Perl Einführung

„Im Bestreben das Programmieren zu einer zuverlässigen Disziplin zu machen, haben die Informatiker weitgehend erreicht, dass das Programmieren langweilig geworden ist“

`Larry Wall`

Hagen Paul Pfeifer

https://www.jauu.net ● hagen@jauu.net

17. April 2005

Überblick

- Sprachumfeld
- Hello World
- Datentypen
- Kontext
- Reguläre Ausdrücke
- IO (minimal)
- CPAN

Sprachumfeld

- Larry Wall als der Initiator und Wegweiser (zumindest bis Perl 6)
- Erwachsen aus: sh, sed, awk und c
- Wirkt und ist anders; mal chaotisch, mal clever aber immer irgendwie Aufregend!

Überlegungen

Wo setze ich Perl ein?

- Stringmanipulation (Filter aller Art, XML, WEB)!
- Probleme welcher durch Clevernes schnell gelöst werden können
- Wrapper für Binarys
- Netzwerktools (automatisierte pings, pcap, rawsocket, . . .)
- als besseren Shellscripting Ersatz
- quick and dirty h4Ck$
- zum Spaß haben, da wo schöne CPAN Module erhältlich sind („auf den Schultern von Riesen“)

Vorteile

Wo sind die Stärken von Perl

- keine strikte Deklarationen von Variablen (u_int, int oder short?)
- Portabel (Perl ist von A (wie Amiga) bis Z (wie z-series portiert)
- CPAN als gigantische Quelle von Problemlösungen
- Kein Probleme mit Dangling Pointer'n, Speicherbereichen, etc
- Komplexe Datenstrukturen sind einfach zu implementieren
- . . .

Ungeeignete Einsatzszenarien

Bei welchen Programmieraufgaben sollte ich auf Perl verzichten?

- mittlere bis größere Projekte
- bei Code welcher performancekritisch ist (aber Achtung!)
- Code welcher sehr Systemnah ist
- Code welcher als `binary only` vermarktet werden soll (furchtbar!)
- eure Ideen hier!

Vorurteile

Als diese in den meisten Fällen als solche zu bewerten: bei genaueren Hinschauen entpuppen sich viele als Feature!

- line noisy
- schwer zu verstehen
- Verhalten abhängig vom Kontext (scheinbar nicht konsistent, keine Angst: es bleibt deterministisch! :-)
- Reichhaltigkeit der Sprache
- . . .

Nun wollen wir aber unser Rennauto besteigen
und ein paar Schalter betätigen!

Hello World

- `print "howdy world\n";`
  - Ausführung: `perl programmname`

oder:

- `#!/usr/bin/perl`
  `# möglichst sinnvoller Kommentar ...`
  `print "I am alive\n";`
  - Ausführung: `chmod +x name; ./name`

oder auch:

- `perl -e 'print "Howdy World";'`

Hello World

Motto: „Einfache Dinge sollen einfach sein, schwere Dinge sollten möglich sein.“

- She Bang: teilt OS mit welcher Interpreter gestartet werden soll
- Kommentare werden durch # eingeleitet
- Funktionsargumente müssen nicht geklammert werden

Datentypen (man perldata)

Grundsätzlich

- Skalare ($preis)
- Arrays (@liste)
- Hashes (%optionen)

$ wie Skalar

Bestehen nur aus **einem** Wert
Larry Wall würde eventuell auf die Singularität hinweisen . . .

- Kann ein String (begrenzt durch die Größe des Arbeitsspeichers), Zahl oder Referenz auf `etwas` sein.
- String:
  `$name = "hagen";`
- Numerische Wert:
  `$studiengebuehr = "1500";`
- Referenz:
  `$pref = sub {printf "howdy!";}; $arrref = [1,2,3]`

@ wie Array

Geordnete Liste von von Skalaren beginnend bei Index 0 welche homogen oder heterogene Elemente enthalten kann.

- `@days = gw(Montag Dienstag Mitwoch Donnerstag Freitag Samstag Sonntag);`
- Zugriff auf 4. Element:`$days[3];`
- Auf Anfang, Ende: `shift, unshift, push, pop`
- Letzter Index: `$days[$#days];` oder `$days[-1];`
- Slice: `@days[3,4,5];` auch anders: `$days[2,3] = (Mittwoch, Donnerstag);`
- Generierte Liste: `$buchstaben = ('a' .. 'z')`
- Sortierte Liste: `sort { $a <=> $b } $zahlen;`
- Liste erzeugen: `@felder = split(/:/, $passwd);`
- Umkehrung: `$passwd = join(":", @felder);`

% wie Hash

Ungeordnete Liste von Skalaren welche über einen Namen indiziert/angesprochen werden

- Init: `%xyz = (key1 => value1, key2 => value2)`
- Zugriff: `$xyz{'key1'} # value1`
- Zugriff ueber Schlüssel: `foreach my $keys (keys %xyz) {print $xyz{$keys}, "\n"};`
- Oder paarweise: `($key, $value) = each %hash;`
- Nur Werte? `$val = values %hash;` Kein Problem!
- Hash löschen: `undef %hash;`

Kontrollfluß

- if elsif else:

```
if (23 > 5) {
    print "SNAFU!\n";
} elsif (5 > 23) {
    print "Hagbard was here!\n";
} else {
    print "Akte X\n";
}
```

- bedingte Auswertung perlisch:

```
print "r00t!\n" if $< == 0;
```

- foreach (jedes Element in einem Array)

```
foreach $file (sort </usr/include/*.h>) {
    print "$file: ", -s $file, "\n";
}
```

- Klassisch: for

```
for($i = 0; $i < 10; ++i) {
    doSomethingRealyNeccesaryWithI($i);
}
```

- while

```
while ($i) {
    $i--;
}
```

- `next; last`

Funktionen

- `sub nothing { print "still living!\n"; }`
- Argumente werden in einer flachen Liste übergeben
- @_ ist repräsentiert den Inhalt

Strings, Quotation und Interpolation

- `$var = "Vimversion: $version"; # doppelt gequotet: interpolation`
- `$var = 'Vimversion: $$'; # einfach gequotet: keine interpolation`
- `$var = 'ps -xaf'; # backticks -> ausführen und Ausgabe speichern`
- Vergleichsoperatoren: `eq, ne, lt, gt, le, ge, cmp`
  - `$print "String's identisch!" if $str1 eq $str2;`
- Länge eines Strings
  - `$len = length($string)`
- Stringkonkatenation
  - `$foobar = $foo . $bar;`
- `reverse` dreht String um
  - `reverse "otto"; # schlechtes Beispiel ;-(`
- `substr` (substitute string)

- ```
  $name = "UNixFreaksandUserGroup";
  substr($name, 4, 5) = "Friends"; # substr als lvalue!
  ```

- Wiederholungen
  - `perl -e 'print "a" x 100'`
- In/Dekrementoperatoren
  - ```
    $name = "Holla";
    $name++;
    print $name # Hollb
    ```

Zahlen

- `$zahl = 23;`
- Vergleichsoperatoren: `<, <=, >, >=, ==, !=, <=>`
- Arithmetik: `\`, -, *, /,
- Arithmetik und Zuweisung `\=`, -=, *=+ ...
- Bittwidling: `|, ^, <<, >>, ~`
- Ausnahme Abfangen: `eval{23 / $i;}`
- `Math::BigInt`
- Implizierte Konvertierung (String -> Zahl): `$string \ 0;+` (atof)

Kontexte

- `Perl wertet Ausdrücke (und Operatoren) in Abhängigkeit des Kontextes aus`
- Dies ist **eine** Ursache für das magische Verhalten von Perl: „Perl denkt zu wissen was man will . . . (zumindest denken wir das!) “

Kontexte

Grundsätzlich

- Skalarkontext
- Listenkontext
- Boolscher Kontext
- void Kontext
- Stringkontext (eventül; Ansichtssache)

Kontexte

Beispiel:

- Skalarkontext und Listenkontext

```
$line = <HANDLE>; @all_lines = <HANDLE>;
```

- Boolescher Kontext: eigentlich skalarer Kontext. Alles außer 0 und "0ßind wahr.
- void Kontext: es wird kein Rückgabewert erwartet

Sprachschatz

War das alles? Nein!

```
"chomp", "chop", "chr", "crypt", "hex", "index", "lc", "lcfirst", "length",
"oct", "ord", "pack", "q/STRING/", "qq/STRING/", "reverse", "rindex", "sprintf",
"substr", "tr///", "uc", "ucfirst", "y///" "m//", "pos", "quotemeta", "s///",
"split", "study", "qr//" "abs", "atan2", "cos", "exp", "hex", "int", "log",
"oct", "rand", "sin", "sqrt", "srand" "pop", "push", "shift", "splice",
"unshift"  "delete", "each", "exists", "keys", "values", "binmode", "close",
"closedir", "dbmclose", "dbmopen", "die", "eof", "fileno", "flock", "format",
"getc", "print", "printf", "read", "read- dir", "rewinddir", "seek", "seekdir",
"select", "syscall", "sysread", "sysseek", "syswrite", "tell", "telldir",
"truncate", "warn", "write", "pack", "read", "syscall", "sysread", "syswrite",
"unpack", "vec", "-X", "chdir", "chmod", "chown", "chroot", "fcntl", "glob",
"ioctl", "link", "lstat", "mkdir", "open", "opendir", "readlink", "rename",
"rmdir", "stat", "symlink", "sysopen", "umask", "unlink", "utime", "alarm",
"exec", "fork", "getpgrp", "getppid", "getpriority", "kill", "pipe",
"qx/STRING/", "setpgrp", "setpriority", "sleep", "system", "times", "wait",
"waitpid",  "accept", "bind", "connect", "getpeername", "getsockname",
"getsockopt", "listen", "recv", "send", "setsockopt", "shutdown", "socket",
"socketpair", "endprotoent", "endservent", "gethostbyaddr", "gethostbyname",
"gethostent", "getnetbyaddr", "getnetbyname", "getnetent", "getprotobyname",
```

```
"getprotobynumber", "getprotoent", "getservbyname", "getservbyport",
"getservent", "sethostent", "setnetent", "setprotoent", "setservent",  "abs",
"bless", "chomp", "chr", "exists", "formline", "glob", "import", "lc",
"lcfirst", "map", "my", "no", "our", "prototype", "qx", "qw", "readline",
"readpipe", "ref", 0"sub*", "sysopen", "tie", "tied", "uc", "ucfirst", "untie",
"use", "caller", "continue", "die", "do", "dump", "eval", "exit", "goto",
"last", "next", "redo", "return", "sub", "wantarray",
```

- grosser Sprachschatz (man perlfunc)
- endlich Platformunabhängig programmieren (fork() auf M$ Wintendo)

Geltungsbereiche

- lokale Variablen (`my`)
- globale Variablen (default)
- dynamischer Geltungsbereich (`$_`)
- Achtung: `local` erzeugt einen neunen Geltungsbereich fr eine Variable; keine neue Variable!

Reguläre Ausdrcke (man (perlrequick | perlretut)

- Eng mit Perl verbunden (Grundstein des Sprachkerns)
- begründet teilweise die Mächtigkeit der Sprache
- Verleitet teilweise jedes Problem als RegEx zu betrachten (Nachdenken!)

Reguläre Ausdrcke - Grundsätzlich

- Sucht in einem String (Ausdruck) nach einem Muster ($_)
- ```
  $string = " one nuclear bomb can ruin your whole day.";
  print "BOMB\n" if $string =~ /bomb/i;
  ```

- Fast jedes Zeichen steht für sich selbst (ausser:`{()+.*$?^|\\`)
- = ; Negation: !
  ```

Reguläre Ausdrcke - Spezialvariablen

- `.` steht für jedes Zeichen (bis auf Newline (naja fast))
- Zeilenanker
  - ^ Anfang des des Ziels (mitunter Newlinezeichen)
  - $ Ende des Ziels (bis vor Newline (wieder Mehrzeilenmodus beachten!)
- `( )` Gruppieren Muster
- `[]` Klasse von Zeichen: `[a-f0-9]`
- `|` Alternativen
- `(??{code})` fürt code aus und interpretiert in als Regex
- Quantifier
  - + erkennt Muster ein oder mehrmals
  - ? nicht oder einmal
  - * nicht, ein oder mehrmals
  - `{n,m}` n: minimal; m:maximal

- `\1` verweist auf Treffer
- Zeichenklassen
  - `\w` matcht auf alphanumerische Zeichen und Unterstrich
  - `\W` matcht NICHT auf alphanumerische Zeichen und Unterstrich
  - `\s` Whitspaces (Reziproke: `\S`)
  - `\d` Digit
  - `\b` Wortgrenze
- RegEx Modifikatoren
  - `s` betrachtet String einzeilig
  - `g` erkennt sooft wie möglich (global)
  - `i` ignoriert Gross Kleinschreibung

Reguläre Ausdrcke - Substitution

- ```
  $string = "Java programmieren macht Spass";
  $string =~ s/Java/Perl/;
  ```
- Perl Idiom(?):

  ```
  (my $newstring = $oldstring) ~= s/java/perl/;
  ```
- sRegEx Modifikatoren
  - e evaluiert Ersetzung als Perl Ausdruck;
  ```

IO (man perlfunc)

- `<>, <ARGV>` liest aus Dateien via argv[0 ..] oder von STDIN
  Cat in 7 Zeichen: `print while (<>);`
- `print STDOUT "still alive\n";`
- Eingabe: `$input = <STDIN>;`
- `open(FHI, "<  in") or die;`
- `open(FHO, ">  in") or die;`
- `open(FHA, ">> in") or die;`
- `open(WEX, "| nc 10.0.0.1 2345") or die;`
- `open(REX, "ps -xaf|") or die;`
- `open(REX, "ps -xaf|") or die;`
- Dateitests: `-r, -w, -e, -x, -S, -M ...`
- `unlink, rename, opendir, readdir, mkdir, sysmlink, stat, ...`
- Fileglobs: `@texfiles = <*.tex>;`

Beispiele

- Hunger?

Beispiel - Wieviel neue Post ist in meinem IMAP Postfach?

```
use strict;
use IO::Socket::SSL;
use Mail::IMAPClient;
use Term::ReadKey;
use Term::ANSIColor;

my $username = 'pfeifer';
my $server = '0xdef.net';

print "*** unseen emails for user $username for imap-server $server ***\n\n";

print "Password: "; ReadMode('noecho'); my $password = ReadLine(0);
print "\n"; chomp $password; ReadMode('normal');

my $ssl  = IO::Socket::SSL->new("0xdef.net:imaps") or die $@;
my $imap = <$ssl>;
$imap = Mail::IMAPClient->new(
      Socket=>$ssl,
      User=>$username,
      Password=>$password,
   Peek => 1
      );
```

```
unless($imap) { die "Couldn’t log in - $@\n"; }
$imap->State($imap->Connected());
$imap->login() or die ’login failed’;

$imap->select(’INBOX’);

my @mails = ($imap->unseen);
foreach my $id (@mails) {
print "\n\tSubject: ";
   print color("red"), $imap->parse_headers($id,"Subject")->{"Subject"}->[0];
   print color("reset") . "\n";
print "\tFrom:    ";
   print color("magenta"), $imap->parse_headers($id,"From")->{"From"}->[0];
   print color("reset") . "\n";
print "\tDate:    ";
   print color("green"), $imap->parse_headers($id,"Date")->{"Date"}->[0];
   print color("reset"). "\n";
}
```

Beispiel - Vorkommen Wörter zählen

```
$fl=shift;
open(INPUT,"<$fl") || die "NO INPUTFILE DUDE";
%words=();
while (<INPUT>) {
    while (/(\w['\w-]*)/g) { $words{lc $1}++ }
}
foreach $word(sort{$words{$b} <=> $words{$a}} keys %words) {
    printf "%5d %s\n", $words{$word}, $word;
}
```

Beispiel - Heartbeat in drei Zeilen

```
use Net::Ping;

$p = Net::Ping->new();
print "$host is alive.\n" if $p->ping($host);
$p->close();
```

(aus Net::Ping)

Beispiel - CIA Factbook Visualisieren

```
use strict; use Pg;
use Chart::Pie;

my $databasename = "factbook";

my $conn = Pg::connectdb("dbname=$databasename");
my $p    = Chart::Pie->new(1900, 1700);

unless ($conn->status == PGRES_CONNECTION_OK) {
   die "Can't connect to database $databasename: " . $conn->errorMessage . "\n";
}

my $dbname = $conn->db;
my $dbuser = $conn->user || $ENV{'USER'};
my $dbhost = $conn->host || "localhost";
print "Connected with $dbname as user $dbuser with $dbhost\n";

my $result = $conn->exec("select name,population from wfb.countries where countries.population > 50000000");
next if ($result->resultStatus == PGRES_COMMAND_OK);

my @names; my @population;
for(my $cnt = 0; $cnt < ($result->ntuples); ++$cnt) {
```

```
   push @names,      ($result->getvalue($cnt, 0) . " ");
   push @population, ($result->getvalue($cnt, 1) / 1000000);
}
$p->add_dataset(@names);
$p->add_dataset(@population);
my %opt = ( "title"        => "World Population",
            "label_values" => "both",
            "legend"       => "none",
            "text_space"   => 10,
            "png_border"   => 1,
            "colors"       => {
                    "x_label"   => "red",
                    "dataset0"  => "blue"
                    }
            );
$p->set(%opt);
$p->png("population.png");
```

Beispiel - Primzahlen

```
use strict;
use Crypt::Random qw( makerandom makerandom_itv);
use Math::BigInt;

my $bit_size = shift || 512;
my $primenumber;
do {
        $primenumber = Math::BigInt->new(makerandom( Size => $bit_size, Strenth => 0 ));
} until ( miller_rabin($primenumber) );

print "\n" . $primenumber . "\n";

sub miller_rabin {
        print ".";
        my $ANZEIGE = 0;
        my $primenumber = Math::BigInt->new($_[0]);
        my $b = 0;
        my $m = Math::BigInt->new($primenumber -1);
        my $test_loops = 5;

        return 0 if $primenumber->is_even();
        while( 0 ==  $m->copy()->band('1')  ) {
```

```
                $m->brsft('1 '); # right shift
                $b++;
               print "+";
        }
LOOP:
        for ( 0 .. $test_loops ) {
                print "X";
                my $rand = Math::BigInt->new('0');
                do {
                        $rand = makerandom_itv( Strenth => 0, Upper => $primenumber->copy()->bdec() );
                        $rand++;
                } while ( Math::BigInt::bgcd($rand, $primenumber) > 1 );

                my $j = 0;
                my $z = Math::BigInt->new($rand);
                $z->bmodpow($m, $primenumber);
                next LOOP if $z == 1;
                while ( $b > ++$j ) {
                        return 0 if $z == 1;
                        next LOOP if $primenumber->copy()->bdec() == $z;
                        $z = ($z * $z) % $primenumber;
                }
                if ( $primenumber->copy()->bdec() != $z ) {
                        return 0;
                }
        }
```

```
        return 1;
}
```

Beispiel - Foo Fork()

```
#!/usr/bin/perl
my @except = ("luckyluke.foo.fh-furtwangen.de", "141.28.64.60",
              "141.28.65.50", "father.foo.fh-furtwangen.de",
              "samson.foo.fh-furtwangen.de", "141.28.64.150");
my @hosts = ‘dig \@141.28.2.19 foo.fh-furtwangen.de AXFR‘;
my %validhost = ();

# test for ssh connectivity
foreach my $host (@hosts) {
   $host =~ /(\w+\.foo\.fh-furtwangen\.de).*\b(([0-2]?\d{1,2}\.){3}[0-2]?\d{1,2})\b/;
   next unless defined $1;
   next unless defined $2;
   my $name = $1; my $ip = $2;
   foreach my $exn (@except) {
      if ($name eq $exn or $ip eq $exn) {
         goto NEWPROPE;
      }
   }
   open(NCHND, "echo coredump | nc -v -w 1 $2 22 2>&1 |");
   while(<NCHND>) {
      if (/open/) {
         $validhost{$ip} = $name;
```

```
         print "TAKE: hostname: $name\tIp: $ip\n";
         next; }
   }
NEWPROPE:
}

# ... and do the dirty
foreach my $target (keys %validhost) {
   eval {
      local $SIG{ALRM} = sub { goto NEXT; };
      alarm 2;
      system("scp ./pid.c $target:~ 2>/dev/null");
      alarm 100;
      system("ssh $target \"gcc ~/pid.c\" 2>/dev/null");
      alarm 100;
      my $os = `ssh $target \"uname -a\"`; chomp $os;
      alarm 100;
      my $ret = `ssh $target \"~/a.out\"`; chomp $ret;
      alarm 100;
      system("ssh $target \"rm -f ~/pid.c ~/a.out\" 2>/dev/null");
      print "$validhost{$target} ($target)\t=> $ret [$os]\n";
      NEXT:
   };
}
```

Beispiel - XML

- Code zeigen (doch größer geworden)!

Literatur / Kontakt / Freunde ?

- Programmieren mit Perl
- Perl Kochbuch
- Perl für Systemadministration
- perl.org
- lokale Perl Gruppe (Perl Mongers)
- . . .