Eine Einführung in Perl

entstanden im Proseminar Unix-Tools
Technische Universität München

von Sebastian Vogl

Inhaltsverzeichnis

1 Geschichte, Ursprung und Merkmale

1.1 Wie ist Perl enstanden ?

Im Jahre 1987 war der Linguist und Systemadministrator Larry Wall an der Entwicklung eines sicheren Netzwerks für die NSA beteiligt. Hierbei war es seine Aufgabe Administrations-Tools für die neue Software zu entwickeln. So erhielt er den Auftrag ein Fehlersuch-Tool für verstreute Logdateien zu entwerfen. Da Larry die vorhandenen Werkzeuge zu umständlich erschienen und er nichts finden konnten das genau seinen Bedürfnissen entsprach entschied er sich für die Lösung des Problems ein neues Mehrzweck-Tool zu programmieren. Im Dezember 1987 publizierte er dieses unter dem Namen Perl 1.0 im Usenet. Damit nahm die Geschichte ihren Lauf. Perl liegt heute in der Version 5.8.7 vor.

1.2 Der Ursprung von Perl

1.2.1 Geistig

Es liegt auf der Hand, dass Perl sehr stark durch seinen Entwickler geprägt wurde, der Linguistik studierte, als Systemadministrator und Programmierer arbeitete, humorvoll und streng gläubig ist. Somit orientiert sich Perl stark an den menschlichen Sprachgewohnheiten und soll in Larrys Augen möglichst vielen Menschen als „bescheidener Diener“ bei der Arbeit helfen. Daher ist es wichtig zu verstehen wie Larry denkt, um zu verstehen wie Perl sich in bestimmten Situationen verhält.

1.2.2 Praktisch

Perl ist eine Kombination aus der Sprache C und Unix-Tools wie sed, awk und grep, sowie vielen eigenen Ideen des Entwicklers. Perl versucht die Lücke zwischen Low-Level-Programmierung und High-Level-Programmierung zu schließen und damit die Geschwindigkeit und die Möglichkeiten der Low-Level-Programme mit einer verständlichen und komfortableren Syntax zu verbinden. Perl ist also einfach, meistens schnell, fast unbegrenzt und ein bißchen häßlich.

1.3 Was bedeutet Perl ?

Ursprünglich nannte Larry seine Sprache “Pearl“[1] nach einem Zitat aus dem Matthäus-Evangelium [13,46]. Es stellte sich jedoch heraus, dass es bereits eine Sprache mit dem Namen Pearl gab, darum änderte er den Namen in “Perl“. Später entwickelten sich die allgemein verbreiteten und von Larry anerkannten Backronyme „Practical Extraction And Report Language“ und „Pathologically Eclectic Rubbish Lister“[2].

1 dt. die Perle
2 dt. krankhaft zusammengeschustertes Auflistungsprogramm für wirres Zeug

1.4 Die Perl Philosophie

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

Perl gestattet dem Programmierer viele Freiheiten und bietet daher absichtlich für jedes Probleme verschiedene Formulierungs- und Lösungsmöglichkeiten an. Damit erlaubt Larry dem Programmierer seine persönlichen Vorlieben in seine Programme einzubringen, was für viele ein Grund ist warum sie sich für Perl entscheiden. Viele Kritiker sehen hierin aber auch eins der größten Probleme von Perl, da es ihrer Meinung nach sehr schwer ist leserlichen Code zu schreiben. Gerade diese Freiheit, ermöglicht es aber auch besonders Nahe am Probleme zu programmieren was, richtig angewendet, den Code verständlicher macht, als es in einer anderen Sprache möglich wäre.

1.4.2 Perl makes easy Jobs easy and hard jobs possible

Dieses Perl Prinzip besagt, dass einfache Dinge in Perl schnell realisiert werden können und für schwierige Dinge zu mindestens Möglichkeiten zur Realisierung geboten werden sollen.

1.4.3 Kontextsensitiv

In Perl kann ein Befehl verschiedene Bedeutungen haben. Welche Wirkung dieser Befehl dann hervorruft ist davon abhängig in welchem Zusammenhang er benutzt wird.

1.5 Technische Merkmale

Das Programm perl selbst ist ein in C geschriebener Interpreter. Dieser ist mittlerweile für fast alle Betriebssysteme verfügbar. Ein perlscript wird beim starten von perl interpretiert, in Bytecode umgewandelt und dann ausgeführt. Perlscript selbst werden als Textdatei abgespeichert.

2 Die Perl Syntax

2.1 Freie Formatierung und Kommentare

Wie in vielen anderen Sprachen kann man in Perl beliebige Whitespace-Zeichen verwenden, um seinen Code übersichtlicher zu gestalten. Dies nennt man “Freie Formatierung“. Um diese Art der Formatierung zu ermöglichen, muss in Perl jeder Befehl mit einem Semikolon (;) abgeschlossen werden. Desweiteren gibt es in Perl Programmen keine main Routine wie z.B. in Java. Kommentare beginnen einem Doppelkreuz (#) und enden mit dem Zeilenende.

2.2 Die erste Zeile eines Perl Programms

Ein Perl Programm beginnt in der Regel mit der Zeile !#/usr/bin/perl. Beginnt in Unix Systemen eine Textdatei mit den Zeichen !#, so bedeutet dies, dass die Datei von dem Programm ausgeführt wird, welches nach !# angegeben wird. In diesem Fall wäre das

/usr/bin/perl, also der Perl Interpreter. Dem Interpreter können auch Optionen mit übergeben werden. So werden mit der Option -w (!#/usr/bin/perl -w) die Warnungen eingeschaltet.

2.3 Operatoren

2.3.1 Arithmetische Operatoren

Operator	*Bedeutung*
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Module
**	Exponentieren

2.3.2 String Operatoren

Operator	*Bedeutung*
.	Zwei Strings aneinanderhängen
x	Wiederholungsoperator

2.3.3 Logische Operatoren

Operator	*Bedeutung*
&& bzw. and	Und
|| bzw. or	Oder
! bzw. not	Nicht

2.3.4 Vergleichs Operatoren

Operator	*Bedeutung*
== bzw. eq	gleich
!= bzw. ne	ungleich
< bzw. lt	kleiner
> bzw gt	grösser
<= bzw. le	kleiner gleich
>= bzw. ge	grösser gleich

2.4 Datenstrukturen

2.4.1 Skalare

Die einfachsten Datentypen die Perl kennt sind Skalare. In den meisten Fällen sind Skalare eine Zahl oder ein String. Im allgemeinen gilt jedoch, ist in Perl von etwas Einzelnem die Rede, so handelt es sich um einen skalaren Wert bzw. einen Skalar. Daher kann ein Skalar auch eine Referenz oder ein Objekt oder ähnliches sein. Skalare sind so zu sagen der “Singular“ in der Sprache Perl.

Desweiteren können auf einen skalaren Wert Operatoren angewandt werden und man kann diese in sog. Skalaren Variablen abspeichern. Eine Skalare Variable beginnt mit dem Dollar-Zeichen ($) gefolgt von dem Variablenname. Dieser darf aus alphanumerischen Zeichen und Unterstrichen bestehen, muss jedoch mit einem Buchstaben beginnen. Hierbei unterscheidet Perl zwischen Groß- und Kleinschreibung (Das heißt die Variable $test ist nicht gleich der Variable $Test.). Um einer skalaren Variable einen Wert zu zuweisen, benutzt man den Zuweisungsoperator (das Gleichheitszeichen =).

2.4.2 Listen und Arrays

Eine Liste ist eine geordnete Sammlung von Skalaren Werten. Hierbei ist es nicht notwendig, dass alle Skalare den “gleichen Typ“ haben. Es sind also auch Listen möglich die sowohl Strings als auch Zahlen etc. enthalten. Listen repräsentieren daher den “Plural in der Sprache Perl.

Der Begriff Array wird oft als Synonym für eine Liste benutzt, das ist jedoch nicht ganz richtig. Genau genommen, ist ein Array eine Variable die eine Liste enthält. Ein Array beginnt mit dem
@-Zeichen gefolgt von einem beliebigen Namen (für die Namensgebung gelten die selben Regel wie bei Skalaren). Um die Werte in einem Array verändern oder auslesen zu können, besitzt jedes Arrayelement einen Index. Durch diesen Index kann auf die jeweilige Skalare Variable zugegriffen werden: $<Arrayname>[<Index>].

Arrays (und damit auch Listen) können eine beliebige Anzahl von Elementen enthalten. Die einzige Grenze wird hierbei durch die Größe des Hauptspeichers gesetzt. Damit ist auch hier das Perl-Prinzip der “Grenzenlosigkeit“ verwirklicht.

2.4.3 Hashes

Ein Hash ist im Prinzip ein Array, nur das hier der Index ein String ist, den der Programmierer frei wählen darf. Diesen Index nennt man auch Schlüssel. Ein Schlüssel ist immer ein String. Das heißt auch wenn man z.B. eine Zahl als Schlüssel angibt, wird diese automatisch in einen String konvertiert. Des weiteren darf ein Schlüssel nur einmal in einem Hash vorkommen. Jeder Schlüssel ist also eindeutig einem Wert zugeordnet. Somit handelt es sich bei einem Hash also um eine Liste von Schlüssel und Werte Paaren.

Ein Hash beginnt mit dem %-Zeichen gefolgt von einem beliebigen Namen (auch hier gelten die üblichen Konventionen). Der Zugriff auf ein Element im Hash erfolgt über den Schlüssel: $<Hashname>{<Schlüssel>}.

2.4.4 Kontextabhängig

An dieser Stelle sei nochmals darauf verwiesen, dass die oben genannten Datenstrukturen je nach Kontext eine verschiedene Bedeutung haben können. So repräsentiert z.B. @myarray im Kontext $test = 5 + @myarray kein Array sondern einen Skalaren Wert, der die Anzahl der Arrayelemente enthält. Gerade bei Funktionen spielt diese Kontextabhänigkeit eine große Rolle. Verwendet man eine Funktion im “falschen“ Kontext, so kann man nicht genau sagen, welches Ergebnis man erhalten wird. Dies hängt nur von der Implementierung der jeweiligen Funktion ab und kann somit stark variieren.

2.5 Kontrollstrukturen

2.5.1 If, unless

if (*Bedingung*) { *Ausdruck* } (**elsif** (...) {...}) (**else** {...})
unless (*Bedingung*) (**else** {...})
Bedingung **?** *Ausdruck wahr* **:** *Ausdruck falsch*

2.5.2 while, until, for

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

2.5.3 foreach

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

2.6 Subroutinen

Um eine Subroutine zu definieren verwendet man das Schlüsselwort **sub** gefolgt vom Namen der Funktion und einem Anweisungsblock: **sub** <*Name*> { *Ausdruck* }. Im Gegensatz dazu sieht die Syntax zum Aufruf einer Funktion wie folgt aus: &<Name>(arg1,arg2, ...,arg_n). Wobei arg_1 – arg_n die Funktionsargumente darstellen. Will der Programmierer der Subroutine auf die übergebenen Argumente zugreifen, so findet er diese in der Standardvariable @_. Diese Standardvariable ist, wie an dem @-Zeichen erkennbar, ein Array. Dieses Array enthält die Liste mit den Funktionsargumenten.

In Perl hat jede Subroutine einen Rückgabewert. Wird dieser nicht explizit mit dem Schlüsselwort **return** festgelegt, so wird das Ergebnis des letzten berechneten Ausdrucks zurückgegeben. Dieser Rückgabewert muss nicht unbedingt verwendet werden. Er muss also auch nicht in einer Variable zwischengespeichert werden. Wird eine Subroutine ohne Zuweisung aufgerufen, so wird der Rückgabewert einfach verworfen.

Desweiteren ist noch zu beachten das alle Variablen ohne das Schlüsselwort **my** global sind, d.h. auch in Subroutinen verändert werden können. Um einen lokale Variable in einem Block zu definieren verwendet man folgende Syntax: **my $**<*Name*>.

2.7 Ein- und Ausgabe

Mit Hilfe des **<STDIN>** Operators kann die nächste Zeile von der Standardeingabe gelesen werden. In der Regel ist dies die Tastatur. Durch diesen Operator ist es also möglich Benutzereingaben zu lesen. Es gilt zu beachten, dass der Operator auch das Newline-Zeichen(\n) mit einliest. Um diesen zu entfernen kann die **chomp** Funktion verwendet werden.

Die Ausgabe auf dem Terminal erfolgt durch **print** Funktion.

2.8 Reguläre Ausdrücke

Reguläre Ausdrücke sind ein sehr mächtiges Werkzeug und wurden schon früh in die Sprache Perl integriert. So war Perl eine der ersten Sprachen die reguläre Ausdrücke zuließ und dabei auch den Standard PCRE[3] setzte. Da es ganze Bücher gibt die sich allein mit dem Thema reguläre Ausdrücke beschäftigen, soll nachfolgend nur eine kurze Einführung zu regulären Ausdrücken in der Sprache Perl gegeben werden.

Ein regulärer Ausdruck besteht aus einem Stringmuster, dass eine Menge von Zeichketten beschreibt, d.h. die Möglichkeit bietet einen String nach bestimmten Wörtern oder Zeichenfolgen zu durchsuchen. Diese Suchmuster unterliegen einer gewissen Syntax, die im Unix-Tools Vortrag „Reguläre Ausdrücke“ vom 08.11.05, vorgestellt wurde und auf die daher an dieser Stelle nicht näher eingegangen wird. Nachdem ein Suchmuster erstellt wurde, kann man einen String durchsuchen in dem man den Bindungsoperator (=~) verwendet. Die Syntax sieht hierbei wie folgt aus:

<String> **=~** <Suchmuster><Optionen>.

Die Optionen wirken sich hierbei auf die Art der Suche aus. Wichtige Optionen wären z.B.:

- **g** – globale Suche, d.h. es werden alle Vorkommen des Suchmusters im String zurückgegeben
- **i** – es wird bei der Suche nicht zwischen Groß- und Kleinschreibung unterschieden

Nach der Verwendung eines regulären Ausdrucks stehen ein paar Sondervariablen zu Verfügung:

- **$&** - enthält den erkannten String
- **$`** - enthält den Teilstring vor dem erkanntem String
- **$'** - enthält den Teilstring nach dem erkanntem String
- **$1..$9** - Ergebnisse der geklammerten Subausdrücke

Neben dem Bindungsoperator steht in Perl auch noch ein „suchen und ersetzen“ Befehl zur Verfügung. Die Syntax des Befehls sieht wie folgt aus:

s/<Suchmuster>**/**<Ersetzungsausdruck>**/**<Optionen>

Mit Hilfe dieses Befehls werden Strings die dem Suchmuster entsprechen durch den Ersetzungsausdruck ersetzt. Als Optionen stehen die gleichen Möglichkeiten zur Verfügung wie beim Bindungsoperator.

3 Perl Compatible Regular Expressions vgl. Vortrag über Reguläre Ausdrücke vom 08.11.05

3 Anhang

3.1 Ausblick

Da dies lediglich als ein Einführung in Perl gedacht ist, gibt es natürlich noch eine Reihe von weiteren interessanten Themen rund um die Sprache Perl. Hier wären zum Beispiel folgende Themen zu nennen:

- Module (www.cpan.org)
- Webprogrammierung
- String und Sortierfunktionen
- Datenbanken
- Dateihandles und Dateitests
- Fehlerbehandlung

3.2 Wo bekomme ich Hilfe ?

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

3.3 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