Perl-Referenzkarte V2.6

(http://www.ostc.de/perl-refcard.pdf)

$kalar (*Zahl, String/Zeichenkette*)

```
$v = undef          # Automatisch Std.wert
$n = 1              $m = 314.1516E-2
$s = "Hallo"        $s = qq/Hallo/
$s = 'Hallo'        $s = q/Hallo/
$t = "$s Welt!"     $t = "${s} Welt!"
```

Konvertierung je nach Bedarf automatisch:
Zahlpräfix von String in Zahl, Zahl in ident. String, Skalar in 1-elem. Array, Array in Skalar (Länge), Hash in Array, Array in Hash, usw.

***Einbetten** von Skalaren, Arrays, Arrayelem/slices, Hashelem/slices (nicht Hashes) in Strings **möglich**.*

Numerische Operatoren

```
$c = $a + $b    $c += 2    # Verkürzte
$c = $a - $b    $c -= 2    # Operationen
$c = $a * $b    $c *= 2
$c = $a / $b    $c /= 2
$rest = $a % $teiler       # Modulo
$c = $a ** $b   $c **= 2   # Potenz
++$a   $a++   --$a   $a--  # Inc/Dec
```

***Präfix/Postfix** ändert **vor/nach** Zuweisung um 1.*

String-Operatoren

```
$text = $n . " Euro"   # Dot-Operator
$text .= "mehr text"   #  = Verketten

$linie = "-" x 80      # Times-Operator
$linie x= 10           # Vervielfachen

$text <<"EOF"          # Here-Operator
Text mit " und Variable $a und
Zeilenumbruch bis EOF alleine
EOF                    # ";" notwendig!

sprintf(FMT,@l)   # String analog printf
```

```
chomp($text)      # Entfernt "\n" am Ende
chop($text)       # Entf. letztes Zeichen
length($text)     # Stringlänge
```

Vergleiche (*String und numerisch*)

```
$s eq "Hallo"     # Stringvergleich
$n == 12345       # Zahlenvergleich
```

String: `ne gt ge lt le cmp` *# 1/0/-1*
Zahl: `!= > >= < <= <=>` *# Space-Ship*
*Vergleich **erzwingt** Datentyp der Operanden.*
`cmp` *und* `<=>` *nur für Sortieren mit* `sort` *relevant.*

Boolesche Werte und Operatoren

False = `0 "0" "" () undef` *# 5 Werte!*
True = ***Alle*** *anderen Werte (insbesondere* `1`*!)*

```
not $a          # Logisch Nicht
! $a            # Logisch Nicht
$a and $b       # Logisch Und
$a && $b        # Logisch Und
$a or $b        # Logisch Oder
$a || $b        # Logisch Oder
$a xor $b       # Log. Entweder-Oder
defined($v)     # Wahr w. $v ≠ undef
```

Achtung: `or and xor not` *haben niedrigsten Vorrang, daher Klammern bei Zuweisung!*

Short-Circuit Auswertung: `or` *(*`||`*) und* `and` *(*`&&`*) liefern Wert des **letzten** ausgewerteten Ausdrucks:*

```
$plz = $val{"plz"} or 90425
$dbg = $val{"dbg"} and $val{"lev"}
open(IN, $file) or
    die("$file nicht offen: $!\n")
```

@rray (*Liste, indiziert, geordnet*)

```
@a = (1, "abc", undef)
@a = qw/ Tom Hans Rick / # quote word
@a = ("A" .. "Z")    @a = (1 .. 99)
@a = (1) x 10        # Array mit 10 Elem.
```

Dot-Operator `..` *zählt Bereich von...bis auf.*

Array-Elemente (*Index 0 bis Anz.Elem.-1*)

```
$a[0]=1  $a[1]="abc"  $a[2]=undef
```

Array-Länge/-Index (*hier: 3 bzw. 2*)

```
$anz = @a  oder  @a+0    # Skalarer Kontext!
$anz = scalar @a         # Analog
$last = $#a  oder  -1    # Index letztes Elem.
```

Array-Operationen

```
foreach (@a) {...}          # Laufvar. $_
foreach my $e (@a) {...}    # Laufvar. $e
push(@a, "ende")  $ende = pop(@a)
unshift(@a,"anf")  $anf = shift(@a)
@part = splice(@a,1,2)      # Hans Rick
@alt = splice(@a,$ofs,$len,@repl)
@b = sort(@a)      @b = reverse(@a)
@a = split(/:/, "aa:bb:cc")
$s = join("-", @a)          # "aa-bb-cc"
```

Hash (*%, Assoziatives Arr., ungeordnete Paare*)

```
%age = ("Tom", 18, "Hans", 52, ...)
%age = ("Tom" => 18, "Hans" => 52)
```

Hash-Elemente (*ungeordnet!*)

```
$age{"Rick"} = 36
if (exists($age{"Hans"})) {...}
delete $age{"Rick"}
```

Hash-Schlüssel / -Werte (*ungeordnet!*)

```
@keys = keys %age      # Alle Schlüssel
@vals = values %age    # Alle Werte
```

Schleife über Hash-Elem. (*2* unsort., 2* sort.*)

```
foreach (keys %age) {...}
while (($k,$v) = each(%age)) {...}
foreach (sort keys %age) {...}
foreach (sort { $age{$a} <=>
          $age{$b} } keys %age){...}
```

Bezeichner: *Gross/Klein + getr. Namensräume*

`$a` ≠ `@a` ≠ `%a` ≠ `&a` ≠ `$A` ≠ `@A` ≠ `%A` ≠ `&A`

Funktions-Aufruf (*&*)

```
print &FUNC(10, "Text")
```

Reihenfolge *von Definition und Aufruf egal.*

Funktions-Definition

```
sub FUNC {                      # ... Name!
    my ($nr, $text) = @_;       # Kopie +
    KMDO;
    if (wantarray) {
        #  Fkt. im Listenkontext aufgerufen
    } else {
        #  Fkt. im Skalaren Kontext aufgerufen
    }
    return $erg oder @erg;
}
```

Parameter *"namenlos" und "call-by-reference" im Array* `@_` *übergeben (*`my` *= lokale Variable)*

Verzweigungen

```
if (BED1) {
    KMDO1
} elsif (BED2) {    # beliebig oft wiederh.
    KMDO2
} else {
    KMDOn
}

unless (BED) {      # if (not BED)
    KMDO1
} else {
    KMDO2
}
```

Bedingung / Schleife als Modifikator

```
KMDO if (BED)          # Klammer darf fehlen
print "Hier" if ($debug)
KMDO unless (--$zaehler == 0)
KMDO foreach (@person)
KMDO while ($zaehler != 0)
KMDO until ($text eq "STOP")
```

Eine *Anw.* `KMDO` ***ohne*** *Block-Klammern* `{...}`*!*

Bedingter Ausdruck (*Dreiwertiger Operator*)

```
$erg = ($x == 1) ? "An" : "Aus"
```

Entspricht

```
if ($x == 1) {
    $erg = "An"
} else {
    $erg = "Aus"
}
```

Sprungmarken (*nur Großbuchstaben!*)

```
LABEL: KMDO
goto / next / last / redo LABEL
```

Damit auch mehrstufige Schleifen abbrechbar.

Zählschleifen (*Liste abarbeiten*)

```
foreach my $e (LISTE) { # $e enth. Elem.
    KMDO
}

foreach (split(/\s/, $text)) {
    KMDO                # $_ enth. Worte
}

foreach my $i (1..10) { # Bereichs-Op.
    KMDO                # $i enth. Zähler
}

for (my $i = 1; $i <= 10; ++$i) {
    KMDO      # Analoge Zählschleife von 1..10
}
```

Bedingte Schleifen (*kopf/fußgesteuert*)

```
while (BED) {          until (BED) {
    KMDO                   KMDO
}                      }

do {                   do {
    KMDO                   KMDO
} until (BED)          } while (BED)
```

`until (BED)` *entspricht* `while (! BED)`

Schleifenabbruch/wiederholung (*innerste*)

```
LABEL: while (BED) {
    KMDO;
    next if BED;        # continue in C
    last if BED;        # break in C
    redo LABEL if BED;  # nicht in C
    KMDO;
}
continue { KMDO }       # bei next + normal
```

Programm-Fehler (*bzw. Warnung*)

```
die("Fehler")      # "\n" od. Datei + Zl.nr
warn("Warnung")    # "\n" od. Datei + Zl.nr
exit(2)            # Exit-Status 0-255
```

`die` + `exit` *brechen ab, Ausgabe auf* `STDERR`.

Standard-Ein/Ausgabe lesen / schreiben

```
$zeile = <STDIN>       # Std-Eingabe (1 Zl.)
@zeilen = <STDIN>      # Std-Eing. (alle Zl.)
$zeile = <>            #  Diamond (1 Zl.)
@zeilen = <>           # UNIX-Filter (alle Z.)
print "text"           # Standard-Ausgabe
print STDOUT "text"    # Standard-Ausgabe
print STDERR "bla"     # Standard-Fehler
select STDERR          # Std.Handle wählen
```

Formatierte Ausgabe

```
printf("%s %d %f", "abc", 2, 3.14)
```

```
%d      # Dezimal (abgeschnitten!)
%04d    # 4 Stellen mit führenden 0
%f      # Fließkomma (gerundet!)
%8.2f   # 8 Stellen gesamt, 2 Nachkommast.
%s      # String
%-10s   # Min. 10 Stellen breit + linksbündig
%4.4s   # Min. + max. 4 St. breit + rechtbünd.
%c      # Zeichen (character)
%%      # Prozent-Zeichen
%b %o %x  # Binär/Oktal/Hexadezimal-Zahl
```

Dateien öffnen / lesen / schreiben / schließen

```
open(IN,  "<",  "lesen")     or die
open(OUT, ">",  "schreiben") or die
open(OUT, ">>", "anhängen")  or die

$zeile = <IN>          # 1 Zeile lesen
@zeilen = <IN>         # Alle Zeilen l.
foreach my $zeile (<IN>) {...}
$zeile = readline(*IN) # Ersatz für <IN>
while (<IN>) {...}     # $_ st. $zeile
$cnt = read(IN, $str, $len)
$ok = seek(IN,$pos,0/1/2=abs/rel/end)
$pos = tell(IN)        # -1 bei Fehler
eof(IN)                # Ende erreicht?

print OUT "text"       # kein ","!
printf(OUT "%s%d%f", "ab", 2, 3.14)
close(IN)  close(OUT)  # Schließen
```

Von/auf Prozesse lesen / schreiben (*Pipe*)

```
open(IN, "KMDO |")     # Von KMDO lesen
open(OUT, "| KMDO")    # In KMDO schreiben
$erg = qx/ KMDO /      # KMDO ausführen +
$erg = `KMDO`          # Erg. abfang. (1 Zl.)
@erg = `KMDO`          # Analog (alle Zl.)
$exit=system("KMDO")   # Erg. nicht abfang.
$err = exit("KMDO")    # Analog+nicht zur.
```

Dateien bearbeiten

```
@info = stat("file")       # 13 Datenelem.
@info = lstat("file")      # Link analog
link("alt", "neu")         # Hardlink
symlink("alt", "neu")      # Symbol. Link
readlink("link")           # Ziel v. Symlink
$n=unlink("weg", @alt)     # Löschen
$n=chmod($mode, @files)    # oct(...)!
$n=chown($uid,$gid,@files) # Besitz(gr.)
$n=utime($acs,$chg,@files) # Zeiten
rename("alt", "neu")       # Umbenennen
truncate($file, $len)      # Abschneiden
flock($f,$mode)  # 1/2/4/8=SH/EX/NB/UN
```

Verzeichnisse bearbeiten

```
@files = glob "*"        # Filename-
@baks  = glob "*.bak"    # Globbing

chdir("dir")             # Arb.verz. wechs.
mkdir("dir", 0777)       # Verz. erzeugen
$n = rmdir(@dirs)        # Verz. entf. (leer!)
opendir(DIR, "dir")      # Verz. öffnen
$elem = readdir(DIR)     # 1 Eintrag lesen
@elems = readdir(DIR)    # Alle Einträge les.
$pos = telldir(DIR)      # Verz.Pos. lesen
seekdir(DIR, $pos)       # Verz.Pos. setzen
rewinddir(DIR)           # Verz.Pos. Anfang
closedir(DIR)            # Verz. schließen
```

Datei-Tests

```
-e DATEI   # Datei existiert (exists)
-f DATEI   # Normale Datei (file)
-d DATEI   # Verzeichnis (directory)

-b DATEI   # Blockgerät (UNIX)
-c DATEI   # Zeichengerät (character, UNIX)
-l DATEI   # Symbol. Link/Verknüpfung (UNIX)
-S DATEI   # Socket (UNIX, großes S!)
-p DATEI   # Named Pipe/FIFO (UNIX)
-t DATEI   # Terminal (UNIX)

-z DATEI   # Datei ist leer (zero)
-s DATEI   # Grösse der Datei (size)

-M DATEI   # Dateialter in Tagen (modified)
-A DATEI   # Dateizugriff in Tagen (access)
-C DATEI   # Unix: Inode-Change; Win: Create

-r DATEI   # Datei lesbar (readable)
-w DATEI   # Datei schreibbar (writable)
-x DATEI   # Datei ausführbar (executable)

-R DATEI   # Datei lesbar (realer Benutzer)
-W DATEI   # Datei schreibbar (realer Benutzer)
-X DATEI   # Datei ausführbar (realer Benutzer)

-u DATEI   # Set-UID-Recht gesetzt
-g DATEI   # Set-GID-Recht gesetzt
-k DATEI   # Sticky-Recht gesetzt

-o DATEI   # Datei gehört effekt. UID (owner)
-O DATEI   # Datei gehört realer UID (Owner)

-T DATEI   # ASCII-Textdatei (Heuristik)
-B DATEI   # Binärdatei (Heuristik)
```

Reguläre Ausdrücke (*Regex*)

`.`	1 beliebiges Zeichen (*außer* `\n`!)
`[abc]`	1 Zeichen aus Menge `abc`
`[^abc]`	1 Zeichen *nicht* aus Menge `abc`
Z`*`	Zeichen *Z* beliebig oft (*auch 0-mal!*)
Z`+`	Zeichen *Z* ein- oder mehrmals
Z`*?` *Z*`+?`	Wie `*`/`+` (*matcht möglichst wenig!*)
Z`?` *Z*`??`	Zeichen *Z* *0-1*-mal (*mögl. wenig!*)
Z`{`*n*`,`*m*`}`	Zeichen *Z* *n* bis *m*-mal
Z`{`*n*`,`*m*`}?`	Analog (*matcht möglichst wenig!*)
Z`{`*n*`,}`	Zeichen *Z* *n* bis ∞-mal
Z`{`*n*`}`	Zeichen *Z* genau *n*-mal
`^` `$`	Anfang / Ende **String** (*Opt.* `m`→*Zeile*)
`\A` `\z`	Anfang / Ende **String**
`\d` `\D`	Ziffer / keine Ziffer (*0-9 digit*)
`\w` `\W`	**W**ortzeichen / kein Wz. (*0-9_a-zA-Z*)
`\s` `\S`	Leerr. `[ \t\n\r\f]`/ kein (*space*)
`\b` `\B`	Wortgrenze / keine Wortgr. (*break*)
A`|`*B*	*A* oder *B* passt
`(...)`	Klammern / Merken für später
`$1` `\1`	Was 1/2/... Klammer matchte (`\1` *alt*)

Pattern Matching und Substitution

```
$a = "Hallo Hallo Hallo"
if ($a =~ m/Ha/) {     # match
    print "Ha gefunden"
}

$a =~ s/Ha/he/         # substitute
# danach $a eq "hello Hallo Hallo"

$a =~ s/Ha/he/g        # substitute global
# danach $a eq "hello hello hello"
```

Regex-Trick: Andere Zeichen statt //

```
$a = "<H2><B>hallo</B></H2>"
$a =~ s@</B>@@        # einzeln
$a =~ s(</B>){}       # paarweise
```

Regex-Modifizierer

```
s/.../.../g    # Mehrfach (global)
s/.../.../i    # Gross/Kleinschr. (ignorecase)
s/.../.../s    # Singleline (\n mit in .drin)
s/.../.../m    # Multiline (^$ = Zeile)
s/.../.../x    # Extended (#-Komment.+Whitesp.)
```

Regex-Beispiele

```
s/^\s#.*//              # Kommentar entf.
s/^\s+//                # Leerr. am Zl.anf. entf.
s/\s+$//                # Leerr. am Zl.end. entf.
s/^\s+|\s+$//           # L. links+rechts entf.
s/<\/?B>//ig            # <B> + </B> entf.
s/<\/?FONT.*?>//ig      # FONT-Tag entf. (v1)
s/<\/?FONT[^>]*>//ig    # Analog (v2)
```

Posix Zeichenklassen (*Auswahl*)

```
[:alpha:]   # Buchstaben
[:alnum:]   # Buchstaben + Ziffern
[:ascii:]   # ASCII-Zeichen
[:blank:]   # Leerzeichen (White[:space:])
[:ctrl:]    # Steuerzeichen
[:digit:]   # Ziffern
[:lower:]   # Kleinbuchstaben
[:punct:]   # Interpunktionszeichen
[:upper:]   # Großbuchstaben
```

Standard-Handles (*bereits geöffnete Dateien*)

```
STDIN     # Standardeingabe (Tastatur)
STDOUT    # Standardausgabe (Bildschirm)
STDERR    # Fehlerkanal (Bildschirm)
ARGV      # Kommandozeilenargumente
ARGVOUT   # Akt. geöff. Datei bei Inplace-Ed.
DATA      # Daten ab __END__/__DATA__
```

Standard-Variablen

```
@ARGV       # Aufrufparameter (für <>)
$ARGV       # Aktueller Dateiname (für <>)

%ENV        # Umgebungsvar. (environment)
%SIG        # Signale abfangen

@INC        # Include-Verz. für Module
%INC        # Geladene Module (Name → Pfad)

@ISA        # Basisklassen eines Pakets ( is a )
@EXPORT     # Meth. + Symb. stdm. exportiert
@EXPORT_OK  # Meth. + Symb. auf Anford. exp.

@F          # Felder bei -a (-F bzw. $FS)
$_          # Std.var. für viele Funktionen
@_ @ARG     # Argumentliste pro Funktion
$a $b       # Zu vergl. Elem. in sort-Vgl.Fkt.
$AUTOLOAD   # Name der undef. aufgeruf. Fkt.

$0          # Programmname (inkl. Pfad)
$$          # Prozessnummer

$@          # Fehler nach eval
$!          # Fehler nach I/O-Op. (Nr + Text)
$?          # Fehler extern Pgm (Exit-Status)

$\          # Abschluss print (Std: leer)
$/          # Abschluss <...> (Std: \n)
$,          # Trennz. print-Elemente (leer)
$"          # Tz. eingebettete Array-Elem. (Sp)

$1 $2 ...   # Regex: Was 1/2/... Klam. matchte
$`          # Regex: Teil links von Match
$&          # Regex: Match
$'          # Regex: Teil rechts von Match

$.          # Aktuelle Zeilennummer
$|          # 1=Autoflush, 0=Pufferung

$<          # UID realer Benutzer
$>          # UID effektiver Benutzer
$(          # GID reale Gruppe(n)
$)          # GID effektive Gruppe(n)

$^O         # Betriebssystem (operating sys.)
```

Standard-Konstanten (*2 Unterstriche!*)

```
__FILE__      # Akt. Perl-Skriptname
__LINE__      # Akt. Perl-Skriptzeile
__PACKAGE__   # Akt. Perl-Modulname
__DATE__      # Akt. Datum
__TIME__      # Akt. Uhrzeit
__END__       # Programmende (<DATA>)
__DATA__      # Programmende (<DATA>)
```

Referenzen (*immer ein Skalar!*)

```
\$s             # Ref. auf Skalar
\@a             # Ref. auf Array
\%h             # Ref. auf Hash
\&f             # Ref. auf Funktion

$a = [...]      # Ref. auf anonymes Array
$h = {...}      # Ref. auf anonymen Hash
$f=sub{...}     # Ref. auf anonyme Funktion

$$s  ${$s}      # Ref. auf Skalar dereferenz.
@$a  @{$a}      # Ref. auf Array dereferenz.
%$h  %{$h}      # Ref.z auf Hash dereferenz.
&$f  &{$f}      # Ref. auf Funktion dereferenz.

$a->[0]         # Ref. auf Array → Arrayelem.
$h->{Tom}       # Ref. auf Hash → Hashelem.

ref $x          # Ref.typ ermitteln (Classname
                # SCALAR ARRAY HASH
                # CODE REF GLOB LVALUE)
```

Skriptaufbau

```
#!/usr/bin/perl -w   # Shee-Bang-Zeile
use warnings         # Warnungen zusätz.
use strict           # Var.dekl. erzwingen
```

Module

```
use modul        # modul.pm laden + akt.
no modul         # modul.pm deaktivieren
```

Moduldatei `modul.pm` *enthält Variablen, Code, Funktionen und wird geladen.* ***Muss*** *mit* `1;` *enden.*