

# Programmieren mit Perl



## Referenz zum Kurs

Dr. Kuhlmann Beratung Software-Engineering

Copyright ©2002 Dr. Kuhlmann Beratung Software-Engineering [www.dr-kuhlmann.com](http://www.dr-kuhlmann.com)

Diese Unterlage darf nur vervielfältigt werden, wenn sie nicht verändert wurde.

Irrtümer vorbehalten, für deren Folgen wird keine Haftung übernommen.

**Dr. Kuhlmann Beratung Software-Engineering**

Unter den Eichen 30

28857 Syke

Tel. 04240/95078

Email: [info@dr-kuhlmann.com](mailto:info@dr-kuhlmann.com)

FAX. 04240/95077

URL: [www.dr-kuhlmann.com](http://www.dr-kuhlmann.com)

07.03.02

Inhaltsverzeichnis

- 1 Perl-Aufruf.....3
- 2 Debugger.....3
- 3 Skalare.....4
  - 3.1 Literale.....4
  - 3.2 Sonderzeichen in Textliteralen.....4
  - 3.3 Zeichenattribute in Textliteralen.....4
  - 3.4 Skalare Variable.....5
- 4 Logische Werte.....5
- 5 Listen und Arrays.....5
- 6 Hashes.....6
- 7 Operatoren.....6
  - 7.1 Arithmetische Operatoren.....7
  - 7.2 String-Operatoren.....7
  - 7.3 Zuweisungsoperatoren.....7
  - 7.4 Auto-Operatoren.....7
  - 7.5 Boolesche Operatoren.....8
  - 7.6 Vergleichsoperatoren.....8
  - 7.7 Ein- Ausgabe-Operatoren.....8
  - 7.8 Bit-Operatoren.....8
  - 7.9 Prüfoperatoren für Dateien.....9
  - 7.10 Monadische Operatoren.....10
  - 7.11 Q-Operatoren.....10
  - 7.12 Modifikatoren von Q-Operatoren.....10
  - 7.13 Sonstige Operatoren.....11
- 8 Kontrollstrukturen.....11
  - 8.1 Modifikatoren von Ausdrücken.....11
- 9 Referenzen.....12
  - 9.1 Einfache Referenzen.....12
  - 9.2 Datenstrukturen mit Referenzen.....12
  - 9.3 Referenzen auf Unterprogramme.....12
  - 9.4 Anonyme Referenzen.....12
  - 9.5 Komplexe Datenstrukturen.....13
- 10 Ein- Ausgabe.....14
  - 10.1 open-Modus einer Datei.....14
  - 10.2 Operationen auf Dateihandles.....14
  - 10.3 Formatierte Ausgabe.....15
  - 10.4 Feldtyp bei formatierter Ausgabe.....15
  - 10.5 Globbing von Dateien.....15
- 11 Unterprogramme.....16
  - 11.1 Prototypen.....16
- 12 Pattern-Matching.....17
  - 12.1 Text-Transformationen.....17
  - 12.2 Operatoren für Pattern-Matching.....17
  - 12.3 Modifikatoren für Pattern-Matching.....17
- 13 Reguläre Ausdrücke.....18
  - 13.1 Metazeichen in reg. Ausdrücken.....18
  - 13.2 Sonderzeichen in reg. Ausdrücken.....18
  - 13.3 Untermuster in reg. Ausdrücken.....19
- 14 Pakete.....20
  - 14.1 Module.....20
- 15 Vordefinierte Variable.....21
- 16 Hilfreiche Funktionen.....21
- 17 CGI-Interface.....23
- 18 Modul CGI\_Interface.....23

Haftungsausschluss

Diese Referenz soll Anfängern Anwender von Perl bei der täglichen Arbeit unterstützen. Sie umfasst nur einen Teil des Sprachumfangs von Perl 5.5. Diese Unterlage wurde sorgfältig erstellt. Dennoch könnten sich Irrtümer eingeschlichen haben.

Für Schäden, die unmittelbar oder mittelbar durch die Anwendung dieser Unterlage entstehen übernimmt *Dr. Kuhlmann Beratung Software-Engineering* keinerlei Haftung. Die Benutzung dieser Unterlage erfolgt auf Ihr eigenes Risiko.

Falls Sie Fehler, Irrtümer entdecken oder Anregungen haben, senden Sie bitte eine E-Mail an [info@dr-kuhlmann.com](mailto:info@dr-kuhlmann.com)

Copyright ©

Diese Unterlage darf von jedermann kopiert und verwendet werden. Sie darf ohne schriftliche Einwilligung von *Dr. Kuhlmann Beratung Software-Engineering* nicht verändert werden.

17 CGI-Interface

Umgebungsvariable	Inhalt
\$ENV{'Variablenname'}	Umgebungsvariablen unter Perl über den Hash %ENV
CONTENT_LENGTH	Anzahl von Zeichen, die Post übergeben wird
CONTENT_TYPE	MIME-Type der Daten, die Post übergeben wird
GATEWAY_INTERFACE	Version der installierten CGI-Schnittstelle
HTTP_ACCEPT	Dateiformate (MIME-Types), die der Server unterstützt /': Server unterstützt alle Dateitypen
HTTP_REFERER	URL-Adresse, von der aus zugegriffen wird
HTTP_USER_AGENT	Browserversion, Browsertyp ...
PATH_INFO	Bei Get: Pfad der aufrufenden Datei auf dem Server
PATH_TRANSLATED	Bei Get: Pfadnamen im ServerCode
QUERY_STRING	Bei Get: übergebene Zeichenkette
REMOTE_ADDR	IP-Adresse des aufrufenden Servers
REMOTE_HOST	IP-Adresse der aufrufenden Domain
REMOTE_IDENT	Protokollinformation, wenn auf dem Server das Protokoll 'ident' für geschützte Zugriffe aktiv ist.
REMOTE_USER	Name des aufrufenden Users - nur bei aktivierter Server-Authentifizierung.
REQUEST_METHOD	Aufruf durch Get oder Post
SCRIPT_NAME	Adresse und Namen des aufgerufenen Scripts
SERVER_NAME	Name des Servers aus auf dem das Script liegt
SERVER_PORT	Port-Nummer des Servers. In der Regel 80 (http) oder 21 (ftp).
SERVER_PROTOCOL	Angaben über die unterstützte http-Version.
SERVER_SOFTWARE	Gibt aus, welche Server-Software installiert ist

18 Modul CGI\_Interface

	Beispiel
Beispiel	<pre> use strict; use CGI_Interface;  my \$request; my %rqpairs;  CGI_Interface::get_request (\\$request, \%rqpairs);  ...  print "Content-type: text/html\n\n";  # print "Content-type: text/plain\n\n";                     </pre>

## Programmieren mit Perl

Funktion	Beschreibung
quotemeta AUSDR	gibt Text AUSDR mit quotierten (\) Metazeichen von regulären Ausdrücken zurück
rand AUSDR	liefert einen Zufallswert zwischen 0 und AUSDR
ref AUSDR	ist AUSDR eine Referenz
return LISTE	beendet Unterprogramm, das dann LISTE zurückgibt
reverse LISTE	Listen-Kontext: gibt LISTE in umgekehrter Reihenfolge zurück Skalar-Kontext: erstes Element von LISTE mit den Bytes in umgekehrter Folge
shift @ARRAY	holt erstes Element aus @ARRAY, das erste Element wird in @ARRAY gelöscht.
sin AUSDR	liefert den Sinus von AUSDR im Bogenmaß
sleep AUSDR	hält das aktuelle Programm für (mindestens) AUSDR Sekunden an
sort LISTE	gibt die asciibetisch sortierte LISTE zurück
split /MISTER/, AUSDR	gibt Liste mit den Teilen von AUDR zurück, die durch MUSTER getrennt sind
sqrt AUSDR	liefert die Quadratwurzel von AUSDR
time	gibt die aktuelle Zeit in absoluten Sekunden.
uc AUSDR	gibt Text AUSDR mit Großbuchstaben zurück
ucfirst AUSDR	gibt Text AUSDR mit 1. Buchstaben in Großbuchstaben zurück
unshift @ARRAY, LISTE	schiebt LISTE vorne in @ARRAY
values HASH	erzeugt Liste mit den Werten der Liste

## Programmieren mit Perl

### 1 Perl-Aufruf

Betriebssystemen	Beispiel / Aufruf	
Unix	Programm	<pre>#!/usr/bin/perl -w use strict; print 'hallo';</pre>
	Datei	<code>-rwxr-x--- emil user 35 Apr 26 14:11 hallo</code>
	Programmaufruf	<code>perl -w hallo</code> hallo
	Debugger	<code>perl -w -d hallo.pl</code>
	graphischer Debugger	<code>perl -d:ptkdb hallo.pl</code>
	Profiler (erstellen)	<code>perl -d:DProf hallo.pl</code>
	Profiler (anzeigen)	<code>dprofpp</code>
Windows	Programm	<pre>#!/usr/bin/perl -w use strict; print 'hallo';</pre>
	Datei	hallo.pl
	Programmaufruf	<code>perl -w hallo.pl</code>
	Debugger	<code>perl -w -d hallo.pl</code>
	graphischer Debugger	<code>perl -d:ptkdb hallo.pl</code>
	Profiler (erstellen)	<code>perl -d:DProf hallo.pl</code>
	Profiler (anzeigen)	<code>dprofpp</code>

### 2 Debugger

Befehl	Beschreibung
h	Hilfe
h BEFEHL	Hilfe zum Debugger-BEFEHL
s	Einzelschritt auch in ein Unterprogramm
n	Einzelschritt über ein Unterprogramm
<RETURN>	letzten Einzelschritt-Befehl ( s oder n ) wiederholen
c [ZEILE]	Bis Zeile oder nächsten Breakpoint fortsetzen
p AUSDR	Drucke AUSDR ( mit Bezug auf Variablen des Programms )
l	liste einen Bereich um die aktuelle Zeile
b [ZEILE]	setze auf ZEILE einen Breakpoint
d [ZEILE]	lösche den Breakpoint auf ZEILE
D	lösche alle Breakpoints
L	liste alle Breakpoints
q	beende Debugger / Programm

### 3 Skalare

#### 3.1 Literale

Typ	Variante	Beispiel
Texte	Textkonstante	'Ein Text'
	interpolierter Text	"Ein Text mit <CR>\n"
Ganzzahlen	Dezimal	123 123_456_789
	Oktal	0377
	Hexadezimal	0xab7ff
Gleitkommazahlen	standard	123.79 -55.88
	wissenschaftlich	123.45E05

#### 3.2 Sonderzeichen in Textliteralen

Kode	Wirkung in Textliteralen mit "...'' (außer '/' und '//, die auch in '...' wirken)
\'	' (wirkt auch in Textliteralen mit '..')
\\	\ (wirkt auch in Textliteralen mit '..')
\"	"
\n	Zeilenvorschub (newline)
\r	Wagenrücklauf (carrage return)
\t	Horizontaler Tabulator
\f	Seitenvorschub (form feed)
\b	Backspace
\a	Alarm (bell)
\e	Escape-Zeichen (ESC)
\033	ESC in Oktal
\x7f	DEL in Hexadezimal

#### 3.3 Zeichenattribute in Textliteralen

Kode	Wirkung in Textliteralen mit "...''
\u	Nächstes Zeichen wird groß geschrieben
\l	Nächstes Zeichen wird klein geschrieben
\U	Alle nachfolgenden Zeichen werden groß geschrieben
\L	Alle nachfolgenden Zeichen werden klein geschrieben
\Q	Alle nachfolgenden nicht alphanumerischen Zeichen werden mit Backslash versehen
\E	Beendet \U \L und \Q

### 15 Vordefinierte Variable

Variable	Bedeutung
@_	Liste der aktuellen Parameter eines Unterprogramms
#!	Fehlermeldung des letzten Systemaufrufs. Ist nur unmittelbar nach dem Systemaufruf zuverlässig!
\$0	Befehlsname der Kommandozeile
\$1 ...	Text, der der 1. ... Klammer beim letzten Pattern-Matching
\$\$	Nummer des aktuellen Prozesses
@ARGV	Liste der Argumente der Kommandozeile

### 16 Hilfreiche Funktionen

Funktion	Beschreibung
abs AUSDR	liefert den absoluten Wert von AUSDR
char AUSDR	liefert den Buchstaben, zu AUSDR in der ASCII-Tabelle
chomp LISTE	schneidet ein eventuelles \n am Ende allen Elementen der LISTE ab
chop LISTE	schneidet letztes Zeichen am Ende allen Elementen der LISTE ab
cos AUDR	liefert den Sinus von AUSDR im Bogenmaß
defined AUSDR	hat AUSDR einen Wert
die LISTE	beende das laufende Programm mit der Meldung LISTE
delete \$HASH{KEY}	löscht den Eintrag KEY aus dem %HASH
each HASH	while ( (\$schlüssel, \$wert) = each %Hash ) ...
exists \$HASH{KEY}	gibt es KEY in HASH
exit AUSDR	beendet das laufende Programm und gibt den Wert von EXPR zurück
exp AUSDR	liefert e hoch AUSDR
glob AUSDR	liefert Liste von Dateien, die AUSDR (mit *) entsprechen, zurück
int AUSDR	liefert den ganzzahligen Anteil von AUSDR.
join AUSDR, LISTE	bildet aus LISTE einen Text wobei deren Elemente durch AUSDR getrennt sind
keys HASH	gibt die Liste die Schlüssel des HASH zurück
scalar keys HASH	gibt die Anzahl der HASH-Elemente zurück
lc AUDR	gibt Text AUSDR mit Kleinbuchstaben zurück
lfirst AUSDR	gibt Text AUSDR mit 1. Buchstaben in Kleinbuchstaben zurück
length AUSDR	Textlänge von AUSDR
localtime AUSDR	(\$sec, \$min, \$std, \$ntag, \$mon, \$jahr, \$wtag, \$jtag, \$isdst) = localtime ( time )
scalar localtime AUSDR	gibt die lokale Zeit von AUSDR als Text zurück
log AUSDR	liefert den natürlichen Logarithmus von AUSDR
ord AUSDR	liefert die Zahl, die dem Buchstaben AUSDR in der ASCII-Tabelle entspricht
pop @ARRAY	Letzter Wert aus @ARRAY, des letzte Element wird aus @ARRAY entfernt
print LISTE	schreibt alle Elemente von LISTE auf STDOUT
print HANDLE LISTE	schreibt alle Elemente von LISTE in die Datei unter HANDLE
push @ARRAY, LISTE	hängt LISTE an @ARRAY an

## 14 Pakete

	Syntax / Beispiel
Deklaration	<code>package Paketname;</code>
Sichtbarkeit	<code>my \$var; sub up1 { ... };</code>
Default-Paket	<code>package main;</code>
Erreichbarkeit	<code>\$Paketname::var=1; Paketname::up1();</code>

### 14.1 Module

	Syntax / Beispiel
Modulname	<code>package Modulname;</code>
Dateiname	<code>Modulname.pm</code>
Verwendung	<code>require Modulname;</code>
Struktur	<code>package Modulname; ... # end Modulname  1;</code>
Submodule	<code>package PAKET::SUBPAKET;</code>
Dateiname	<code>PAKET/SUBPAKET.pm</code>
Verwendung	<code>require PAKET::SUBPAKET;</code>
Erreichbarkeit	<code>#!/perl -w -I /usr/local/perl</code>
Exporter ins Modul binden	<code>package My_Modul; use Exporter (); @ISA = qw( Exporter );</code>
default exportierte Symbole	<code>@EXPORT = qw( ... );</code>
auf Anfrage exportierte Symbole	<code>@EXPORT_OK = qw ( ... );</code>
Sätze von exportierten Symbole	<code>@EXPORT_TAGS = ( tag =&gt; [ ... ] );</code>
Default-Symbole importieren	<code>use Modul;</code>
angegebene Symbole importieren	<code>use Modul qw ( ... );</code>
keinerlei Symbole importieren	<code>use Modul();</code>

## 3.4 Skalare Variable

Kontext	Beispiel
Bezeichnung	<code>\$variable</code>
lexikalisch lokale Deklaration	<code>my \$variable; my \$zahl = 55; my \$kurt = 'Kurt'; my (\$links, \$rechts);</code>
dynamische lokale Deklaration	<code>local \$nur_wenn_Sie_wissen_was_Sie_tun;</code>
Verwendung	<code>\$variable = 55; \$variable = \$zahl - 6;</code>
bei interpolierten Texten	<code>\$variable = "Sein Name war \$kurt.\n"; \$variable = "Sein Name war \${kurt}chen.\n";</code>

## 4 Logische Werte

Bedeutung	Wert
falsch, false	<code>0 0.0 " '0' undef</code>
wahr, true	alles, was nicht falsch ist, ist wahr.

## 5 Listen und Arrays

Liste bzw. Array	Beschreibung / Beispiel
literale Werte	<code>('Kopf', 'Mitte', 'Schwanz')</code>
Deklaration	<code>my @Woche; my @Werkstage = ('Mo', 'Di', 'Mi', 'Do', 'Fr'); my @Wochenende = ('Sa', 'So');</code>
Verwendung als Liste	<code>@Woche = (@Werkstage, @Wochenende); foreach \$tag ( @Woche ) { ... };</code>
Zugriff auf einzelne Elemente	<code>\$Array[0] = 'erstes Element'; \$Array[4] = 55;</code>
Indexbereich von Arrays	<code>0..\$#Array</code>
Wert des letzten Elements	<code>\$Array[\$#Array]</code>
Ausschnitt eines Arrays	<code>@Array[0..1] @Array[0..4] = ( 0, 1, 4, 9, 16 );</code>
Werte für Arrays berechnen	<code>foreach \$i ( 5..100 ) { \$Array {\$i} = \$i * \$i; };</code>

## 6 Hashes

Hash	Beschreibung / Beispiel
literale Werte	( 'key' => 'value', 'schluessel' => 'wert' )
Deklaration	<pre>my %Deutsch; my %Pronomen = ( 'he' =&gt; 'er',                  'she' =&gt; 'sie',                  'it' =&gt; 'es' ); my %Ich = ( 'I' =&gt; 'ich' );</pre>
Verwendung	<pre>%Deutsch = (%Pronomen, %Ich); \$Deutsch{'we'} = 'wir'; foreach \$english ( keys %Deutsch ) {     print "\$english -&gt; \$Deutsch{\$english}; };</pre>

## 7 Operatoren

Vorrang	Assoziativität	Operatoren
höchster	links	Terme und Listenoperatoren (nach links)
	links	->
	nicht assoziativ	++ --
	rechts	**
	rechts	! ~ \ sowie monadisches + und -
	links	== !~
	links	* / % x
	links	+ - .
	links	<<>>
	nicht assoziativ	benannte monadische Operatoren wie log, sin
	nicht assoziativ	<> <= >= lt gt le ge
	nicht assoziativ	== != <=> eq ne cmp
	links	&
	links	^
	links	&&
	links	
	nicht assoziativ	.. ...
	rechts	?:
	rechts	= += -= *= etc.
	links	, =>
	nicht assoziativ	Listenoperatoren (nach rechts)
	rechts	not
	links	and
niedrigster	links	or xor

## 13.3 Untermuster in reg. Ausdrücken

Muster	Bedeutung
()	Das Untermuster in der Klammer wird als ein Zeichen aufgefasst ( Gruppierung )
[]	prüft ob ein Zeichen aus denen in der Klammer vorhanden ist ( Zeichenklasse )
[^]	prüft ob kein Zeichen aus denen in der Klammer vorhanden ist
\1 ... \9 ...	\1 ... \9 ... meint die 1te bis 9te Klammer usw. im regulären Ausdruck
\$1 \$2 ... \$10 ...	\$1 gefundener Wert der 1. Klammer ... \$10 gefundener Wert der 10. Klammer usw.
( )	Alternativen werden durch ein   getrennt
{n,m}	Muss mindestens n aber nicht mehr als m Mal vorkommen
{n,}	Muss mindestens n Mal vorkommen
{n}	Muss genau n Mal vorkommen
*	Muss 0 oder mehr Mal vorkommen ( entspricht {0,} )
+	Muss ein- oder mehrmals vorkommen ( entspricht {1,} )
?	Muss kein- oder einmal vorkommen ( entspricht {0,1} )

### 13 Reguläre Ausdrücke

#### 13.1 Metazeichen in reg. Ausdrücken

Metazeichen	Beispiel	Beschreibung
( )	(mm)	zusammenfassen
[ ]	[a-z]	ein Zeichen aus
{ }	a{n,m}	n bis m Mal wiederholen
?	a?	0 oder 1 Mal vorhanden
*	a*	0 oder mehr Mal vorhanden
+	a+	1 oder mehr Mal vorhanden
^	^muster [^ausw]	Anfang des Textes kein Zeichen aus
\$	muster\$ \$variable	Ende des Textes Variable
	(ma mb)	entweder muster ma oder Muster mb
\	\\. \s	Metazeichen einsetzen Sonderzeichen, Menge von Zeichen usw.
.	.	Punkt steht für ein beliebiges Zeichen

#### 13.2 Sonderzeichen in reg. Ausdrücken

Symbol	Zeichen / Erläuterung
\\ usw.	literale Metazeichen
\a	Alarm (beep)
\n	Zeilenvorschub (newline)
\r	Wagenrücklauf (carriage return)
\t	Tabulator
\f	Seitenvorschub (formfeed)
\e	Escape
\012	\0 gefolgt von 2 Oktalziffern entspricht dem Zeichen mit dem Oktalwert 12
\x3f	\x gefolgt von 2 Hexziffern entspricht dem Zeichen mit den Hexwert 3f
\cD	\c gefolgt von einem Zeichen entspricht dem Steuer-Zeichen C <sup>D</sup>
\0 \1 .. \9	\0 meint ", \1 .. \9 meint die 1te bis 9te Klammer im regulären Ausdruck
\12	meint die 12. Klammer, falls vorhanden sonst Zeichen mit dem Oktalwert
\d	Ziffer (digit), wie [0-9]
\D	Nicht-Ziffer (non digit)
\s	Whitespace, wie [ \t\n\r\f]
\S	Nicht-Whitespace
\w	Wort-Zeichen (aphanumerisch), wie [a-zA-Z_0-9]
\W	Nicht-Wort (non word)

### 7.1 Arithmetische Operatoren

Operator	Beispiel	Ergebnis bei \$a=124 und \$b=3
+	\$a + \$b	127
*	\$a * \$b	372
%	\$a % \$b	1
**	\$a ** \$b	0

### 7.2 String-Operatoren

Operator	Beispiel	Ergebnis bei \$a=124, \$b=3	Interpretation der Operanden
.	\$a . \$b	1243	\$a String \$b String
x	\$a x \$b	124124124	\$a String \$b ganze Zahl

### 7.3 Zuweisungsoperatoren

Operator	Beispiel	Ergebnis bei \$a=123, \$b=3
Lvalue = Ausdruck	if ( \$a = \$b)	\$a=\$b; if ( \$a )
Lvalue OP= Ausdruck	\$a OP= \$b;	\$a = \$a OP \$b;
**=	\$a **= \$b	
+=	\$a += \$b	
*=	\$a *= \$b	
&=	\$a &= \$b	
<<=	\$a <<= \$b	
&&=	\$a &&= \$b	
=	\$a = \$b	
/=	\$a /= \$b	
=	\$a  = \$b	
>>=	\$a >>= \$b	
>>=	\$a >>= \$b	
.=	\$a .= \$b	
%=	\$a %= \$b	
^=	\$a ^= \$b	
x=	\$a x= \$b	

### 7.4 Auto-Operatoren

Operator	Beispiel	Ergebnis bei \$a=124
++	\$b=\$a++;	\$a=125, \$b=124
++	\$b=++\$a;	\$a=125, \$b=125
--	\$b=\$a--;	\$a=123, \$b=124
--	\$b=--\$a;	\$a=123, \$b=123

### 7.5 Boolesche Operatoren

Operator	Beispiel	Ergebnis bei \$a=124 \$b=0
&&	\$a && \$b	0
	\$a    \$b	1
!	! \$a	0
and	\$a and \$b	0
or	\$a or \$b	1
not	not \$a	0
xor	\$a xor \$b	1

### 7.6 Vergleichsoperatoren

Operator	Beispiel	Ergebnis bei \$a=124 \$b=3	Operator	Beispiel	Ergebnis bei \$a=124 \$b=3
==	\$a == \$b	0	eq	\$a eq \$b	0
!=	\$a != \$b	1	ne	\$a ne \$b	1
<	\$a < \$b	1	lt	\$a lt \$b	1
>	\$a > \$b	0	gt	\$a gt \$b	0
>=	\$a >= \$b	1	le	\$a le \$b	1
<=	\$a <= \$b	0	ge	\$a ge \$b	0
<=>	\$a <=> \$b	1	cmp	\$a cmp \$b	0
\$a <=> \$b wenn:	\$a > \$b \$a == \$b \$a < \$b	1 0 -1	\$a cmp \$b wenn:	\$a gt \$b \$a eq \$b \$a lt \$b	1 0 -1

### 7.7 Ein- Ausgabe-Operatoren

Operator	Beispiel	Beschreibung
<HANDLE>	\$a = <HANDLE>;	
<>	\$a = <>;	Diamond-Operator
<*.htm>	@html_files = <*.htm>;	
`ls`	@files = `ls -l`;	Backtick-Operator `...`
print	print @list; print HANDLE @list;	print ist ein Listenoperator

### 7.8 Bit-Operatoren

Operator	Beispiel	Ergebnis bei \$a=122 \$b=3
&	\$a & \$b	2
	\$a   \$b	123
^	\$a ^ \$b	121
>>	\$a >> \$b	976
<<	\$a << \$b	15

## 12 Pattern-Matching

### 12.1 Text-Transformationen

Transformationen	Syntax / Beschreibung
Syntax	\$text =~ tr/VonListe/NachListe/ \$text =~ y/VonListe/NachListe/
Modifikatoren	\$text =~ tr/VonListe/NachListe/cds
c	die eigentliche Suchliste sind alle Zeichen, die nicht in VonListe stehen
d	Jedes Zeichen, das nicht in VonListe steht, wird in \$text gelöscht
s	Pakete - aufeinander folgende gleiche Zeichen - ersetzen

### 12.2 Operatoren für Pattern-Matching

Operation	Syntax / Beispiel
Musterbindungsoperator	\$text =~ m/gibt_es_muster/ \$text !~ m/gibt_es_muster_nicht/
Mustererkennung	\$text =~ m/muster/ \$text =~ /muster/ \$text =~ m%muster%
Text ersetzen	\$text =~ s/ersetze/durch/
Modifikatoren	\$text =~ m/mu*h/io \$text =~ s/ersetze/durch/g

### 12.3 Modifikatoren für Pattern-Matching

Modifikator	Bedeutung
e	betrachte die rechte Seite von s/links/rechts/ als Ausdruck
ee	wie e anschließend wird der Rückgabewert erneut berechnet
g	globale Suche, d.h. alle Vorkommen in dem Text
gc	Suchposition bei fehlgeschlagenem Matching nicht zurücksetzen
i	ignoriert Groß- und Kleinschreibung
m	betrachtet Text als mehrere Zeilen ( ^ und \$ prüfen \n )
o	der reguläre Ausdruck wird nur einmal berechnet - ist nur möglich, wenn darin keine Variablen verwendet werden
s	betrachtet Text als eine Zeile ( ^ und \$ ignorieren \n, aber . prüft \n )
x	erhöht die Lesbarkeit durch Leerzeichen und Kommentare

## 11 Unterprogramme

Unterprogramm	Beispiel
Deklaration	<code>sub Unterprogramm { BLOCK };</code>
Parameter	<code>sub Unterprogra {   my \$message = shift;   my @receiver = @_;</code>
Aufruf	<code>Unterprogramm; Unterprogramm ( LISTE );</code>
Rückgabe	<code>return ( LISTE );</code>
Prototyp	<code>sub Unterprogramm (\$@) ...</code>
Vorwärts-Deklaration	<code>sub Unterprogramm;</code>
explizite Referenz	<code>my \$command = \&amp;test_rate_plotter;</code>
implizite Referenz	<code>my \$command = sub { exit; };</code>

### 11.1 Prototypen

Element	Beschreibung	Beispiel
\$	Skalar	<code>sub Beispiel (\$\$) ...</code>
@	Liste	<code>sub Beispiel (\$@) ...</code>
%	Hash	<code>sub Beispiel (\$\$%) ...</code>
&	Unterprogramm	<code>sub Beispiel (&amp;%) ...</code>
*	Typeglobe	<code>sub Beispiel (**) ...</code>
;	folgende Parameter sind optional	<code>sub Beispiel (\$\$;\$)\$) ...</code>
\	Referenz auf	<code>sub Beispiel (\$@\%) ...</code>
	kein Parameter	<code>sub Beispiel () ...</code>

## 7.9 Prüfoperatoren für Dateien

Operator	Beispiel	Beschreibung
-r	<code>if (-r \$a)</code>	Datei ist lesbar von effektiver uid/gid
-w	<code>if (-w \$a)</code>	Datei ist beschreibbar von effektiver uid/gid
-x	<code>if (-x \$a)</code>	Datei ist ausführbar von effektiver uid/gid
-o		Datei gehört dem effektiver uid
-R		Datei ist lesbar von realer uid/gid
-W		Datei ist beschreibbar von realer uid/gid
-X		Datei ist ausführbar von realer uid/gid
-O		Datei gehört dem realer uid
-e	<code>if (-e \$a)</code>	Datei existiert
-z		Datei hat Größe 0
-s		Datei hat Größe > 0, gibt Dateigröße zurück
-f	<code>if (-f \$a)</code>	Datei ist normale Datei
-d	<code>if (-d \$a)</code>	Datei ist ein Verzeichnis
-l	<code>if (-l \$a)</code>	Datei ist ein symbolischer Link
-p		Datei ist eine benannte Pipe (FIFO) oder Filehandle ist eine Pipe
-b		Datei ist eine Block-Spezial-Datei
-c		Datei ist eine Text-Spezial-Datei
-t		Datei ist auf eine Konsole (tty) geöffnet
-u		Datei hat das setuid Bit gesetzt
-g		Datei hat das setgid Bit gesetzt
-k		Datei hat das sticky Bit gesetzt
-T		Datei ist eine Text-Datei
-B		Datei ist eine binäre Datei (das Gegenteil von -T)
-M		Alter der Datei in Tagen als das Script startet
-A		Zugriffszeit der Datei in Tagen als das Script startet
-C		Inode-Änderung der Datei in Tagen als das Script startet

7.10 Monadische Operatoren

Operator	Beispiel	Ergebnis bei \$a=122 \$b=2.75 \$c='Hallo'	Beschreibung
lc AUSDR	lc ( \$c )	'hallo'	lc : Klein- uc : Großbuchstaben
ref AUSDR	ref ( \$a )	0	ist AZSDR eine Referenz
cos AUSDR	cos ( \$a )	0.9243	abs exp log rand sin sqrt tan
int AUSDR	int ( \$b )	2	ganze Zahl von AUSDR
chr AUSDR	chr ( \$a )	'z'	ascii-Zeichen der Zahl AUSDR
undef VARIABLE	undef ( \$a )	\$a = undef	VARIBALE hat keinen Wert mehr
length AUSDR	length ( \$b )	5	Länge des Textes AUSDR
scalar AUSDR	scalar ( \$a )		erzwingt skalaren Kontext
defined AUSDR	defined ( \$a )	1	

7.11 Q-Operatoren

Operator	Beschreibung/Beispiel	entspricht
q/STRING/	literaler Text, String	'STRING'
qq/STRING/	interpolierter Text	"STRING"
qx/STRING/	interpolierter Systemaufruf	`STRING`
	@result = qx/find . -type f/; @result = `find . -type F`;	
qw/STRING/	Liste durch Leerzeichen getrennt	split('_', q/STRING/);
	qw ( Mo Di Mi Do Fr Sa So )	
qr/PATTERN/	\$rex = qr/my.STRING/is; s/\$rex/foo/;	s/my.STRING/foo/is;
	re = qr/\$pattern/; \$string =~ /foo\${re}bar/; \$string =~ \$re; \$string =~ /\$re/;	

7.12 Modifikatoren von Q-Operatoren

Modifikator	Bedeutung
i	ignoriert Groß- und Kleinschreibung
m	betrachtet Text als mehrere Zeilen ( ^ und \$ prüfen \n )
o	der reguläre Ausdruck wird nur einmal berechnet - ist nur möglich, wenn darin keine Variablen verwendet werden
s	betrachtet Text als eine Zeile ( ^ und \$ ignorieren \n, aber . prüft \n )
x	erhöht die Lesbarkeit des regulären Ausdrucks durch Leerzeichen und Kommentare

10.3 Formatierte Ausgabe

Aktion / Feld	Syntax / Beschreibung
printf	printf HANDLE FORMAT, LIST; printf FORMAT, LIST;
sprintf	sprintf FORMAT, LIST;
FORMAT-Sting	enthält Text mit Feldplatzhaltern
Feldplatzhalter	%m.nx
m, n	optionale Breite und Größe in Abhängigkeit vom Feldtyp x
x	Feldtyp

10.4 Feldtyp bei formatierter Ausgabe

Kode	Bedeutung
c	Zeichen
d	Dezimal-Integer
e	Fließkommazahl im Exponentenformat
f	Fließkommazahl im festen Format
g	Fließkommazahl im Kompaktformat
ld	Dezimal-Integer vom Typ long
lu	Oktal-Integer vom Typ long
lx	Hexadezimal-Integer vom Typ long ohne Vorzeichen
o	Oktal-Integer
s	String
u	Dezimal-Integer ohne Vorzeichen
x	Hexadezimal-Integer
X	Hexadezimal-Integer mit Großbuchstaben

10.5 Globbing von Dateien

Syntax	Beschreibung
<Muster>	Eine Lister aller Dateien, die Muster entsprechen
glob (Muster)	Operator-Schreibweise
<b>Muster</b>	
Text	Alle Zeichen außer den speziellen Zeichen
*	entspricht einer beliebigen Folge von Zeichen
?	ein beliebiges Zeichen
[abc]	ein Zeichen aus abc
<b>Beispiele</b>	
@files = <*.html>	alle html-Dateien im aktuellen Verzeichnis
@files = glob (" \$dir/*.[ch] ")	alle c- oder h-Dateien im Verzeichnis \$dir

## 10 Ein- Ausgabe

Kanal / Dateihandle	Name	Kommentar
Standardeingabe	STDIN	kann mit open geändert werden
Standardausgabe	STDOUT	kann mit open geändert werden
Standardfehlerausgabe	STDERR	kann mit open geändert werden
Diamond-Operator	<>	liest aus allen Dateien, die in der Kommandozeile angegeben sind, falls keine angegeben, von STDIN.
Datei lesend öffnen	open (IN, '<datei') or die ...	
Datei schreibend öffnen	open (OUT, '>\$datei') or die ...	
Datei schließen	close ( IN );	
in Datei schreiben	print OUT "Text\n";	
von Datei lesen	while ( defined(\$line = <IN> ) ) { ... };	

### 10.1 open-Modus einer Datei

Dateiname	Lesen	Schreiben	Anhängen	Erzeugen	Abschneiden
<Datei	ja	nein	nein	nein	nein
>Datei	nein	ja	nein	ja	ja
>>Datei	nein	ja	ja	ja	nein
+<Datei	ja	ja	nein	nein	nein
+>Datei	ja	ja	nein	ja	ja
+>>Datei	ja	ja	ja	ja	nein

### 10.2 Operationen auf Dateihandles

Aktion	Syntax / Beschreibung
Dateizeiger bestimmen	tell HANDLE
	liefert die aktuelle Position in Bytes ( 0-basiert )
Dateizeiger positionieren	seek HANDLE, OFFSET, VON_WO_CODE
VON_WO_CODE = 0	positioniert vom Beginn der Datei
VON_WO_CODE = 1	positioniert relativ zur aktuellen Position
VON_WO_CODE = 2	positioniert relativ zum Dateieinde
auf Dateieinde prüfen	eof HANDLE eof () # prüft für <>

## 7.13 Sonstige Operatoren

Operator	Beispiel	Bemerkung, Ergebnis bei \$a=122, \$b=3, \$c='Hallo'
..	( \$b .. \$a )	( 3, 4, 5, ... 120, 121, 122 )
,	( \$a, \$b, \$c )	( 122, 3, 'Hallo' )
=>	( \$a => \$b, \$c ) ( er => 'he', sie => 'she' )	( 122, 3, 'Hallo' ) ( 'er' => 'he', 'sie' => 'she' );
->	\$struct->{'componet'}	deferenzieren
? .. :	\$a = \$b > 5 ? \$b : \$c;	if ( \$b > 5 ) { \$a = \$b; } else { \$a = \$c; };

## 8 Kontrollstrukturen

Operation / Modifikator	Syntax, Beispiel
if	if ( AUSDR ) BLOCK;
unless	unless ( AUSDR ) BLOCK; # if ( not AUSDR ) BLOCK;
if - else	if ( AUSDR ) BLOCK else BLOCK;
if - elsif else	if ( AUSDR ) BLOCK elsif ( AUSDR ) BLOCK ... else BLOCK;
while	LABEL: while ( AUSDR ) BLOCK;
for	LABEL: for ( AUSDR; AUSDR; AUSDR ) BLOCK;
foreach	LABEL: foreach VAR ( LIST ) BLOCK;
next	next LABEL;
last	last LABEL;
redo	redo LABEL;
continue	LABEL: while ( AUSDR ) BLOCK continue BLOCK; LABEL: foreach VAR ( LIST ) ontinue BLOCK;

### 8.1 Modifikatoren von Ausdrücken

Modifikator	Beispiel
ANWEISUNG if AUSDR	schlafe_bis ( 10.00 ) if ( \$heute eq 'So' );
ANWEISUNG unless AUSDR	schlafe_bis ( 7.00 ) unless ( \$heute eq 'So' );
ANWEISUNG while AUSDR	push ( @text, \$line ) while ( defined(\$line = <> ) );
ANWEISUNG until AUSDR	kiss ( 'me' ) until ( \$_!_die );

## 9 Referenzen

### 9.1 Einfache Referenzen

Referenzen	Syntax
Auf Variable	<code>\$R_Skalar = \ \$Skalar;</code> <code>\$R_Liste = \@Liste;</code> <code>\$R_Hash = \%Hash;</code>
Auf Unterprogramme	<code>\$R_Unterprogramm = \&amp;Unterprogramm;</code>
Anonyme Listen	<code>\$R_A_Liste = [ 'Anfang', 'Ende' ];</code>
Anonyme Hashes	<code>\$R_A_Hash = { 'ich' =&gt; 'I' };</code>
Anonyme Unterprogramme	<code>\$R_A_Unterprogramm = sub { print 'Hallo\n' };</code>
Zugriff auf Skalare	<code>\$Erg = \$\$R_Skalar;</code>
Zugriff auf Listen	<code>@Erg = @{ \$R_A_Liste };</code>
Zugriff auf Hashes	<code>%Erg = %{ \$R_A_Hash };</code>
Unterprogrammaufruf über Referenzen	<code>&amp;\$R_Unterprogramm ();</code> <code>&amp;\$R_A_Unterprogramm ();</code>

### 9.2 Datenstrukturen mit Referenzen

Referenzen	Beschreibung
Referenzen	<code>%Adam_Meier = ( 'Name' =&gt; 'Adam Meier' );</code> <code>%Kurt_Meier = ( 'Name' =&gt; 'Kurt Meier',</code> <code>                  'Vater' =&gt; \%Adam_Meier</code> <code>                  'Mutter' =&gt; { 'Name' =&gt; 'Meta Meier' }</code> <code>);</code>

### 9.3 Referenzen auf Unterprogramme

Referenzen	Beschreibung
explizite	<code>my \$command = \&amp;test_rate_plotter;</code>
implizite	<code>my \$command = sub { exit; };</code>

### 9.4 Anonyme Referenzen

Referenzen	Beschreibung
Anonyme Liste	<code>\$ano_list = [ 'Ingeborg', 'Max' ];</code>
Anonymer Hash	<code>\$ano_hash = { 'Name' =&gt; 'Fred', 'Frau' =&gt; 'Wilma' };</code>
Zugriff auf anonyme Liste	<code>\$ano_list-&gt;[0] = 'Herta';</code>
Zugriff auf anonymen Hash	<code>\$ano_hash-&gt;{'Frau'} = 'Berta';</code>
Komplexe Struktur	<code>\$list_of_lists = [ [1, 1], [1, 2], [2, 1], [2, 2] ];</code>

## 9.5 Komplexe Datenstrukturen

Struktur	Beschreibung
Listen von Listen	<code>@Pairs = ( [ 'Ingeborg', 'Max' ],</code> <code>          [ 'Simone', 'Jean-Paul' ],</code> <code>          [ 'Cleopatra', 'Caesar' ]</code> <code>);</code> <code>@A_Pair = @ { \$Pairs[1] };</code> <code>\$A_Person = \$Pairs[1][0];</code>
anonyme Listen von Listen	<code>\$Friends = [ [ 'Bert', 'Ernie' ],</code> <code>          [ 'Hans', 'Rita', 'Hauke' ]</code> <code>];</code> <code>@The_Friends = @ { \$Friends-&gt;[1] };</code> <code>\$A_Person = \$Friends-&gt;[1][2];</code>
Matrizen	<code>@Matrix = ( [00, 01, 02],</code> <code>          [10, 11, 12],</code> <code>          [20, 21, 22]</code> <code>);</code> <code>\$x = \$Matrix[1][2];</code> <code>\$x = \$Matrix[1]-&gt;[2];</code> <code>\$x = \${\$Matrix[1]}[2];</code>
anonyme Matrizen	<code>\$Matrix2 = [ [00, 01, 02],</code> <code>          [10, 11, 12],</code> <code>          [20, 21, 22]</code> <code>];</code> <code>\$x = \$\$Matrix2[1][2];</code> <code>\$x = \$Matrix2-&gt;[1]-&gt;[2];</code> <code>\$x = \$Matrix2-&gt;[1][2];</code>
Arrays von Hashes	<code>@Persons = ( { 'Name' =&gt; 'Fred',</code> <code>              'Frau' =&gt; 'Wilma' },</code> <code>              { 'Name' =&gt; 'Homer',</code> <code>              'Frau' =&gt; 'Marge',</code> <code>              'Sohn' =&gt; 'Bart' }</code> <code>);</code> <code>%A_Person = % { \$Persons[1] };</code>
Hashes von Arrays	<code>%Families = ( 'flintstones' =&gt; [ 'Fred',</code> <code>                                  'Wilma' ],</code> <code>              'simpsons' =&gt; [ 'Homer',</code> <code>                                  'Marge',</code> <code>                                  'Bart' ]</code> <code>);</code> <code>@A_Family = @ { \$Families{'simpsons'} };</code> <code>\$A_Person = \$Families{'simpsons'}-&gt;[0];</code>
Hashes von Hashes	<code>%Persons = ( 'Fred' =&gt; { 'wife' =&gt; 'wilma' },</code> <code>              'Homer' =&gt; { 'wife' =&gt; 'Marge',</code> <code>                          'son' =&gt; 'Bart' }</code> <code>);</code> <code>%A_Person = % { \$Person{'Fred'} };</code> <code>\$Person{'Fred'}-&gt;{'son'} = 'Pebbles';</code>
anonyme Hashes von Hashes	<code>\$Persons-&gt;{'Fred'} = { 'wife' =&gt; 'wilma' };</code> <code>\$Persons-&gt;{'Homer'} = { 'wife' =&gt; 'Marge',</code> <code>                      'son' =&gt; 'Bart' };</code> <code>%A_Person = % { \$Persons-&gt;{'Fred'} };</code> <code>\$Persons-&gt;{'Fred'}{'son'} = 'Pebbles';</code>
anonyme Hashes von Hashes mit Arrays	<code>\$Persons = { 'Fred' =&gt; { 'wife' =&gt; 'Wilma',</code> <code>                      'children' =&gt; [],</code> <code>                      'friends' =&gt; [ 'Barney' ] },</code> <code>              'Homer' =&gt; { 'wife' =&gt; 'Marge',</code> <code>                      'children' =&gt; [ 'Bart' ]</code> <code>};</code> <code>%A_Person = % { \$Persons-&gt;{'Fred'} };</code> <code>push ( @ { \$Persons-&gt;{'Fred'}{'children'} }, 'Pebbles' );</code>