zu extrahieren?

# Syntax

- Sehr freie Syntax (Problem)
- Klammern können ignoriert werden, wenn Bedeutung klar ist:

```
$text = lc $text;
```

```
$text = lc($text); #identisch
```

```
$text = lc 'Hallo' . ' welt'; # hallo welt
```

```
$text = lc('Hallo') . ' welt'; # hallo welt
```

# Strings

- single quoted strings:

'Hallo welt' # -> Hallo welt

'Hallo welt\n' # -> Hallo welt\n

- double quoted strings:

"Hallo welt" # -> Hallo welt

"Hallo welt\n" # -> Hallo welt +  
Zeilenumbruch

# Operatoren

## □ Vergleichsoperatoren

Symbol (Numerisch)	Symbol (String)	Bedeutung
==	eq	Gleich
<	lt	Kleiner
<=	le	Kleiner oder Gleich
>	gt	Größer
>=	ge	Größer oder gleich
!=	ne	Ungleich

„hallo“ == 0, „3hallo“ == 3, „3hallo“ ne „0“

# Operatoren

- Logische Operatoren:  
 && || ! sowie and or not
- Rechenoperatoren:  
 + - \* / \*\* sowie += -= \*= /= \*\*= und ++ --
- Stringoperatoren: . (Punkt) x sowie . =
- Vergleichsoperatoren: <=> cmp

# Variablen

- Perl kennt keine Datentypen
- Es gibt 3 Arten von Variablen
  - ▣ Skalare
  - ▣ Arrays
  - ▣ Hashs („assoziierte Arrays“)
- Funktionen können Daten Kontext-sensitiv (Array oder Skalar) zurückgeben.

# Variablen

- Variablen werden implizit deklariert
- Für explizite Deklaration:

**use strict;**

- Die explizite Deklaration erfolgt mit my:

**my \$var;**

**my (\$var1, @var2, \$var3);**

- Namen dürfen aus A-Z, a-z, 0-9 und „\_“ bestehen.

# Variablen

```
$var = undef  
$var = 42
```

```
$copy = $var
```

```
@arr = ()  
@arr = (1, 2)
```

```
$arr[0] = 25
```

```
$cnt = @arr  
@cpy = @arr
```

```
%hsh = ()  
%hsh = (  
  a => 'b',  
  c => 'd')
```

```
$hsh{a} = 'c'
```

```
@el = %hsh  
%cpy = %hsh
```



# Funktionen

## □ Stringmanipulation

- ▣ substr → extrahiert und ersetzt Stringteile.

`$Erg = substr($Text, $Start [, $Länge, [$Ersatz]])`

```
$text = 'Hallo ActiveVB!';  
$hello = substr($text, 0, 5); # left  
$avb = substr($text, -9); # right  
$space = substr($text, 5, 1); # mid  
substr($text, 6, 8, 'welt'); # (mid)  
print $text; # Hallo welt!
```

# Funktionen

## □ Stringmanipulation

- `$str = reverse($Text)` → dreht Text um

- `@arr = reverse(@array)` → dreht Array um

```
print lc(reverse("Die Liebe ist Sieger; "  
    ".stets rege ist sie bei Leid")) . "\n";
```

- `$len = length($Text)` → len

- `$pos = index($Text, $Suchwort [, $Startpos])` → instr

# Funktionen

- Funktionen zur Manipulation von Arrays
  - ▣ `$len = push(@array, @werte)`
  - ▣ `$len = unshift(@array, @werte)`
  
  - ▣ `$deleted = shift(@array)`
  - ▣ `$deleted = pop(@array)`

# Funktionen

- Funktionen zur Manipulation von Arrays
  - ▣ `@deleted = splice(@Array, $Start[, $Anzahl[, @Ersatz]])`
  - ▣ `sort [{Code}] Array`
    - Code vergleicht \$a und \$b
    - Zahlen sortieren: `sort {$a <=> $b} @zahlen`
    - Texte umgekehrt sortieren: `sort {$b cmp $a} @texts`

# Funktionen

## □ Funktionen zur Manipulation von Arrays

### ▣ map {Code} Array

```
my @array = 1 .. 10;  
my @double = map {$_ * 2} @array;
```

### ▣ grep {Code} Array

```
my @even = grep {$_ % 2 == 0} @array;  
my @defs = grep {defined} @array;
```

# Variablen – Funktionen

- Funktionen zum Zugriff auf Hashs
  - ▣ @keys = keys(%hash)
  - ▣ @values = values(%hash)

```
my @names = keys(%mails);
```

```
my @mails = values(%mails);
```

# Bedingungen und Schleifen

## □ Bedingungen:

```
if($a == 3) {  
    print "a ist 3\n";  
}
```

```
unless($a == 3) {  
    print "a ist nicht 3\n";  
}
```

```
print "a ist 3\n" if $a == 3;  
print "a ist nicht 3\n" unless $a == 3;
```

```
print 'a ' . ($a == 3 ? 'ist' : 'ist nicht') . " 3\n";
```

# Bedingungen und Schleifen

## □ Schleifen:

```
while($a > 3)
{
    $a--;
}
```

```
until($a == 5)
{
    $a++;
}
```

```
for($i = 0; $i < @array; $i++)
{
    print "$i. Element: " .
        $array[$i] . "\n";
}
```

```
for(@array) # foreach
{
    print "$_\n";
}
```



# Eigene Funktionen

- Deklaration:

sub Name {Code}

- Aufruf:

&Name(Parameter);

- Aufruf ohne Klammern nur mit Prototypen möglich (wird hier nicht behandelt)
- Parameter werden in @\_ übertragen
- wantarray für kontextsensitive Rückgabewerte

# Eigene Funktionen

```
#!/usr/bin/perl
use strict ;

my $x = 30;

print "fib($x) = " . &fib($x) . "\n";

sub fib {
    my ($f) = @_;

    return 1 if $f <= 1;
    return fib($f - 1) + fib($f - 2);
}
```

# Referenzen

- Erzeugung:
  - \\$var, \@var, \%var, \&func
  - [...], {...}, sub {...}
  
- Zugriff:
  - ▣ \$\$ref, @\$ref, %\$ref
  - ▣ \$ref->[\$nr]
  - ▣ \$ref->{\$key}
  - ▣ \$funcref->(@args)

# Referenzen – Beispiel

```
@children = ('Tom', 'Hans');
```

```
%person = (  
    name1 => 'Max', name2 => 'Mustermann',  
    children => \@children,  
);
```

```
push @person, %person;
```

```
print $person[-1]->{name} = 'Susanne';  
print $person[-1]->{children}->[0] . "\n" # Tom
```

# Dispatch Tables

```
sub alias {  
    my ($new, $old) = @_;  
    $dispatch{$new} = $dispatch{$old};  
}
```

```
%dispatch = (  
    a => sub{print "Funktion A\n"},  
    b => sub{print "Funktion B\n"},  
    alias => \&alias,  
);
```

```
$dispatch{a}->(); # Funktion A  
$dispatch{alias}->('c', 'b');  
$dispatch{c}->(); # Funktion B
```

# Dateizugriff

- Öffnen und schließen:

```
my $fh;
```

```
open $fh, 'datei'; # lesen
```

```
open $fh, '<datei'; # identisch
```

```
open $fh, '>datei'; # schreiben
```

```
open $fh, '>>datei'; # anhängen
```

```
close $fh;
```

# Dateizugriff

## □ Lesen:

```
my $zeile = <$fh>; # Zeile 1
```

```
$zeile = <$fh>; # Zeile 2 ...
```

```
read($fh, $bytes, 512); # 512 Zeichen
```

## □ Schreiben:

```
print $fh "Zeile\n";
```

# Dateizugriff

- Weitere Funktionen
  - ▣ Spulen (seek), Kürzen (truncate)
  - ▣ Sperren (flock)
- Informationen erhalten
  - ▣ stat, -s (size), -f (file exists), -d (dir exists), ...
- Verzeichnishandhabung
  - ▣ opendir, readdir, closedir



# Module

- Pragas werden zur Compilezeit eingebunden
  - ▣ strict, warnings, perligata
- Module werden zur Laufzeit eingebunden
  - ▣ CGI, template, memoize, bignum, imap, Dumper, lwp, und ca. 15.000 weitere CPAN-Module
- Einbinden mit use
  - use Modulname 'Parameter'

# Module – Beispiel

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

```
my $x = 32; my $calls = 0;
print "fib($x) = " . fib($x) . " - Needed $calls calls\n";

sub fib {
    my ($f) = @_ ; $calls++;

    return 1 if $f <= 1;
    return fib($f - 1) + fib($f - 2);
} # fib(30) = 2692537 - Needed 2692537 calls
```

# Module – Beispiel

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

```
use Memoize; memoize 'fib';
```

```
my $x = 99; my $calls = 0;
```

```
print "fib($x) = " . fib($x) . " - Needed $calls calls\n";
```

```
sub fib {  
    my ($f) = @_; $calls++;
```

```
    return 1 if $f <= 1;
```

```
    return fib($f - 1) + fib($f - 2);
```

```
} # fib(98) = 3.54224848179262e+020 - Needed 100 calls
```

# CGI

- ❑ Lesen von STDIN
- ❑ Lesen der Umgebungsvariablen
- ❑ Exportieren der interessanten Informationen
- ❑ Bereitstellung von Funktionen für hochgeladene Dateien
- ❑ Bereitstellung von HTML & HTTP-Elementen (z.B. Tabellen, HTTP-Header)
- ❑ Debugging für die Konsole

# CGI



Change me! go

```
<html><head><title>Ein Test</title></head><body>  
<form method="post" action="test.pl">  
    <input type="hidden" name="versteckt" value="1">  
    <input type="text" name="msg" value="Change me!">  
    <input type="submit" name="go" value="go">  
</form>  
</body></html>
```

# CGI

```
#!/usr/bin/perl
use strict;
use warnings;
use CGI ':standard';

print header(); # aus CGI
print start_html('testseite'); # aus CGI

my @fields = param(); # aus CGI

print "Gesetzte Felder: " . join(', ', @fields) . "<br>";
print "Your Message: " . param('msg');
print end_html(); # aus CGI
```



# CGI – Weitere Module

- Sessionverwaltung
- Cookieverwaltung
- Ajax & Webapplikationen
- Abstrahierter Datenbankzugriff
- Dynamische Erzeugung von Bildern mit ImageMagick oder libGD

# Quellen

---

- ❑ [perl.com](http://perl.com)
- ❑ [wall.org/~larry](http://wall.org/~larry)
- ❑ [perlmonks.com](http://perlmonks.com)
- ❑ [rocketaware.com/perl/](http://rocketaware.com/perl/)
- ❑ [wikipedia.org](http://wikipedia.org)



"WHAT IS THE SOUND OF PERL? IS IT NOT THE  
SOUND OF A WALL THAT PEOPLE HAVE STOPPED  
BANGING THEIR HEADS AGAINST? " – LARRY WALL

Vielen Dank für Ihre Aufmerksamkeit