Eine Einführung in Perl

Sebastian Vogl
vogls@in.tum.de

Proseminar Unix-Tools
Technische Universität München

06.12.2005

Gliederung

Gliederung

Gliederung

Wie ist Perl entstanden ?

- 1987 ist Larry Wall an der Entwicklung eines sicheren Netzwerks für die NSA beteiligt

Wie ist Perl entstanden ?

- 1987 ist Larry Wall an der Entwicklung eines sicheren Netzwerks für die NSA beteiligt
- seine Aufgabe: Entwicklung von Administrations-Tools
  $\Rightarrow$ Fehlersuch-Tool für verstreute Logdateien

Wie ist Perl entstanden ?

- 1987 ist Larry Wall an der Entwicklung eines sicheren Netzwerks für die NSA beteiligt
- seine Aufgabe: Entwicklung von Administrations-Tools
  $\Rightarrow$ Fehlersuch-Tool für verstreute Logdateien
- vorhandenen Werkzeuge zu umständlich
  $\Rightarrow$ Entwicklung eines Mehrzweck-Tools

Wie ist Perl entstanden ?

- 1987 ist Larry Wall an der Entwicklung eines sicheren Netzwerks für die NSA beteiligt
- seine Aufgabe: Entwicklung von Administrations-Tools
  $\Rightarrow$ Fehlersuch-Tool für verstreute Logdateien
- vorhandenen Werkzeuge zu umständlich
  $\Rightarrow$ Entwicklung eines Mehrzweck-Tools
- Dezember 1987: Publizierung von Perl 1.0 im Usenet

Wie ist Perl entstanden ?

- 1987 ist Larry Wall an der Entwicklung eines sicheren Netzwerks für die NSA beteiligt
- seine Aufgabe: Entwicklung von Administrations-Tools
  $\Rightarrow$ Fehlersuch-Tool für verstreute Logdateien
- vorhandenen Werkzeuge zu umständlich
  $\Rightarrow$ Entwicklung eines Mehrzweck-Tools
- Dezember 1987: Publizierung von Perl 1.0 im Usenet
- Aktuelle Version 5.8.7

Gliederung

Wo liegt der Ursprung von Perl ? (I)

Geistig

- geprägt durch Larry Wall

Einschub: Wer ist Larry Wall ? (I)

Einschub: Wer ist Larry Wall ? (II)

- Erfinder und Entwickler von Perl

Einschub: Wer ist Larry Wall ? (II)

- Erfinder und Entwickler von Perl
- Hauptverantwortlicher für Weiterentwicklung

Einschub: Wer ist Larry Wall ? (II)

- Erfinder und Entwickler von Perl
- Hauptverantwortlicher für Weiterentwicklung
- streng gläubiger Christ

Einschub: Wer ist Larry Wall ? (II)

- Erfinder und Entwickler von Perl
- Hauptverantwortlicher für Weiterentwicklung
- streng gläubiger Christ
- Linguist

Einschub: Wer ist Larry Wall ? (II)

- Erfinder und Entwickler von Perl
- Hauptverantwortlicher für Weiterentwicklung
- streng gläubiger Christ
- Linguist
- Humorist

Wo liegt der Ursprung von Perl ? (I)

Geistig

- geprägt durch Larry Wall

⇒ Perl Eigenschaften

- Orientierung an menschlichen Sprachgewohnheiten

Wo liegt der Ursprung von Perl ? (I)

Geistig

- geprägt durch Larry Wall

⇒ Perl Eigenschaften

- Orientierung an menschlichen Sprachgewohnheiten
- reicher Wortschatz

Wo liegt der Ursprung von Perl ? (I)

Geistig

- geprägt durch Larry Wall

⇒ Perl Eigenschaften

- Orientierung an menschlichen Sprachgewohnheiten
- reicher Wortschatz
- praktisch ausgelegt

Wo liegt der Ursprung von Perl ? (I)

Geistig

- geprägt durch Larry Wall

⇒ Perl Eigenschaften

- Orientierung an menschlichen Sprachgewohnheiten
- reicher Wortschatz
- praktisch ausgelegt
- Perl handelt wie Larry denkt

Wo liegt der Ursprung von Perl ? (II)

Praktisch: Kombination aus

- C
- Unix-Tools (sed, grep, awk etc.)
- Bourne Shell
- eigenen Ideen

Wo liegt der Ursprung von Perl ? (II)

Praktisch: Kombination aus

- C
- Unix-Tools (sed, grep, awk etc.)
- Bourne Shell
- eigenen Ideen

⇒ Perl Eigenschaften

Wo liegt der Ursprung von Perl ? (II)

Praktisch: Kombination aus

- C
- Unix-Tools (sed, grep, awk etc.)
- Bourne Shell
- eigenen Ideen

$\Rightarrow$ Perl Eigenschaften

- einfach zu benutzen

Wo liegt der Ursprung von Perl ? (II)

Praktisch: Kombination aus

- C
- Unix-Tools (sed, grep, awk etc.)
- Bourne Shell
- eigenen Ideen

$\Rightarrow$ Perl Eigenschaften

- einfach zu benutzen
- meistens schnell

Wo liegt der Ursprung von Perl ? (II)

Praktisch: Kombination aus

- C
- Unix-Tools (sed, grep, awk etc.)
- Bourne Shell
- eigenen Ideen

⇒ Perl Eigenschaften

- einfach zu benutzen
- meistens schnell
- fast unbegrenzt

Wo liegt der Ursprung von Perl ? (II)

Praktisch: Kombination aus

- C
- Unix-Tools (sed, grep, awk etc.)
- Bourne Shell
- eigenen Ideen

$\Rightarrow$ Perl Eigenschaften

- einfach zu benutzen
- meistens schnell
- fast unbegrenzt
- ein bißchen häßlich

Gliederung

Was bedeutet "Perl" ?

- ursprünglicher Name "Pearl" (die Perle)

Was bedeutet "Perl" ?

- ursprünglicher Name "Pearl" (die Perle)
- Practical Extraction and Report Language

Was bedeutet "Perl" ?

- ursprünglicher Name "Pearl" (die Perle)
- Practical Extraction and Report Language
- Pathologically Eclectic Rubbish Listener[1]

Was bedeutet "Perl" ?

- ursprünglicher Name "Pearl" (die Perle)
- Practical Extraction and Report Language
- Pathologically Eclectic Rubbish Listener[1]

[1] krankhaft zusammengeschustertes Auflistungsprogramm für wirres Zeug

Gliederung

Die wichtigsten Perl-Prinzipien (I)

There is more than one way to do it (TIMTOWTDI)

Es gibt absichtlich verschiedene Forumlierungs- und Lösungsmöglichkeiten

$\Rightarrow$ Der Programmierer kann seine persönlichen Vorlieben einbringen

Die wichtigsten Perl-Prinzipien (II)

There is more than one way to do it (TIMTOWTDI)

Es gibt absichtlich verschiedene Forumlierungs- und Lösungsmöglichkeiten

$\Rightarrow$ Der Programmierer kann seine persönlichen Vorlieben einbringen

Perl makes easy jobs easy and hard jobs possible

- einfache Dinge sollen schnell realisierbar sein

Die wichtigsten Perl-Prinzipien (II)

There is more than one way to do it (TIMTOWTDI)

Es gibt absichtlich verschiedene Forumlierungs- und Lösungsmöglichkeiten

$\Rightarrow$ Der Programmierer kann seine persönlichen Vorlieben einbringen

Perl makes easy jobs easy and hard jobs possible

- einfache Dinge sollen schnell realisierbar sein
- schwierige Dinge sollen wenigstens möglich sein

Die wichtigsten Perl-Prinzipien (III)

There is more than one way to do it (TIMTOWTDI)

Es gibt absichtlich verschiedene Forumlierungs- und Lösungsmöglichkeiten

$\Rightarrow$ Der Programmierer kann seine persönlichen Vorlieben einbringen

Perl makes easy jobs easy and hard jobs possible

- einfache Dinge sollen schnell realisierbar sein
- schwierige Dinge sollen wenigstens möglich sein

Kontextsensitiv

Befehle haben je nach Kontext verschiedene Bedeutung

Gliederung

Das Programm perl

- in C geschriebener Interpreter

Das Programm perl

- in C geschriebener Interpreter
- für alle gängigen Betriebssystem verfügbar

Das Programm perl

- in C geschriebener Interpreter
- für alle gängigen Betriebssystem verfügbar
- wandelt Perlscript(Textdateien) in Bytecode um

Gliederung

Gliederung

Freie Formatierung und Kommentare

- Quellcode frei formatierbar

Freie Formatierung und Kommentare

- Quellcode frei formatierbar
- Befehle werden mit ; abgeschlossen

Freie Formatierung und Kommentare

- Quellcode frei formatierbar
- Befehle werden mit ; abgeschlossen
- keine main Routine

Freie Formatierung und Kommentare

- Quellcode frei formatierbar
- Befehle werden mit ; abgeschlossen
- keine main Routine
- Kommentare beginnen mit #-Zeichen

Freie Formatierung und Kommentare

- Quellcode frei formatierbar
- Befehle werden mit ; abgeschlossen
- keine main Routine
- Kommentare beginnen mit #-Zeichen

Beispiel

```
#!/usr/bin/perl
# Ich bin ein Kommentar
print "Hallo Welt!";
```

Die erste Zeile eines Perl Programms

#!/usr/bin/perl (-w)

- Bedeutung: Diese Datei soll von /usr/bin/perl ausgeführt werden

Die erste Zeile eines Perl Programms

#!/usr/bin/perl (-w)

- Bedeutung: Diese Datei soll von /usr/bin/perl ausgeführt werden
- Optionen können übergeben werden

Die erste Zeile eines Perl Programms

#!/usr/bin/perl (-w)

- Bedeutung: Diese Datei soll von /usr/bin/perl ausgeführt werden
- Optionen können übergeben werden
- **-w**: Warnungen einschalten

Gliederung

Operatoren I

Arithmetische Operatoren

+	Addition
-	Subtraktion
*	Multiplikation
/	Divison
%	Modulo
**	Exponent

Operatoren II

String Operatoren

.	aneinanderhängen
x	Wiederholungsoperator

Beispiel

```
"Hallo " . "Welt!"; # liefert "Hallo Welt!"
"ha" x 3; # liefert "hahaha"
```

Operatoren III

Logische Operatoren

&&	oder	and	Und
||	oder	or	Oder
!	oder	not	Nicht

Operatoren IV

Vergleichs Operatoren

==	oder	eq	gleich
!=	oder	ne	ungleich
<	oder	lt	kleiner
>	oder	gt	größer
<=	oder	le	kleiner gleich
>=	oder	ge	größer gleich

Gliederung

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...
- auf Skalare kann man Operatoren anwenden

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...
- auf Skalare kann man Operatoren anwenden

Skalare Variablen

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...
- auf Skalare kann man Operatoren anwenden

Skalare Variablen

- beginnen mit dem **$**-Zeichen

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...
- auf Skalare kann man Operatoren anwenden

Skalare Variablen

- beginnen mit dem **$**-Zeichen
- Name: alphanummerischen Zeichen und Unterstriche

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...
- auf Skalare kann man Operatoren anwenden

Skalare Variablen

- beginnen mit dem **$**-Zeichen
- Name: alphanummerischen Zeichen und Unterstriche
- Name: muss mit Buchstaben beginnen und ist case sensitiv

Datenstrukturen - Skalare I

Was ist ein Skalar ?

- einfachste Datentyp in Perl
- repräsentiert etwas Einzelnes ("Singular")
  z.B eine Zahl, einen String, eine Referenz, ein Objekt...
- auf Skalare kann man Operatoren anwenden

Skalare Variablen

- beginnen mit dem **$**-Zeichen
- Name: alphanummerischen Zeichen und Unterstriche
- Name: muss mit Buchstaben beginnen und ist case sensitiv
- haben standardmäßig den Wert **undef**

Datenstrukturen - Skalare II

Beispiele

```
$ein_string    = "Hallo Welt!";              # Hallo Welt!
$ein_string2   = "Ich sage: $ein_string";    # Ich sage: Hallo Welt!
$string_single = 'Ich sage: $ein_string';    # Ich sage: $ein_string
$eine_zahl     = 3_768;                      # eine_zahl = 3768
$rechnung      = $eine_zahl - 26;            # rechnung = 3742
$rechnung2     += 42;                        # rechnung2 = 42
```

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich
- beliebige Anzahl von Elementen $\Rightarrow$ "Grenzenlosigkeit"

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich
- beliebige Anzahl von Elementen $\Rightarrow$ "Grenzenlosigkeit"

Arrays

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich
- beliebige Anzahl von Elementen $\Rightarrow$ "Grenzenlosigkeit"

Arrays

- Variable die eine Liste enthält

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich
- beliebige Anzahl von Elementen $\Rightarrow$ "Grenzenlosigkeit"

Arrays

- Variable die eine Liste enthält
- beginnt mit dem **@**-Zeichen

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich
- beliebige Anzahl von Elementen $\Rightarrow$ "Grenzenlosigkeit"

Arrays

- Variable die eine Liste enthält
- beginnt mit dem **@**-Zeichen
- Zugriff auf die Elemente erfolgt durch **$arrayname[Index]**

Datenstrukturen - Listen und Arrays I

Listen

- geordnete Sammlung von skalaren Werten("Plural")
- verschiedene skalare "Typen" möglich
- beliebige Anzahl von Elementen $\Rightarrow$ "Grenzenlosigkeit"

Arrays

- Variable die eine Liste enthält
- beginnt mit dem **@**-Zeichen
- Zugriff auf die Elemente erfolgt durch **$arrayname[Index]**
- **Nützliche Funktionen**: pop, push (Array Ende) bzw. shift, unshift (Array Anfang), reverse und sort

Datenstrukturen - Listen und Arrays II

Beispiele

```
($zahl1, $zahl2, $string) = (7, 42, "Test"); # Listenzuweisung
($word[0], $word[1], $word[2]) = qw/ Der Die Das /;
@myarray = 1..42; # Array mit Elementen von 1-42
$ende = $#myarray; # $ende = 41
$erstes_e = $myarray[0]: # $erstes_e = 1
$letztes_e = $myarray[-1]; # $letztes_e = 42
$test = $myarray[-43]; # ERROR
$letztes_e = pop(@myarray); # @myarray = 1-41, $letztes_e = 42
$erste_e = shift(@myarray); # @myarray = 2-42, $erste_e = 1
```

Datenstrukturen - Hashes I

Hashes - assoziative Arrays

- besteht wie ein Array aus einer Liste von Werten(**values**)

Datenstrukturen - Hashes I

Hashes - assoziative Arrays

- besteht wie ein Array aus einer Liste von Werten(**values**)
- der Index heisst hier Schlüssel(**key**)

Datenstrukturen - Hashes I

Hashes - assoziative Arrays

- besteht wie ein Array aus einer Liste von Werten(**values**)
- der Index heisst hier Schlüssel(**key**)
- Schlüssel sind beliebige einmalige Strings

Datenstrukturen - Hashes I

Hashes - assoziative Arrays

- besteht wie ein Array aus einer Liste von Werten(**values**)
- der Index heisst hier Schlüssel(**key**)
- Schlüssel sind beliebige einmalige Strings
- Zugriff erfolgt über **$hash{$schluessel}**

Datenstrukturen - Hashes I

Hashes - assoziative Arrays

- besteht wie ein Array aus einer Liste von Werten(**values**)
- der Index heisst hier Schlüssel(**key**)
- Schlüssel sind beliebige einmalige Strings
- Zugriff erfolgt über **$hash{$schluessel}**
- beginnt mit dem **%**-Zeichen

Datenstrukturen - Hashes I

Hashes - assoziative Arrays

- besteht wie ein Array aus einer Liste von Werten(**values**)
- der Index heisst hier Schlüssel(**key**)
- Schlüssel sind beliebige einmalige Strings
- Zugriff erfolgt über **$hash{$schluessel}**
- beginnt mit dem **%**-Zeichen
- **Nützliche Funktionen**: reverse, sort, keys, values, delete und exists

Datenstrukturen - Hashes II

Beispiele

```
%myhash = ("schluessel1", "wert1", "schlussel2", "wert2");
%myhash = ("schluessel1" => "wert1", "schluessel2" => "wert2");
$einwert = $myhash{"schluessel1"};
$myhash{"schluessel3"} = "wert3";
@schluessel = keys %myhash;
delete $myhash{"schluessel1"};
```

Datenstrukturen - Kontextabhänig

Beispiele

```
$test    =  5 + @myarray;              # 5 + Arraylaenge
$test    =  reverse qw/ ha ho hi/;     # $test = ihohah
@test    =  reverse qw/ ha ho hi/;     # (hi, ho, ha)
$test    =  scalar @myarray . " Elemente";
@myarray =  %myhash;                   # Liste key/value Paaren
```

Kontextabhänig

- je nach Kontext verschiedene Bedeutung
- Rückgabe je nach Implementierung

Gliederung

Kontrollstrukturen - if

Syntax

if (*Bedingung*) { *Ausdruck* } (**elsif** (...) {...}) (**else** {...})
unless (*Bedingung*) (**else** {...})
Bedingung **?** *Ausdruck_wahr* **:** *Ausdruck_falsch*

Beispiel

```
if ($meintest) {
   print "wahr";
} else {
   print "falsch";
}
```

Kontrollstrukturen - while

Syntax

while (*Bedingung*) { *Ausdruck* }
until (*Bedingung*) { *Ausdruck* }
for (*Initialisierung; Test; Increment*) { *Ausdruck* }

Beispiel

```
$n = 0;
while ($n < 10) {
        $n++;
}
```

Kontrollstrukturen - foreach

Syntax

foreach *$Kontrollvariable* (*Liste*) { *Ausdruck* }

Beispiel

```
foreach (@myarray) {
        print; # $_ Standardvariable
}
```

Kontrollstrukturen - Schleifen kontrollieren

Syntax

last - beendet eine Schleife sofort (vgl. break)
next - springt an das Ende des aktuellen Blocks (vgl. continue)
redo - springt zum Anfang des Schleifenblocks zurück

Gliederung

Subroutinen I

Syntax

- **sub** *Name* { *Ausdruck* }

Subroutinen I

Syntax

- **sub** *Name* { *Ausdruck* }
- Aufruf: **&Name**($arg_1, arg_2, ..., arg_n$)

Subroutinen I

Syntax

- **sub** *Name* { *Ausdruck* }
- Aufruf: **&Name**($arg_1, arg_2, ..., arg_n$)
- der Wert der zuletzt berechnet wird ist der Rückgabewert, dieser kann aber auch mit **return** *Wert* festgelegt werden.

Subroutinen I

Syntax

- **sub** *Name* { *Ausdruck* }
- Aufruf: **&Name**($arg_1, arg_2, ..., arg_n$)
- der Wert der zuletzt berechnet wird ist der Rückgabewert, dieser kann aber auch mit **return** *Wert* festgelegt werden.
- Argumente befinden sich in der Standardvariable @_
  $\Rightarrow$ Zugriff erfolgt durch $_[n]

Subroutinen I

Syntax

- **sub** *Name* { *Ausdruck* }
- Aufruf: **&Name**($arg_1, arg_2, ..., arg_n$)
- der Wert der zuletzt berechnet wird ist der Rückgabewert, dieser kann aber auch mit **return** *Wert* festgelegt werden.
- Argumente befinden sich in der Standardvariable @_
  $\Rightarrow$ Zugriff erfolgt durch $_[n]
- **my** Variablenname - private Variable

Subroutinen II

Beispiel

```
sub max {
        my ($a,$b) = @_;
        $a >= $b ? $a : $b;
}
print &max(7,42);
```

Gliederung

Ein- und Ausgabe

Syntax

- Eingabe: <**STDIN**> - nächste Zeile einlesen

Ein- und Ausgabe

Syntax

- Eingabe: <**STDIN**> - nächste Zeile einlesen
- Ausgabe: **print**

Ein- und Ausgabe

Syntax

- Eingabe: <**STDIN**> - nächste Zeile einlesen
- Ausgabe: **print**

Beispiel

```
print "Warte auf Eingabe: ";
chomp($a = <STDIN>); # Newline Zeichen entfernen
print "Sie haben $a eingegeben";
```

Gliederung

Reguläre Ausdrücke I

Syntax

- Suchmuster Syntax → Vortrag Reguläre Ausdrücke

Reguläre Ausdrücke I

Syntax

- Suchmuster Syntax $\rightarrow$ Vortrag Reguläre Ausdrücke
- Bindungsoperator: **String** $=\sim$ **Suchmuster**

Reguläre Ausdrücke I

Syntax

- Suchmuster Syntax $\longrightarrow$ Vortrag Reguläre Ausdrücke
- Bindungsoperator: **String** $=\sim$ **Suchmuster**
- Sondervariablen: **$‘**(String davor), **$&**(String Treffer), **$’**(String danach)

Reguläre Ausdrücke I

Syntax

- Suchmuster Syntax $\longrightarrow$ Vortrag Reguläre Ausdrücke
- Bindungsoperator: **String** $=\sim$ **Suchmuster**
- Sondervariablen: **$‘**(String davor), **$&**(String Treffer), **$’**(String danach)
- Rückwärtsreferenzen: Zugriff durch **$1 ... $n**

Reguläre Ausdrücke I

Syntax

- Suchmuster Syntax → Vortrag Reguläre Ausdrücke
- Bindungsoperator: **String** =~ **Suchmuster**
- Sondervariablen: **$‘**(String davor), **$&**(String Treffer), **$’**(String danach)
- Rückwärtsreferenzen: Zugriff durch **$1 ... $n**
- Ersetzung: **s**/*Suchmuster*/*String*/

Reguläre Ausdrücke I

Syntax

- Suchmuster Syntax $\longrightarrow$ Vortrag Reguläre Ausdrücke
- Bindungsoperator: **String** $=\sim$ **Suchmuster**
- Sondervariablen: **$‘**(String davor), **$&**(String Treffer), **$’**(String danach)
- Rückwärtsreferenzen: Zugriff durch **$1 ... $n**
- Ersetzung: **s**/*Suchmuster*/*String*/
- Optionsmodifier: **g** - global, **i** - Groß- und Kleinschreibung ignorieren

Reguläre Ausdrücke II

Beispiele

```
$_ = "Ein kleiner Regex Test :)";
/\btest\b/ ? print "ein Test" : print "kein Test"; # kein Test
/\btest\b/i ? print "ein Test" : print "kein Test"; # ein Test
s/kleiner/grosser/; # Ein grosser Regex Test :)
s/e/3/; # Ein kl3iner Regex Test :)
s/e/3/gi; # 3in kl3in3r R3g3x T3st :)
$_ =~ /(:){1}(\)){1}/ ? print "$1(" : print "puh..."; # :(
if(=~ /Regex/) {print "$` $' ";} # Ein kleiner Test :)
```

Gliederung

Gliederung

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen
- ... die verschiedenen **Datentypen** kennen

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen
- ... die verschiedenen **Datentypen** kennen
- ... die **Kontrollstrukturen** benutzen können

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen
- ... die verschiedenen **Datentypen** kennen
- ... die **Kontrollstrukturen** benutzen können
- ... **Subroutinen** schreiben können

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen
- ... die verschiedenen **Datentypen** kennen
- ... die **Kontrollstrukturen** benutzen können
- ... **Subroutinen** schreiben können
- ... **Eingaben** vom Terminal lesen können

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen
- ... die verschiedenen **Datentypen** kennen
- ... die **Kontrollstrukturen** benutzen können
- ... **Subroutinen** schreiben können
- ... **Eingaben** vom Terminal lesen können
- ... Daten auf dem Terminal **ausgeben** können

Zusammenfassung

Ihr solltet jetzt...

- ... ein wenig über die **Geschichte** von Perl wissen
- ... die verschiedenen **Datentypen** kennen
- ... die **Kontrollstrukturen** benutzen können
- ... **Subroutinen** schreiben können
- ... **Eingaben** vom Terminal lesen können
- ... Daten auf dem Terminal **ausgeben** können
- ... **Reguläre Ausdrücke** einsetzen können

Wenn es mal Probleme gibt...

Webseiten

- www.perl.com
- perldoc.perl.org - www.perl.org
- de.wikipedia.org/wiki/perl

Wenn es mal Probleme gibt...

Webseiten

- www.perl.com
- perldoc.perl.org - www.perl.org
- de.wikipedia.org/wiki/perl

Bücher

- Einführung in Perl, O'Reilly
- Programming Perl, O'Reilly
- de.wikibooks.org/wiki/Perl-Programmierung

Gliederung

Ausblick

- Module
  $\Rightarrow$ www.cpan.org

Ausblick

- Module
  $\Rightarrow$ www.cpan.org
- Webprogrammierung

Ausblick

- Module
  $\Rightarrow$ www.cpan.org
- Webprogrammierung
- String und Sortierfunktionen

Ausblick

- Module
  $\Rightarrow$ www.cpan.org
- Webprogrammierung
- String und Sortierfunktionen
- Datenbanken

Ausblick

- Module
  $\Rightarrow$ www.cpan.org
- Webprogrammierung
- String und Sortierfunktionen
- Datenbanken
- Dateihandles und Dateitests

Ausblick

- Module
  $\Rightarrow$ www.cpan.org
- Webprogrammierung
- String und Sortierfunktionen
- Datenbanken
- Dateihandles und Dateitests
- Fehlerbehandlung

Ausblick

- Module
  $\Rightarrow$ www.cpan.org
- Webprogrammierung
- String und Sortierfunktionen
- Datenbanken
- Dateihandles und Dateitests
- Fehlerbehandlung
- ...

Danke für die Aufmerksamkeit...

.. we often joke that a camel is a horse designed by a committee, but if you think about it, the camel is pretty well adapted for life in the desert. The camel has evolved to be relatively self-sufficient. On the other hand, the camel has not evolved to smell good. Neither has Perl. (Larry Wall über das Kamel als Perl-Maskottchen)

Quellen

- Randal L. Schwartz & Tom Phoenix, Einführung in Perl, O'Reilly Verlag, 2002
- Larry Wall, Tom Christiansen & Randal L. Schwartz, Programming Perl, O'Reilly Verlag, 1996
- de.wikipedia.org/wiki/perl
- de.wikibooks.org/wiki/Perl-Programmierung
- www.fabiani.net/talks
- perl manpages