Perl – Neues und Interessantes

Eine Nachlese zum Perl-Lehrgang

Wolfgang Friebel
Deutsches Elektronen-Synchrotron

Allgemeines

Bücher

- **Referenz**

  Programmieren mit Perl, L.Wall, T.Christiansen, R.Schwartz, 2. Aufl. 2001 (beschreibt perl 5.6.0) (Kamel Buch)

  Perl - Best Practices (dt.), Damian Conway, 2006

  Das Perl Kochbuch, T.Christiansen, N.Torkington, 2. Aufl. 2004

- **Einsteiger**

  Learning Perl, R.Schwartz, T. Phoenix, brian d foy, 5. Aufl. 2008 (beschreibt perl 5.10) (Lama Buch)

- **Fortgeschrittene**

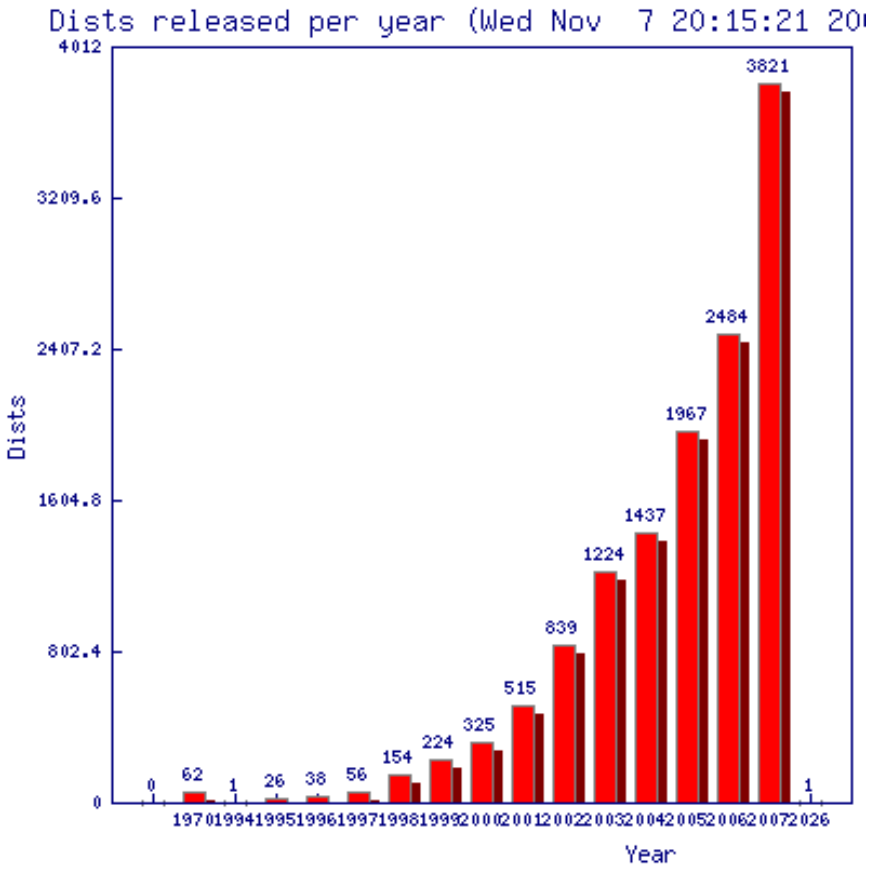
  Advanced Perl Programming (engl.), Simon Cozens, 2. Aufl. 2005

  Intermediate perl (engl.), R.Schwartz, T. Phoenix, b. d foy, 2006

  Mastering Perl (engl.), brian d foy, 2007

Perl Software

- eine stetig wachsende Zahl an perl Modulen findet sich auf CPAN (http, ftp, siehe oben)
  - viele Mirrors, auch in D, z.B. Hamburg, Erlangen
- Sourcecode der meisten in Büchern besprochenen Beispiele ist online verfügbar



Die Perl Installation bei DESY (UNIX)

- /usr/bin/perl ist das mit dem System installierte perl
  - SL3:5.8.0, SL4:5.8.5, SL5:5.8.8, Sol8:5.005, Sol10:5.8.4
- /usr/local/bin/perl ist link auf /opt/products/bin/perl
  - derzeit perl 5.8.2
  - auf SL5 zeigt /usr/local/bin/perl nach /usr/bin/perl !
- nach **`ini perl588`** zeigt perl auf **`/opt/products/perl/5.8.8/bin/perl`**
- nachzulesen auf http://dvinfo.ifh.de/Perl_at_DESY
- es gibt auch perl 5.10 nach **`ini perl510`**

Die Perl Installation bei DESY (Windows)

- Es gibt Netinstall packages für die Versionen 5.8.1.807 und 5.8.8.820 von ActiveState, empfohlen wird 5.8.8
- perl 5.10 gibt es bei Active State zum Download
- Es gibt integrierte Entwicklungsumgebungen (IDE), die (unter UNIX) wenig Produktivitätsgewinn bieten

perl 5.10

- Neue stabile Version seit 18.12.2007
  - 20. Geburtstag von perl
- Viele Verbesserungen und einige Eigenschaften, die auch in perl 6 zu finden sind
- Kann so gebaut werden, dass es unabhängig vom Ort der Installation funktioniert
  - kann ohne Neukompilation kopiert werden
  - ist bei DESY so gebaut (/opt/products/perl/5.10.0)
- neue Konzepte, z.B. say() mit `use feature "say";`

Strawberry perl

- perl 5.10 für Windows
- kommt mit C Compiler und weiteren Tools
- Handhabung (fast) wie unter Linux/UNIX

Perl Module installieren

Perl Module

- Perl kommt mit vielen vorinstallierten Modulen
  - 5:005: ca. 175, 5.6: ca: 220, 5.8: ca 350, 5.10: ca 530
  - nicht alle nützlichen Module sind schon installiert
  - für perl Versionen unter /opt/products/perl wurden Module nachinstalliert, die bei DESY benutzt werden
  - alte perl Versionen bieten weniger
  - Support für Module mit perl 5.005 abgekündigt
    - neue Module müssen nur mit 5.6 oder neuer laufen

Module installieren

- Nicht alle Module können zentral installiert werden
  - daher Module selbst installieren, geht auch

    ohne root bzw. Admin Rechte
  - kleiner Nachteil: Mehraufwand bei Benutzung
- Mehrere Methoden zur Installation (**Linux/UNIX**)
  - **`wget …;tar xzvf …; cd …;perl Makefile.PL \ PREFIX=…;make;make test;make install`**
  - mit **`cpan`** (CPAN) ab perl 5.8.0 installiert
  - mit **`cpanp`** (CPANPLUS) ab perl 5.10.0 installiert

Modulinstallation (Win)

- ohne Admin Rechte
- Mehrere Methoden zur Installation
  - mit **`ppm`** nur in ActiveState perl
  - falls make Ersatz vorhanden ist (**`dmake`**, **`nmake`**)
    Strawberry perl: **`dmake`** in **`C:\strawberry\c\bin`**
    - mit **`cpan`** (CPAN) ab perl 5.8.0 installiert
    - mit **`cpanp`** (CPANPLUS) ab perl 5.10.0 installiert
    - **`wget ...;tar xzvf ...; cd ...;perl Makefile.PL \ PREFIX=...;dmake;dmake test;dmake install`**

Installieren mit cpan

- Bei Erstbenutzung Fragen beantworten
  - für alle Fragen Standard akzeptieren, dann:
  - `o conf makepl_args PREFIX=/tmp/.../perl`
  - erzeugtes Config File ist für jedes OS verschieden!!!
    - Problem kann im AFS mit @sys gelöst werden
- Es gibt OS abhängige Perl Module
  - Problem kann im AFS mit @sys gelöst werden
  - oder `PREFIX=/tmp/.../perl`, dann nur lokal nutzbar
- Nicht gut geeignet für komplexe Aufgaben

Installieren mit cpanp

- Auch bei Erstbenutzung sofort einsatzbereit
- Einfachere Handhabung als **`cpan`**
- Installation ohne root bzw. Admin Rechte auch hier möglich
  - Setzen eines Konfigurationsparameters

```
s conf makemakerflags LIB=/tmp/.../perl/lib
      INSTALLMAN1DIR=/tmp/.../perl/man/man1
      INSTALLMAN3DIR=/tmp/.../perl/man/man3
```

Benutzen eigener Module

- perl muss Pfad kennen, wo Module installiert sind
  - 5 Methoden:
    - Setzen der ENV Variable **`PERL5LIB`**
      - (mit **`export, setenv, set`**)
    - Angabe des Pfades auf Kommandozeile mit **`-I`**
    - Angabe des Pfades auf Kommandozeile mit **`-Mlib=...`**
    - Direktes Verändern von **`@INC`** innerhalb des perl Skripts
    - Verändern von **`@INC`** durch die Anweisung **`use lib ...;`**

Eigene Module (2)

- ENV Variable Methode wirkt auf alle perl Skripte
  - blau: nur für Module mit C Code nötig (nicht auf Win)

```
PERL5LIB=/tmp/.../perl/lib:/tmp/.../perl/lib/i386-li...
```

- Angabe auf der Kommandozeile

```
perl -Mlib=/tmp/.../perl/lib
```
(ohne OS abhängigen Pfad)

- Angabe im Programm

```
use lib qw(/tmp/.../perl/lib);
```

Installierte Module

- Liste aller mit Basisinstallation verfügbarer Module
- ab perl 5.10 funktioniert **`corelist -v 5.8.8`**
- sowie **`corelist File::Temp`**

```
File::Temp was first released with perl 5.006001
```

- Liste **aller** installierten Module
  - viele Methoden, basieren meist auf Annahmen:
    - **`rpm -qa |grep perl`**
    - **`perldoc perllocal`**
  - Korrekte Methode: Suche in **`@INC`** nach **`.pm`** Files:
    - **`/afs/ifh.de/products/scripts/pminst`**

Perl Syntax und Funktionen

Switch

- Es gibt kein Switch Statement (bis perl 5.8.x)

  Nachbildung: siehe z.B perldoc -q switch

  Bei Verwendung von **`last`** (Äquivalent zu **`break`** in C):

```
$_ = <STDIN>;
{
   if (/^a/) {$alpha++; last;}
   if (/^b/) {$beta++;  last;}
   if (/^d/) {$debug=1; last;}
}
```

- Modul Switch existiert, aber anders als 5.10 Syntax

Switch in 5.10

- Ist Feature und muss freigeschaltet werden:

```
use feature "switch";
```

- Syntax mit given, when und default

```
given($_) {
            when (/^abc/) { $abc = 1; }
            when (/^def/) { $def = 1; }
            when (/^xyz/) { $xyz = 1; }
            default { $nothing = 1; }
          }
```

Die Variablen $`, $& und $'

- Werden mit Teilen des Strings gefüllt :
  - $`: Erster Teil des Strings, der nicht matcht
  - $&: Teil des Strings, der Regex matcht
  - $´: Restlicher Teil des Strings
- Variablen nur gefüllt, falls mindestens 1x benutzt
  - Skript ist performanter, falls gar nicht benutzt
- Seit perl 5.6 können @+ und @- benutzt werden
- Kein Geschwindigkeitsverlust, flexibler einsetzbar

Die Variablen `@+` und `@-`

- Diese Felder enthalten erste und letzte Position im String, wo die Patterns gematcht haben
- **`$-[0]`** Startposition des gesamten Matches
- **`$+[0]`** Endposition des gesamten Matches
- **`$-[`*n*`]` `$+[`*n*`]`** sind die entsprechenden Werte für **`$`*n*** d.h. das *n*te Subpattern, was gematcht hat
- Wenn in **`$x`** ein Match war, dann ist **`$``** äquivalent zu

```
substr($x, 0, $-[0])
```

Indirekte Filehandles

- gängige Praxis: `open F, $filename`
- damit ist `F` in Symboltabelle gespeichert und sichtbar
- alter mit `F` verknüpfter offener Stream wird geschlossen
- Ausweg: indirekte Filehandles (seit perl 5.6)
- `open my $FILE, $filename`
- erzeugt anonymes Handle, `$FILE` ist Referenz darauf
  - `$FILE` nicht in Symboltabelle (lexikalische Variable)
  - kann wie normales Filehandle benutzt werden
  - kann an Subroutinen übergeben werden

open mit 3 Parametern

- Öffnen von Files auch mit

```
open HANDLE, mode, filename;
```

- ***mode*** gibt an, wie File zu öffnen ist ('<', '>', '>>', usw.)
- Vorteil: keine Mischung von Name und Art des Öffnens
- erst ab perl 5.6
- Beispiel mit indirektem Handle und 3 Parameter open:

```
open my $F, '<', $file or die "Error in open: $!\n";
```

Input Operationen

- Spezielle Funktion von <>
  - auch Diamantoperator genannt
  - liest aus Files, deren Namen in `@ARGV` stehen
  - danach bzw. stattdessen wird von `STDIN` gelesen
- Spezielles Handle `DATA` liest Text, der nach `__DATA__` oder `__END__` im aktuellen Skript steht

Die readline Funktion

- Term::ReadLine ist für zeilenweise Input zuständig
  - Standard perl liefert Grundfunktionen
  - Erweiterte Funktionen erst mit Zusatzmodulen
    - Kommandozeileneditor, Kommandozeilenhistory, ...
  - Term::ReadLine::Perl oder Term::ReadLine::Gnu sollten unbedingt nachinstalliert werden
  - damit dann Inputstile wie raw, hidden, cooked
  - Ist bei DESY bereits installiert
- Term::ReadKey hat ähnliche Bedeutung

Temporäre Files

- Kann großes Sicherheitsloch bedeuten. Der falsche Code:

```
open (TMP, "/tmp/foo.$$") ...# seen in many places
```

- Um gegen Sicherheitsattacken immun zu sein, sollten Tempfiles
  - nicht in world writable directories angelegt werden
  - keine vorhersagbare Namen haben
  - nicht bereits existieren
- Hacker können einen Hard- oder Symlink dort anlegen, wo gerade ein Tempfile angelegt werden soll. Damit können evtl. wertvolle Daten überschrieben werden.
- Besonders wichtig, wenn Skript mit root Privilegien läuft

Temporäre Files (2)

- Die korrekte Methode ohne File::Temp

```
use POSIX;
do { $name=tmpnam();
} until sysopen(TMP,$name,O_RDWR|O_CREAT|O_EXCL,0600);
# do something with TMP
close TMP;
unlink $name;
```

- Ab perl 5.6 ist File::Temp im Basispaket enthalten

```
use File::Temp "tempfile";
my ($tmp, $filename) = tempfile();
# do something with $tmp
close $tmp;
unlink $filename;
```

min, max und sum

- Keine in perl vordefinierten Funktionen
- Trotzdem vorhanden im Modul List:Util
  - benutzte Funktionen müssen importiert werden

```
use List::Util qw(min max sum);
$ten = max 1..10;
```

- Weitere nützliche Funktionen in diesem Modul

```
use List::Util qw(first);
$p=first{-x "$_/perl"} split/:/,$ENV{PATH};
```

Die eval Funktion

- **`eval`** erwartet perl code als Argument
  - **`eval`** ***`expr;`*** Syntax wird erst zur Laufzeit geprüft, da ***`expr`*** ein dynamisch erzeugter String sein kann
  - **`eval { `*`block`*` }`** Syntaxprüfung beim Kompilieren
- **`eval`** gibt Werte wie bei Funktionsaufrufen zurück
- bei Fehlern während der **`eval`** Ausführung
  - ist der Rückgabewert Null und
  - **`$@`** enthält die Fehlermeldung
  - ansonsten hat **`$@`** den Wert **`undef`**
- Ähnlich wie try und catch in C++

Eval: Ein Beispiel

```
eval "This is not a Perl Program.";
print $@;
# dynamic program generation and execution
$myprog = 'print "3*7 yields ", 3*7, "\n"';
eval $myprog;
eval { 10/$b }; # Division by zero
if ( $@ ) {
  print $@; #or do something else
}
print "... and the Program goes on\n";
```

autodie

- interne Funktionen automatisch auf Fehler testen
- funktioniert sogar mit der system() Funktion
- erst ab perl 5.8 nutzbar
- Siehe auch http://pjf.id.au/blog/?position=543

```
if ($filename) {
    use autodie;  # Turns on all common built-ins by default.
    open(my $fh, '<', $filename);  # This opens or dies
    # Do things with my file.
    close($fh);                    # This closes or dies
}
```

Übergabe von Datenstrukturen

- Bei Subroutinenaufruf können Klammern entfallen

  `mysub($arg1,$arg2);` oder `mysub $arg1,$arg2;`

- Übergabe eines **einzelnen** anonymen Arrays oder Hashes sieht wie Benutzen falscher Klammern aus

```
mysub[$arg1, $arg2]; # passing anon array
mysub{$arg1, $arg2}; # passing anon hash
```

- einzelner anonymer Hash kann als Parameterliste mit benannten Parametern angesehen werden

```
mysub{arg2=>'val2', arg1='val1'};
```

Testen und Debuggen

Debug Printout

- Geht mit `print $data if $debug;`
- Für Datenstrukturen gibt es effizientere Methoden:

```
use Dumpvalue;
  my $dumper = new Dumpvalue;
  print "The variable \$client type ",
  ref($client), " contains:\n";
  $dumper->dumpValue($client);
# we don't really want that:
# $dumper->dumpvars('main'); #all Variables
```

T(I)MTOWTDI

- There is more than one way to do it:

```
use Data::Dumper;

  print "The variable \$client type ",
  ref($client), " contains:\n",
  Dumper($client);
```

- Printing kann durch Variablen beeinflusst werden
- Output ist gültiger Perl Code

Benutzung von Entwicklungstools

- Benutzung von Devel::Trace
  - printet alle ausgeführten Statements

```
perl -d:Trace program
```

  - kann nach Bedarf ein- und ausgeschaltet werden

```
import Devel::Trace 'trace';
trace 'on'; trace off;
```

- Cross reference Listing (kann sehr lang werden!!)
  - `perl -MO=Xref Ex5.pl > Ex5_references.txt`

Profiling

- `perl -d:DProf script` schreibt File **`tmon.out`**

  `dprofpp` erzeugt daraus Profilinformationen

  (Viele Optionen zur Veränderung des Outputs)

  kann auch in einem Schritt erfolgen

  `dprofpp -p script`

- Optimieren nur dort, wo in wenig Code viel Zeit geht
  - Vermeiden von extrem vielen Funktionsaufrufen
  - Übergabe von riesigen Datenmengen vermeiden ...

Benchmarking

- `use Benchmark :hireswallclock; #ab 5.7.3`

  ```
  $count = 100;
  $t = timethis($count, "CODE");
  print timestr($t), "\n";
  ```

  `"CODE"` ist ein Statement, Code Block etc.

  Benchmark Daten sind im Object $t

- Für einfache Probleme

  ```
  use Time::HiRes;      #get higher resolution
  $start = time(); ... #for time calls
  ```

Testen

- Besonders wichtig, wenn Code
  - auf mehreren Plattformen laufen soll
  - Input verarbeitet, dessen Format sich ändern kann
  - zeitkritische bzw. zeitabhängige Teile hat
- Fast alle perl Module kommen mit Testsuite
  - `perl Makefile.PL;make; make test`
- Testsuite ist standardisiert
  - benutzt Module `Test::Simple, Test::More, ...`

Test::Simple und Test::Harness

- Regeln beim Schreiben eigener Testsuiten einhalten
  - Vorteil: weitere Werkzeuge können benutzt werden
  - Test::Harness vereinfacht Beurteilung von Testen
- Output von Testsuiten für Test::Harness
  - Zeilen mit **`ok <Nummer>`** oder **`not ok <Nummer>`**
  - Alle anderen Zeilen beginnen mit #
- Test::Simple stellt nur eine Funktion bereit: ok
  - `use Test::Simple tests => 1;`
  - `ok(1 == 1, "trivial test");`

Test::More

- Viele weitere Testfunktionen, standardisierter Output
  - Skip von Testen, wenn Bedingungen nicht erfüllt
  - Konzept von TODO Testen
  - Standard Testoutput mit **`diag`** statt **`print`**
- Gute Dokumentation mit perldoc
  - Test::Simple, Test::Tutorial, Test::More

perlcritic

- Buch Perl Best Practices von Damian Conway zählt 256 Regeln für guten Code auf
- Modul Perl::Critic ist Source Code Analyzer
  - stellt binary `perlcritic` bereit
  - kennt viele der Regeln aus dem Buch
  - teilt Regeln nach Wichtigkeit ein
  - lässt andere/eigene Regeln zu
  - hilft, Coderegeln umzusetzen und testet Einhaltung
  - Bei DESY nicht installiert

Module, Klassen und Objekte

Erzeugung von Modulen

- Kochrezept (für Module/Test.pm):

  ```
  h2xs -X -n Module::Test
  ```

  - erzeugt Skelettdateien, die man modifizieren muß
  - Doku sollte im perlpod Format erstellt werden
  - Installation des Modules im Standard Perl mit

    ```
    perl Makefile.PL; make; make test;
    make install
    ```

- Weiterführende Literatur u.a. in http://world.std.com/~swmcd/steven/perl/ (2002)

Vererbungshierarchie

- Vererbung ist Suche nach Methode in **`@ISA`**
  - Suche zuerst nach oben im Vererbungbaum
  - Danach Suche von links nach rechts in **`@ISA`**
- Alle Methoden erben von **`UNIVERSAL`**
- Wird Methode nicht gefunden, wird nach gleichen Regeln nach Methode **`AUTOLOAD`** gesucht
- Ab perl5.10 Beeinflussung der Vererbungsregeln möglich

Die Klasse SUPER

- SUPER ist ein Pseudo-Package
- Idee: Methode in übergeordneter Klasse erledigt Teilaufgabe (Delegation), Rest in aktueller Methode
- SUPER::test sucht in Eltern der aktuellen Klasse
  - gesamte Vererbungshierarchie wird durchsucht
  - erste gefundene Methode test wird verwendet

Die Methode Autoload

- **AUTOLOAD** wird gerufen, falls Methode nicht existiert
- In **$AUTOLOAD** steht Name der fehlenden Methode
  - **$AUTOLOAD** ist Variable des aktuellen Packages
  - **AUTOLOAD** erhält wie üblich Objektref übergeben
- Damit vor allem viele get/set Methoden realisierbar
- Nachteil: es wird jedesmal nach Methode gesucht
  - **AUTOLOAD** wird für **jede** undefinierte Methode gerufen

Closures

- Zugriff auf lokale Variablen mit begrenztem Gültigkeitsbereich (scope) per Subroutine

```
{ my $count=0; #access only to end of block
  sub incr { $count++; print "Count is now $count\n"}
  sub decr { $count--; print "Count is now $count\n"}
}
incr;incr;decr; print "No access to count: $count\n";
```

- Außerhalb des Blocks ist Veränderung von `$count` mit `incr` und `decr` möglich, nicht aber Zuweisung
- Variable lebt weiter, da Referenzcount != 0
- Keine Chance, `$count` direkt zu verändern

Oracle mit DBI

- Windows: SQLPlus installieren (Netinstall)
- UNIX: ORACLE_HOME muss gesetzt werden:

  `$ENV{ORACLE_HOME}='/opt/products/oracle-client/10.2g'`

  - gilt nur ab SL3 und neuer
    und `/opt/products/perl/5.8.8/bin/perl`
  - für ältere Versionen `5.8.8 -> 5.8.2, 10.2g -> 9.2.0`
- Windows User dürfen diese Variable **nicht** setzen
- als DB Name `'dbi:Oracle:desy_db'` verwenden!

Proxy Server Benutzung

- Proxy Server wird von Kommandozeile gestartet
  - `dbiproxy --localport portnumber`

  (für UNIX in /opt/products/perl/5.8.8, für Win im PATH)
  - ENV Vars auf Proxy Host setzen ! (`ORACLE_HOME`)
- Standalone Skript wird zum Proxy Client durch
  - Ändern des DB Namens im connect Aufruf oder
  - Setzen des DB Namens in der ENV Variable `DBI_AUTOPROXY`, dann **keine Änderung** des Skripts!

Systeminformationen (Windows)

- Modul Win32 im Standard perl (ab 5.8.4)
  - stellt Schnittstelle zu vielen Windows Funktionen dar
  - geht weit über reine Informationsgewinnung

```
use Win32;
print "Admin\n" if Win32::IsAdminUser();
```

- Direkte Nutzung von Funktionen aus DLL's
  - mit Modul Win32::API
- viele weitere Module im CPAN vorhanden

Weitere Informationen

- Schulungsmaterial (Folien, Beispiele, Software inclusive CPAN Stand 05/08) auf DVD vorhanden
- Folien des Perl Lehrgangs sind auf

  http://www-zeuthen.desy.de/~friebel/perl/perl2008

Fragen ?