Geschichte

Larry Wall schrieb ein Konfigurationsmanagementsystem und benötigte dazu ein Berichtwerkzeug, das mehrere Dateien gleichzeitig öffnen konnte. Awk konnte es damals nicht, daher schrieb er eine Sprache namens Pearl. Da er zu faul war, beim Aufruf jedesmal 5 Buchstaben einzugeben, strich er das a.
Perl steht für „Practical Extraction and Report Language“.

Dokumentation

- `man perl`

  Die Manpage ist in viele Kapitel geteilt, die alle mit `perl` beginnen, zum Beispiel `man perldata` für Datenstrukturen.

- Siever et al.: Perl in a nutshell

  Eine Referenz des Programms und der wichtigsten Module.

- Wall et al.: Programming Perl

  Eine Beschreibung des Programms und der dahinter stehenden Philosophie.

Datentypen

Verschiedene Datentypen sind syntaktisch durch Präfixe auseinanderzuhalten.
Perl unterscheidet nicht zwischen Integern, Fließkommazahlen und Strings (!). Der Oberbegriff ist „Scalar“. Ein Scalar wird durch das Dollarzeichen kekennzeichnet: `$a=1; $b="Eins";`.
Geordnete Listen sind durch den Klammeraffen dargestellt: `@lottozahlen = (1, 24, 27, 32, 38, 43)`.
Assoziative Hashes sind durch das Prozentzeichen dargestellt:

```
%ziehung = (lottozahlen => [1, 24, 27, 32, 38, 43],
            zusatzzahl => 18,
            superzahl => 6);
```

Alles andere funktioniert mit Zeigern. Zum Beispiel ist `[1, 24, 27, 32, 38, 43]` ein Zeiger auf `(1, 24, 27, 32, 38, 43)`.

Dateien ansprechen

Syntax: `open HANDLE, string`
Ein Dateihandle ist ein Stichwort („Bareword“) in Großbuchstaben. Der String ist ein Dateiname oder ein Pipe.

- öffne zum Lesen: `open HANDLE, "Datei"`
- öffne zum Schreiben: `open HANDLE, ">Datei"`
- öffne als Pipe: `open HANDLE, "Datei |"` oder `open HANDLE "| Datei"`

Die Datei wird unter `<HANDLE>` zeilenweise zur Verfügung.

Gleich angewöhnen:

```
$dateiname = "/etc/passwd";
open USERS, $dateiname or die "Fehler beim Öffnen von $dateiname: $!";
```

Kontrollstrukturen

if Bedingung

```
my $a = "0";
my $b = 0;
if ($a eq $b) {
    print "Die Zeichenketten $a und $b sind gleich."
}
```

for Zähler

```
for ($i=0;$i<10;$i++) {
    print "Das Quadrat von $i ist", $i*$i, ".\n";
}
```

while Solange Bedingung erfüllt

```
my $k = 56;
while ($k>1) {
    print $k, "\n";
    if ($k/2>int($k/2)) {
        $k = $k/2;
    }
    else {
        $k = 3*$k+1;
    }
}
```

Mehr Kontrollstrukturen

foreach Iterator

```
foreach $i(1:9) {
    print "Das Quadrat von $i ist", $i*$i, ".\n";
}
```

oder auch

```
open USERS, "/etc/passwd" or die "Fehler beim Öffnen von /etc/passwd: $!";

foreach $zeile(<USERS>) {
    my @eintraege = split ":", $zeile;
    my $username = $eintraege[0];
    my $realname = $eintraege[4];
    print "User $username heißt in echt $realname\n";
}
```

Reguläre Ausdrücke (man perlre)

Reguläre Ausdrücke sind Suchmuster. Sie enthalten

- Zeichen (zum Beispiel `12:34`)
- allgemeine Zeichen (zum Beispiel `\d\d:\d\d`)
- Quantifizierer (zum Beispiel `\d+:\d+`)
- Positionszeichen (zum Beispiel `\b\d+:\d+\b`)

Der Matching-Operator

Dieser Operator wendet auf einen String eine Such- und eventuell eine Ersetzfunktion an. Syntax: `$string =~ m/regex/mod` oder `$string =~ s/regex/ersatz/mod`.
Was gibt der Operator zurück? Den Erfolg der Operation, wobei es verschiedene Arten von Erfolg gibt:

- Bei einer Suche (`m//`) wird der passende String zurückgegeben.
- Bei einer Ersetzung (`$anzahl = $string =~ s///`) wird die Anzahl der Ersetzungen zurückgegeben.
- Im Listenkontext werden die in Klammern gesetzten Ausdrücke zurückgegeben: `($key, $value) = $string =~ m/(\w+)=(\w*)/`.

Statt dem letzten geht auch:

```
if ($string =~ m/(\w+)=(\w*)/) {
    ($key, $value) = ($1, $2);
}
```

Modifikatoren

g ,,global“: Finde alle Vorkommen

i ,,insensitive“: Groß- und Kleinschreibung ignorieren

m ,,multiple“: Behandle String als mehrere Zeilen

s ,,single“: Behandle String wie eine einzige Zeile

o ,,once“: Kompiliere String nur einmal

x ,,extended“: Erweiterte Ausdrücke

Maulfaul

- Das m kann man auch weglassen.
- Wenn man den zu durchsuchenden String weglässt, dann ist `$_` gemeint.
- Dann muss man auch den Operator `=~` weglassen.

Häufig findet man daher:

```
while (<EINGABE>) {
   if (/<\w+>/) {
       print "Zeile enthält Tag ohne Schlüsselwörter\n";
   }
}
```

Der Debugger (man perldebug)

Aufruf mit `perl -d`.

n Einen Schritt weiter, Subroutinen werden übersprungen.

s Einen Schritt weiter, Subroutinen werden mitgenommen.

r Subroutine bis zum Ende, gib Rückgabewert an.

b Zeilennummer Setze Stoppunkt an Zeilennummer.

x Ausdruck Evaluiere Ausdruck und zeige Ergebnis an.

c Weiter bis zum nächsten Stoppunkt oder Programmende.

R Starte Programm neu.

q Verlasse Debugger.

Perl wie sed oder awk nutzen (man perlrun)

- Die Kommandozeilenoption `-e zeile` übergibt die Zeile als Programm, keinen Dateinamen.
- Die Kommandozeilenoption `-i` ändert Dateien als Argumente und überschreibt das Ergebnis mit dem Original.
- Die Kommandozeilenoption `-p` lässt Perl annehmen, dass das Programm auf jede Zeile der Eingabe angewandt werden soll, und dass jede Zeile (möglicherweise verändert) auch wieder ausgegeben wird.
- Die Kommandozeilenoption `-n` lässt Perl annehmen, dass das Programm auf jede Zeile der Eingabe angewandt werden soll. Zeilen werden nicht explizit ausgegeben.

Also: `perl -pie 's/<([^>]+)>/{$1}/gs' index.html` ersetzt in der Datei `index.html` alle Spitzklammerpaare durch geschweifte Klammern.

Beispiel: aktueller Fahrplan

Das folgende Skript ruft einen aktuellen Fahrplan der Bahn auf und fragt die Abfahrts- und ggf. Verspätungszeiten ab.

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

use strict;

my $Browser = "w3m -dump";
```

```
# . ist Verkettungsoperator
# Adresse ist Essen Hauptbahnhof

my $TafelURL
  ="http://bahnhofstafel.db-ris.de/bht/internet/tafel.html?" .
   "AnAb=AB&m=dn&ident=10.013729201.1036991406&" .
   "seqnr=1&L=&bhf=8000098";

#  Zugnummer; eindeutig außer bei S-Bahnen
my $zug = "RE 10005";

#  Öffne Pipe
open ABFAHRT, "$Browser '$TafelURL' 2>/dev/null |";

#  w3m stellt in Tabellenform dar (im Gegensatz zu Lynx :-P)
#  die Mitteilung über Abfahrt steht in der Spalte, die
#  mit "Aktuelles" überschrieben ist. Also Spaltenposition
#  merken, wenn die erste Zeile eingelesen wird.

my $spalte = 0;
while (<ABFAHRT>) {
    if (/^Zeit/) {
        $spalte = index($_, "Aktuelles");
    }
    if (/$zug/) {
        print "Meldung über Zug $zug: ", substr($_, $spalte);
    }
}
```

Vorsichtsmaßnahmen

- Nutze als Aufruf `#!/usr/bin/perl -w` (für „warnings“).
- Nutze bei größeren Projekten `use strict vars;`