

Jürgen Plate

# Perl-Funktionen und -Module

Supplement zum Buch:  
Jürgen Plate  
Der Perl-Programmierer  
Erschienen im Hanser-Verlag



# Inhaltsverzeichnis

<b>1 Perl-Funktionen</b>	<b>5</b>
<b>2 Perl-Standardmodule</b>	<b>17</b>
2.1 Allgemeines . . . . .	17
2.2 Bearbeitung von Text . . . . .	18
2.3 Berechnungen . . . . .	18
2.4 Dateiverwaltung . . . . .	18
2.5 Datum und Uhrzeit . . . . .	19
2.6 Prozesskommunikation . . . . .	19
2.7 Variablenbindung . . . . .	19
2.8 Modulverwaltung und objektorientierte Programmierung . . . . .	19
2.9 Systemnahe Module . . . . .	19
2.10 Netzwerk und CGI . . . . .	20
2.11 Verwalten von Perl-Erweiterungen . . . . .	20
2.12 Dokumentation mit POD . . . . .	20



# 1

## Perl-Funktionen

Dieser Text enthält eine kurze, alphabetisch geordnete Referenz der vordefinierten Perl-Funktionen. Beachten Sie jedoch, dass nicht alle hier aufgeführten Funktionen von allen Perl-Versionen und allen Betriebssystemen unterstützt werden.

**abs Wert** liefert den absoluten Wert von Wert zurück. Wert kann eine Zahl (wie -7) oder ein Ausdruck sein.

**accept Neuer-Socket, Socket** nimmt eine einkommende Socket-Verbindung an. Wurde die Verbindung hergestellt, wird die gepackte Adresse zurückgeliefert. Andernfalls lautet der Rückgabewert **falsch**.

**alarm Sek** sendet ein SIGALRM an das Programm, nachdem die in Sek übergebenen Sekunden verstrichen sind. Es kann immer nur ein Alarm aktiv sein. Wird diese Funktion aufgerufen, während noch ein anderer Alarm läuft, wird der laufende Alarm durch den aktuelleren Aufruf ersetzt. `alarm 0` stoppt die Alarmfunktion. Wird kein Parameter angegeben, wird der Wert in `$_` verwendet.

**atan2 Y, X** liefert den Arkustangens von X/Y in dem Bereich von  $-\pi$  bis  $\pi$  zurück.

**bind Socket, Address** bindet eine Netzwerkadresse an einen Socket. Address sollte dabei die gepackte Adresse für die verwendete Art von Socket enthalten. Ist die Funktion erfolgreich, wird **wahr** zurückgeliefert, andernfalls **falsch**.

**binmode Dateihandle** binmode übernimmt als Parameter ein Datei-Handle und gibt an, dass die Daten im Binärformat (und nicht im ASCII-Format) in das Datei-Handle geschrieben (beziehungsweise daraus gelesen) werden sollen. Unter Unix hat diese Funktion keine Auswirkung, aber für MS-DOS und andere archaische Plattformen ist sie wichtig. Die Funktion wird sofort nach dem Öffnen einer Datei aufgerufen.

**bless Referenz, Klasse** wird in der objektorientierten Perl-Programmierung verwendet, um die Referenz an das Paket Klasse zu binden. Wird die Klasse fortgelassen, erfolgt eine Zuweisung an das aktuelle Paket.  `bless` liefert die Referenz zurück, die mit  `bless` zugewiesen wurde.

**caller Ausdruck** liefert den Kontext des aktuellen Subrutinenaufrufs zurück. In einem skalaren Kontext liefert  `caller ()` den Namen des Pakets, aus dem die Subroutine aufgerufen wurde, im Listenkontext den Paketnamen, den Dateinamen des Programms und die Zeilennummer des Aufrufs. Wenn ein Ausdruck mit angegeben wird, liefert die

Funktion zusätzliche Informationen zur Verfolgung des Stacks. Ausdruck gibt an, um wie viele Aufruf-Frames der Stack, ausgehend vom aktuellen Frame, zurückgegangen werden soll.

**chdir Ausdruck** übernimmt einen Ausdruck als Parameter und versucht, das aktuelle Verzeichnis auf das im Ausdruck angegebene Verzeichnis zu setzen. Ohne Parameter versucht die Funktion, in das Home-Verzeichnis des aktuellen Benutzers zu wechseln.

**chmod Liste** ändert die Zugriffsberechtigungen für die im Parameter `Liste` aufgeführten Dateien. Das erste Element in `Liste` muss der numerische Modus für die Dateien sein (Oktalzahl ohne Anführungszeichen). Er sollte auch das SUID-Bit mit einschließen.

**chomp Variable** entfernt alle Zeilenenden, die mit `$/` übereinstimmen (die Variable, die das Eingabedatensatz-Trennzeichen, normalerweise ein Newline, enthält). Wenn diese Funktion im Absatzmodus aufgerufen wird, werden alle angehängten Newlines aus dem String entfernt.

**chop Variable** entfernt das letzte Zeichen eines Strings.

**chown Liste** ändert die Besitzrechte der Dateien, die in `Liste` stehen, für Benutzer und Gruppe. Zurückgeliefert wird die Anzahl der Dateien, für die die Besitzrechte erfolgreich geändert wurden. Die ersten beiden Elemente der Liste müssen die numerische UID und GID des neuen Benutzers und der neuen Gruppe sein. Normalerweise kann nur `root` die Besitzrechte ändern.

**chr Zahl** liefert das Zeichen aus der ASCII-Tabelle zurück, das dem Zahlencode entspricht, der der Funktion übergeben wurde.

**chroot Verzeichnis** ist identisch mit dem gleichnamigen Systemaufruf. Für das gerade ausführende Programm und alle Kindprozessen wird das angegebene Verzeichnis zum neuen Root-Verzeichnis. Pfade, die mit `/` anfangen, beginnen danach mit diesem Verzeichnis. Nur für `root`.

**close Dateihandle** schließt ein zuvor geöffnetes Datei-Handle. Sie liefert **wahr** zurück, wenn alle Operationen erfolgreich waren. Beachten Sie, dass alle Datei-Handles automatisch geschlossen werden, wenn ein Perl-Programm beendet wird.

**closedir Verzeichnis** schließt ein Verzeichnis, das mit der Funktion `opendir()` geöffnet wurde.

**connect Socket, Address** `connect` versucht, eine Verbindung zu einem entfernten Socket herzustellen. `Address` sollte die gepackte Adresse entsprechend dem Typ des Sockets enthalten. Die Funktion liefert **wahr** zurück, wenn sie erfolgreich war, im anderen Fall **falsch**.

**cos Ausdruck** liefert den Kosinus des Ausdrucks zurück.

**crypt Text, Salt** Diese Funktion wird dazu verwendet, um Strings auf die gleiche Weise zu verschlüsseln, wie Passwörter in einer Unix-Passwortdatei gespeichert werden. Die Funktion besitzt zwei Parameter: den zu verschlüsselnden String und einen Salt-Wert, mit dem der Verschlüsselungsalgorithmus initialisiert wird.

**defined Ausdruck** stellt fest, ob der Ausdruck den Wert `undef` besitzt (im Gegensatz zu 0, Newline oder einem anderen leeren Rückgabewert).

**delete Ausdruck** entfernt Elemente aus einem Hash. Der Ausdruck besteht aus Hash-Namen und Schlüssel.

**die Liste** übernimmt eine Liste als Parameter, gibt sie auf der Standardfehlerausgabe aus und beendet das Programm. Wenn die Liste nicht mit einem Newline endet, werden der Name des Programms und die Zeilennummer der Zeile, bei der die Ausführung unterbrochen wurde, zusammen mit einem Newline an die Ausgabe angehängt.

**do Block|Ausdruck** führt den übergebenen Block aus und liefert den Wert der letzten Anweisung in dem Block zurück. Wenn `do` zusammen mit einem Schleifenausdruck verwendet wird, wird der Block ausgeführt, bevor die Schleifenbedingung das erste Mal getestet wird. `do Ausdruck` stellt eine Möglichkeit dar, Code aus einer anderen Datei auszuführen. Der Ausdruck wird in diesem Falle als der Dateiname einer Perl-Datei interpretiert.

**dump Label** erzeugt einen Speicherauszug (dump) des Programms.

**each Hash** holt die Werte aus einem Hash für die Verarbeitung in einer Schleife. Im skalaren Kontext liefert `each` den Schlüssel für das nächste Element im Hash zurück (`while ($key = each %hash) . . .`). Im Listenkontext liefert `each` eine zweielementige Liste, die den Schlüssel und den Wert für das nächste Element im Hash enthält.

**eof Dateihandle** liefert **wahr** zurück, wenn beim nächsten Lesen aus `Dateihandle` die Dateiendemarke erreicht wurde oder das `Dateihandle` nicht geöffnet ist. Ohne Parameter wertet `eof` die zuletzt gelesene Datei aus. Erfolgt der Aufruf mit leeren Klammern als Parameter, erkennt `eof` das Ende einer Pseudodatei, die aus all den Dateien besteht, die auf der Befehlszeile angegeben wurden.

**eval Ausdruck** führt einen Ausdruck oder einen Codeblock aus, als ob es sich um ein eigenständiges Perl-Programm handelte. Es wird im Kontext des gerade laufenden Perl-Programms ausgeführt, sodass alle Variablen und anderen Werte des übergeordneten Programms weiter definiert sind, wenn die Ausführung von `eval` beendet ist. Der von `eval` zurückgelieferte Wert ist der Wert des zuletzt ausgewerteten Ausdrucks. Tritt bei der Ausführung ein Syntax- oder ein Laufzeitfehler auf, wird `undef` zurückgeben, und die Variable `$@` enthält die Fehlermeldung.

**exec Liste** führt einen Systembefehl anstelle des laufenden Programms aus. Einen Rückgabewert erhält das Programm nur, falls der Aufruf misslang. Besteht die Liste aus mehreren Elementen, verwendet `exec` den Systemaufruf `execvp` mit den Parametern in `Liste`. Ist der Parameter aus ein Skalar, wird er auf Shell-Metazeichen geprüft und gegebenenfalls durch `/bin/sh` ausgeführt.

**exists Ausdruck** prüft, ob ein bestimmter Schlüssel in einem Hash definiert ist, nicht jedoch, ob es zu dem Schlüssel auch einen Wert gibt.

**exit Ausdruck** wertet den Ausdruck aus und bricht das Programm sofort ab. Das Ergebnis des Ausdrucks wird als Fehlerwert an die Shell übergeben.

**exp Ausdruck** liefert den Wert  $e^{\text{Ausdruck}}$  zurück.

**fcntl Dateihandle, Funktion, Skalar** emuliert den Systemaufruf `fcntl`. Benötigt das Pragma `use fcntl`.

**fileno Dateihandle** liefert einen Dateideskriptor für einen gegebenen Datei-Handle zurück. Dies ist ein Integerwert, der die Datei identifiziert. Er kann beispielsweise für die Verwendung mit `select` genutzt werden. Ist das `Dateihandle` nicht geöffnet, wird `undef` zurückgeliefert.

**flock Dateihandle, Operation** ruft den gleichnamigen Systembefehl zum File-Locking für das `Dateihandle` auf.

**fork** erzeugt einen Kindprozess und liefert die Kind-PID an den übergeordneten Prozess zurück. Diese Funktion ist nur auf Unix-ähnlichen Plattformen implementiert.

**format** erzeugt Schablonen für die formatierte Ausgabe mit `write`.

**formline Muster, Liste** wird intern von den Formaten verwendet, um die Parameterliste entsprechend dem Muster zu formatieren.

**getc Datei** liefert das nächste Zeichen aus der Datei zurück. Wird der Datei-Parameter weggelassen, liefert `getc` das nächste Zeichen aus STDIN. Die Eingabe erfolgt nach wie vor gepuffert, also erst nach Eingabe eines Newline.

**getlogin** liefert die aktuelle Login-Information aus `/etc/utmp` zurück. Falls der Wert Null lautet, sollten Sie stattdessen `getpwuid()` verwenden.

**getpeername Socket** liefert die gepackte `sockaddr`-Adresse des anderen Endes der Socket-Verbindung zurück.

**getpgrp PID** liefert die Prozessgruppe für den spezifizierten Prozess zurück. Wird als PID 0 angegeben, wird die Prozessgruppe des aktuellen Prozesses zurückgeliefert.

**getppid** liefert die Prozess-ID des Elternprozesses zurück.

**getpriority Welche, Wer** liefert die Priorität für einen Prozess, eine Prozessgruppe oder einen Benutzer zurück (sofern implementiert).

**getsockname Socket** liefert die gepackte `sockaddr`-Adresse dieses Endes der Socket-Verbindung zurück.

**getsockopt Socket, Level, Option** liefert die angeforderte Option oder im Fehlerfall `undef` zurück.

**glob Ausdruck** liefert den Wert von Ausdruck mit Dateinamenerweiterungen zurück, wie sie auch unter einer Shell vorkommen könnten.

**gmtime Ausdruck** konvertiert eine Zeitangabe der `time`-Funktion in die Greenwich-Standardzeit (Greenwich Mean Time) und gibt sie als eine Liste mit neun Elementen zurück. Im skalaren Kontext liefert die Funktion die Zeit im `ctime`-Format zurück.

**goto LABEL** sucht die mit LABEL benannte Anweisung und fährt ab dort mit der Ausführung fort. Sie kann die Ausführung nicht mit Anweisungen fortsetzen, die sich in Blocks befinden und initialisiert werden müssen (z. B. Unterprogramme oder Schleifen).

**grep Ausdruck, Liste** durchsucht Listen und liefert alle Elemente in der Liste zurück, die einem bestimmten Muster (Ausdruck) entsprechen, für die also der Ausdruck **wahr** ist.

**hex Ausdruck** liest den Ausdruck als einen hexadezimalen String und liefert den Dezimalwert zurück.

**import Klasse Liste** ist keine vordefinierte Funktion. Sie wird vielmehr von Modulen implementiert, die Namen in andere Module exportieren wollen. `import` wird von `use` aufgerufen, wenn ein Modul in ein Perl-Programm geladen wird.

**index Sting, Substr, Pos** lokalisiert einen Teilstring in einem größeren String. Bei den Parametern handelt es sich um den zu untersuchenden String, den gesuchten Teilstring und optional die Position, ab der die Suche beginnen soll. Es wird die Position im String zurückgegeben, an der der Teilstring das erste Mal auftrat.

**int Ausdruck** liefert den ganzzahligen Anteil des Ausdrucks zurück.

---

**ioctl Dateihandle, Funktion, Skalar** implementiert den Systemaufruf `ioctl`. Gegebenenfalls ist `require "ioctl.ph"` aufzurufen, um die Funktionsdefinitionen für `ioctl` zu importieren.

**join Ausdruck, Liste** ist das Gegenstück zu `split`. Sie fasst die Elemente einer Liste in einem einzigen String zusammen. Der Inhalt des Ausdrucks wird als Begrenzer zwischen den Listenelementen im zurückgelieferten String verwendet.

**keys Hash** liefert ein Array zurück, das alle Schlüssel des angegebenen Hashes enthält.

**kill Signal Liste** sendet ein Signal an eine Liste von Prozessen (bzw. Prozessnummern). Es kann eine Signalnummer oder der Signalname (in Anführungszeichen) angegeben werden.

**last LABEL** veranlasst das direkte Verlassen der in LABEL angegebenen Schleife. Wird kein Wert für LABEL angegeben, wird die innerste Schleife verlassen.

**lc Ausdruck** konvertiert alle alphabetischen Zeichen (nicht unbedingt alle Umlaute) in einem String in Kleinbuchstaben.

**lcfirst Ausdruck** liefert den Wert des durch Ausdruck gegebenen Strings zurück, wobei das erste Zeichen in einen Kleinbuchstaben umgewandelt wurde.

**length Ausdruck** übernimmt einen String als Parameter und liefert die Länge des Strings in Bytes zurück.

**link Alte\_Datei, Neue\_Datei** Erzeugt einen Hardlink von `Alte_Datei` auf `Neue_Datei`.

**listen Socket, Size** erfüllt die gleiche Funktion wie der Systemaufruf `listen`. Die Warteschlangengröße `Size` kann in der Regel mit 5 belegt werden. `listen` liefert im Erfolgsfall **wahr** zurück und ansonsten **falsch**.

**localtime Ausdruck** konvertiert eine Zeitangabe der `time`-Funktion in die lokale Zeit und gibt sie als eine Liste mit neun Elementen zurück. Im skalaren Kontext liefert die Funktion die Zeit im `ctime`-Format zurück.

**log Ausdruck** liefert den natürlichen Logarithmus (Basis  $e$ ) des Ausdrucks zurück.

**lstat Dateihandle** entspricht der Funktion `stat`, nur wird der Dateistatus für einen symbolischen Link und nicht für die Datei, auf die der Link verweist, zurückgeliefert.

**map Ausdruck Liste** bearbeitet alle Elemente einer Liste. Als Ausdruck ist entweder ein Codeblock oder ein Ausdruck möglich, der in einer Schleife auf alle Listenelemente angewendet wird. Die Ergebnisse werden im Listenkontext zurückgegeben.

**mkdir Name, Modus** erzeugt ein neues Verzeichnis, dessen Name als erster Parameter angegeben wird. `Modus` legt die Zugriffsrechte im üblichen Oktalformat fest (reine Oktalzahl ohne Anführungszeichen mit SUID-Bit).

**msgctl ID, CMD, Arg** implementiert den Systemaufruf `msgctl`. Nur auf Maschinen mit System V IPC.

**msgget Schlüssel, Flags** implementiert den Systemaufruf `msgget`. Nur auf Maschinen mit System V IPC. Liefert die ID der Nachrichtenwarteschlange oder `undef` zurück.

**msgrcv ID, Var, Größe, Typ, Flags** implementiert den Systemaufruf `msgrcv`. Nur auf Maschinen mit System V IPC. Die Nachricht wird in der Variablen `Var` gespeichert. Liefert im Erfolgsfall **wahr** zurück, bei einem Fehler **falsch**.

**msgsnd ID, Message, Flags** implementiert den Systemaufruf `msgsnd`. Nur auf Maschinen mit System V IPC. Liefert im Erfolgsfall **wahr** zurück und bei einem Fehler **falsch**.

**next LABEL** Wenn das Programm in einer Schleife auf diesen Befehl stößt, springt die Programmausführung direkt zum nächsten Schleifendurchlauf.

**oct Ausdruck** liest Ausdruck als einen oktalen String ein und liefert den zugehörigen Dezimalwert zurück.

**open (Dateihandle, Modus, Ausdruck)** öffnet die im Ausdruck spezifizierte Datei und weist sie dem Dateihandle zu. Der Modus legt fest, ob die Datei zum Lesen (<), zum Schreiben (>) oder zum Lesen und Schreiben (< + bzw. > +) geöffnet wird. Um Daten an die Ausgabedatei anzuhängen, wird der Modus >> verwendet. Um einen Dateihandle mit einer Pipe anstatt der Standardeingabe oder -ausgabe zu öffnen, wird das Pipe-Zeichen (|) vor oder nach dem Dateinamen verwendet (ohne Modusangabe).

**opendir (Handle, Ausdruck)** öffnet das in Ausdruck spezifizierte Verzeichnis für die Eingabe und weist es dem Handle zu. Aus dem Verzeichnis-Handle kann dann die Liste der Einträge in dem Verzeichnis gelesen werden. Beachten Sie, dass sich der Namensbereich für Verzeichnis-Handles nicht mit dem von Datei-Handles überschneidet.

**ord Ausdruck** liefert den numerischen ASCII-Wert des ersten Zeichens des Ausdrucks zurück.

**pack Template, Liste** übernimmt eine Liste von Werten, packt sie in eine binäre Struktur und liefert den String zurück, der diese Struktur enthält. Template ist eine Liste von Zeichen, die die Reihenfolge und den Typ der Werte vorgibt:

- A Ein ASCII-String, der mit Leerzeichen aufgefüllt wird.
- a Ein ASCII-String, der mit Nullen aufgefüllt wird.
- b Ein Bit-String (aufsteigende Bitreihenfolge wie `vec()`).
- B Ein Bit-String (absteigende Bitreihenfolge).
- h Ein Hexadezimal-String (niedriger Nibble zuerst).
- H Ein Hexadezimal-String (hoher Nibble zuerst).
- c Ein vorzeichenbehafteter char-Wert (signed).
- C Ein vorzeichenloser char-Wert (unsigned).
- s Ein vorzeichenbehafteter short-Wert.
- S Ein vorzeichenloser short-Wert (16 Bit).
- i Ein vorzeichenbehafteter Integer-Wert.
- I Ein vorzeichenloser Integer-Wert (min. 32 Bit).
- l Ein vorzeichenbehafteter long-Wert (32 Bit).
- L Ein vorzeichenloser long-Wert.
- n Ein short-Wert in Netzwerkanordnung (big-endian).
- N Ein long-Wert in Netzwerkanordnung (big-endian).
- v Ein short-Wert in little-endian-Anordnung.
- V Ein long-Wert in little-endian-Anordnung.
- f Fließkommazahl mit einfacher Genauigkeit.
- d Fließkommazahl mit doppelter Genauigkeit.
- p Ein Zeiger auf einen nullterminierten String.
- P Ein Zeiger auf eine Struktur (String fester Länge).
- u Ein uu-codierter String.
- w Ein BER-komprimiertes Integer.
- x Ein Null-Byte.
- X Ein Byte nach vorne rücken (letztes Byte wird überschrieben).
- @ Null-Auffüllung bis zur absoluten Position.

---

Jedem Buchstaben kann eine Zahl folgen, die angibt, wie oft der Buchstabe wiederholt werden soll (\* steht für beliebig viele).

**pipe Lesehandle, Schreibhandle** öffnet eine Pipe von Lesehandle nach Schreibhandle.

**pop Array** entfernt das letzte Element in einem Array (verkürzt es also um ein Element) und liefert es als skalaren Wert zurück.

**pos Skalar** liefert die Position in Skalar zurück, an der die letzte m//g-Suche abgebrochen wurde.

**print Dateihandle Liste** gibt im Listenkontext übergebenen Daten auf der Standardausgabe aus, falls kein Dateihandle spezifiziert wurde. Andernfalls werden die Daten in die Datei geschrieben. Fehlt die Liste, wird der Inhalt von \$\_ ausgegeben. Zwischen dem Dateihandle und der Liste steht kein Komma!

**printf Dateihandle Liste** implementiert die C-Ausgabefunktion gleichen Namens.

**push Array, Liste** hängt ein Element an das Ende eines Arrays an (Verlängerung des Arrays). Sie können auch mehrere Werte an ein Array anhängen, Sie müssen nur eine Liste als Parameter angeben.

**quotemeta Ausdruck** liefert den Wert von Ausdruck zurück, wobei alle nichtalphanumerischen Zeichen mit Escape-Zeichen (Backslash) versehen sind.

**rand Ausdruck** liefert eine Zufallszahl zwischen 0 und Ausdruck. Wird der Ausdruck weggelassen, einen Wert zwischen 0 und 1.

**read Dateihandle, Skalar, Länge, Offset** liest eine beliebige Anzahl von Datenbytes von einem Datei-Handle in ein Skalar ein. Der erste Parameter spezifiziert das Dateihandle. Der Parameter Skalar definiert die Variable, der die Daten zugewiesen werden. Länge gibt an, wie viele Bytes gelesen werden, und der optionale Parameter Offset wird verwendet, wenn die Daten ab einer bestimmten Position abgelegt werden sollen.

**readdir Dateihandle** liest Einträge aus einem Verzeichnis, das mit opendir() geöffnet wurde. Im skalaren Kontext liefert sie den nächsten Eintrag im Verzeichnis zurück. Im Listenkontext liefert sie alle übriggebliebenen Einträge im Verzeichnis.

**readlink Ausdruck** liest den Wert eines symbolischen Links.

**recv Socket, Skalar, Länge, Flags** empfängt eine Nachricht auf einem Socket und speichert sie in der Variablen Skalar. Länge gibt die Anzahl der empfangenen Bytes an.

**redo LABEL** startet den aktuellen Schleifenblock neu, ohne die Testbedingung der Schleife neu zu bewerten. Wurde LABEL nicht angegebend, wirkt redo auf den innersten Block.

**ref Ausdruck** liefert wahr zurück, wenn Ausdruck eine Referenz ist, sonst falsch.

**rename Alternname, Neuername** ändert den Namen der Datei Alternname in Neuername.

**return Ausdruck** beendet die Ausführung von eval(), einem Unterprogramm oder einer do-Datei und liefert den Wert von Ausdruck zurück.

**reverse Liste** übernimmt einen skalaren Wert oder eine Liste als Parameter. Beim Skalar wird die Reihenfolge der Zeichen umgedreht. Im Listenkontext wird die Reihenfolge der Elemente in der Liste umgedreht.

**rewinddir Dateihandle** setzt das Handle für ein mit `readdir` geöffnetes Verzeichnis zurück auf den ersten Eintrag in diesem Verzeichnis.

**rmdir Dateiname** entfernt das in Dateiname spezifizierte Verzeichnis, wenn es leer ist.

**scalar Ausdruck** erzwingt die Auswertung des Ausdrucks in einem skalaren Kontext und liefert seinen Wert zurück.

**seek Dateihandle, Offset, Modus** setzt die Position in der Datei. Ist der Modus = 0, wird die Position auf Offset gesetzt. Ist Modus = 1 wird Offset zur aktuellen Position hinzuaddiert und bei Modus = 2 wird auf End-of-File + Offset positioniert (in diesem Fall ist Offset negativ).

**seekdir Dateihandle, POS** setzt die Position für die `readdir()`-Funktion. POS muss ein Wert sein, der von `telldir()` zurückgegeben wurde.

**select Dateihandle** Erfolgt der Aufruf ohne Parameter, liefert `select()` das Standard-Dateihandle zurück. Wird ein Dateihandle übergeben, wird dieses zum Standardhandle, an das alle Ausgaben gesendet werden.

**semctl ID, NUM, CMD, ARG** implementiert den System-V-IPC-Systemaufruf `semctl()` für Semaphore.

**semget Key, NUM, Größe, Flags** implementiert den System-V-IPC-Systemaufruf `semget()` und liefert die Semaphor-ID zurück (bei Fehler `undef`).

**semop Key, String** implementiert den System-V-IPC-Systemaufruf `semop()`, der Semaphore-Operationen wie Signalisieren und Warten ausführt.

**send Socket, Meldung, Flags, ZU** sendet eine Nachricht über ein Socket. Die Funktion verwendet die gleichen Flags wie der Systemaufruf `send()` und liefert im Erfolgsfall die Anzahl der gesendeten Zeichen zurück (bei Fehler `undef`).

**setpgrp PID, PGRP** setzt die Prozessgruppe für die spezifizierte PID. Wenn als PID 0 angegeben wird, wird die Prozessgruppe auf den aktuellen Prozess gesetzt.

**setpriority Art, Wer, Priorität** setzt die Priorität für einen Prozess, eine Prozessgruppe oder einen Benutzer.

**setsockopt Socket, Level, OPT-Name, OPT-Wert** setzt die spezifizierte(n) Option(en) für einen Socket. Bei Auftreten eines Fehlers wird `undef` zurückgeliefert. Verwenden Sie `undef` für OPT-Wert, um eine Option zu setzen, ohne einen Wert für diese Option anzugeben.

**shift Array** entfernt das erste Element aus einem Array und liefert es als skalaren Wert zurück. Das Array ist danach um ein Element kürzer als vorher.

**shmctl ID, CMD, ARG** implementiert den System-V-Systemaufruf `shmctl()`.

**shmget SCHLUESSEL, GROESSE, FLAGS** implementiert den System-V-Systemaufruf `shmget()`.

**shmread ID, VAR, POS, GROESSE** implementiert den System-V-Systemaufruf `shmread()`.

**shmwrite ID, STRING, POS, GROESSE** implementiert den System-V-Systemaufruf `shmwrite()`.

**shutdown Socket, Art** schließt eine Socket-Verbindung. Dabei wird die Art des Schließens vorgegeben.

**sin Ausdruck** liefert den Sinuswert des Ausdrucks.

**sleep Ausdruck** versetzt das Programm für die in *Ausdruck* definierten Sekunden in Schlaf; falls kein Parameter übergeben wird, für unbegrenzte Zeit. `sleep()` kann mit dem SIGALRM-Signal unterbrochen werden. Die Funktion liefert dann die Anzahl der tatsächlich geschlafenen Sekunden zurück.

**socket Socket, Domain, Typ, Protokoll** öffnet ein Socket, das mit dem Handle `Socket` verbunden ist. *Domain*, *Typ* und *Protokoll* entsprechend den Parametern des `socket()`-Systemaufrufs. Benötigt das Modul „Socket“.

**socketpair Socket1, Socket2, Domain, Typ, Paar** erzeugt ein Paar unbenannter Sockets in der spezifizierten Domain mit dem spezifizierten Typ. Im Erfolgsfall liefert die Funktion **wahr** zurück.

**sort Subr|Block Liste** sortiert die Einträge einer Liste. Rückgabewert ist eine sortierte Liste. Für den Vergleich kann ein Anweisungsblock oder ein Unterprogramm spezifiziert werden.

**splice Array, Offset, Länge, Liste** ist eine Universalfunktion, um Elemente in ein Array einzufügen, daraus zu entfernen oder Elemente darin durch neue Werte zu ersetzen. `splice` kann mit bis zu vier Parametern aufgerufen werden, von denen die letzten beiden optional sind. Der erste Parameter ist das Array, das bearbeitet werden soll. Der zweite Parameter legt die Position im Array fest, an der die Aktion stattfindet, der dritte Parameter ist die Anzahl der Elemente, die zu entfernen sind (fehlt dieser Parameter, werden alle Elemente vom Offset bis zum Ende des Arrays entfernt). Alle weiteren Parameter sind eine Liste von Elementen, die an der durch den Offset festgelegten Stelle eingefügt werden.

**split /Muster/, Ausdruck, Limit** zerlegt einen String in mehrere Teilstrings und gibt diese Teile als Liste zurück. Das Muster legt fest, an welcher Stelle der String zerlegt werden soll, der Ausdruck liefert den zu zerlegenden String und der optionale Parameter *Limit* gibt eine Obergrenze für die Anzahl von Listenelementen an, die zurückgegeben werden sollen.

**sprintf FORMAT, Liste** formatiert Ausgabestrings entsprechend den Konventionen für die C-Funktion `sprintf()`.

**sqrt Ausdruck** liefert die Quadratwurzel des Ausdrucks zurück.

**srand Ausdruck** initialisiert den Zufallszahlengenerator von Perl. Wenn Sie *Ausdruck* fortlassen, wird `srand(time)` verwendet.

**stat Dateihandle** liefert Informationen über die in *Dateihandle* angegebene Datei als Liste zurück. Die von `stat()` zurückgelieferten Daten umfassen:

- die Gerätenummer des Dateisystems,
- die Inode-Informationen der Datei,
- den Typ und die Zugriffsrechte,
- die Anzahl der harten Links auf die Datei,
- die UID und GID des Besitzers der Datei,
- den Geräte-Identifizierer (bei Gerätedateien),
- die Grösse der Datei in Bytes,
- die Zeiten seit dem letzten Zugriff, seit der letzten Änderung und seit Änderung der Inode-Struktur,
- die Blockgröße der Datei,

- die Anzahl der verwendeten Blöcke.

**substr** *Ausdruck*, *Offset*, *Länge* extrahiert eine Folge von Zeichen aus einem String, der als erster Parameter angegeben wird. *Offset* gibt die Position an, ab der mit dem Extrahieren der Zeichen begonnen werden soll, und der dritte, optionale Parameter die Anzahl der zu extrahierenden Zeichen.

**symlink** *AlteDatei*, *NeueDatei* erstellt einen symbolischen Link von *AlteDatei* auf *NeueDatei*.

**syscall** *Befehl*, *Liste* ruft den Systembefehl auf, der als erster Parameter angegeben wurde. Die Elemente der *Liste* werden dem Systembefehl als Parameter übergeben.

**sysopen** *DateiHandle*, *Dateiname*, *Modus* öffnet die angegebene Datei und verbindet sie mit *DateiHandle*. Wenn die Datei noch nicht existiert, wird sie erzeugt.

**sysread** *DateiHandle*, *Skalar*, *Länge*, *Offset* liest mithilfe des Systembefehls `read()` die Anzahl *Länge* Bytes aus der Datei in den *Skalar*-Parameter ein. Zurückgeliefert wird die Anzahl der eingelesenen Bytes oder `undef`, falls ein Fehler auftrat. Der optionale Parameter *Offset* ist nötig, wenn die eingelesenen Bytes nicht an den Anfang des Strings, sondern um den *Offset* verschoben eingetragen werden sollen.

**sysseek** *DateiHandle*, *Position*, *Mode* arbeitet ähnlich wie die `seek()`-Funktion. Jedoch wird hier der Systemaufruf `lseek()` anstelle von `fseek()` verwendet.

**system** *Liste* Ruft das in der *Liste* spezifizierte Kommando als Kindprozess auf.

**syswrite** *DateiHandle*, *Skalar*, *Länge*, *Offset* schreibt mithilfe des Systemaufrufs `write()` *Länge* Bytes aus der Variablen *Skalar* in die angegebene Datei. Zurückgeliefert wird die Anzahl der geschriebenen Bytes (bei Fehler `undef`).

**tell** *DateiHandle* liefert die aktuelle Position des Dateizeigers für die spezifizierte Datei zurück.

**telldir** *DateiHandle* liefert die aktuelle Position in dem angegebenen Verzeichnis.

**tie** *Variable*, *Klasse*, *Liste* bindet eine Variable an eine Paketklasse, die eine Implementierung für die Variable bereitstellt.

**tied** *Variable* Falls die Variable an ein Paket gebunden ist, wird eine Referenz auf das der Variablen zugrundeliegende Objekt zurückgegeben. Ist die Variable nicht gebunden, ist der Rückgabewert `undef`.

**time** liefert die Anzahl an Sekunden zurück, die seit Beginn des systemspezifischen Referenzdatums verstrichen sind. Dieser Zeitpunkt ist bei den meisten Systemen der 1. Januar, 1970 0 Uhr UTC (auf dem Macintosh 1. Januar 1904, 0 Uhr).

**times** liefert ein Array mit vier Elementen zurück, das den Benutzer und die Systemzeiten für den aktuellen Prozess und dessen Kinder enthält.

**truncate** *DateiHandle*, *Länge* verkürzt die angegebene Datei auf die durch *Länge* definierte Länge.

**uc** *Ausdruck* konvertiert alle Buchstaben eines Strings in Großbuchstaben.

**ucfirst** *Ausdruck* liefert *Ausdruck* zurück, nachdem das erste Zeichen in einen Grossbuchstaben konvertiert wurde.

**umask** *Ausdruck* setzt die Standard-umask für den Prozess. Die Funktion übernimmt eine Oktalzahl (keinen String von Ziffern) entsprechend dem `umask`-Kommando von Unix. Ohne Parameter liefert die Funktion die aktuelle `umask`-Maske zurück.

---

**undef Ausdruck** löscht den Wert einer Variablen.

**unlink (Liste)** löscht die Dateien, deren Namen als Liste übergeben wurden. Rückgabewert ist die Anzahl der Dateien, die erfolgreich gelöscht wurden.

**unpack Template, Ausdruck** ist das Gegenstück zu `pack()`. Die Funktion übernimmt eine Datenstruktur und übersetzt sie in eine Liste, die auf einer Template basiert. Das Template-Format ist das gleiche wie für `pack()`.

**unshift Array, Liste** fügt einen skalaren Wert als erstes Element in ein Array ein.

**utime Atime, Mtime, Liste** ist das Perl-Äquivalent zum Unix-Befehl `touch`. Die Funktion setzt die Zugriffs- und Änderungszeiten für eine Liste von Dateien. Die ersten beiden Parameter enthalten die numerischen Zugriffs- und Änderungszeiten für die Dateien. Die Funktion liefert die Zahl der Dateien zurück, die erfolgreich bearbeitet wurden.

**values Hash** liefert ein Array zurück, das die Werte der Elemente eines Hashes enthält.

**vec Ausdruck, Offset, BITS** behandelt einen in `Ausdruck` spezifizierten String als einen Vektor von vorzeichenlosen Integerwerten und liefert den Wert des von `Offset` spezifizierten Bitfeldes zurück.

**wait** wartet auf das Ende eines Kindprozesses und liefert dessen PID zurück.

**waitpid PID, Flags** wartet darauf, dass ein durch `PID` spezifizierter Kindprozess beendet wird. Sie liefert die Prozess-ID des beendeten Prozesses zurück. `Flags` legen den Arbeitsmodus fest.

**wantarray** liefert **wahr** zurück, wenn der Kontext der gerade ausgeführten Subroutine einen Listenwert benötigt. Wird die Funktion in einem skalaren oder leeren Kontext aufgerufen, liefert sie **falsch** zurück.

**warn Liste** schickt eine Nachricht an die Standardfehlerausgabe, ohne wie `die()` das Programm zu beenden. Im Übrigen entspricht diese Funktion der Funktion `die()`.

**write Dateihandle** gibt Daten mithilfe eines Templates aus, das mit der `format()`-Funktion definiert wurde.



# 2

## Perl-Standardmodule

Die Standardmodule werden zusammen mit dem Perl-Interpreter ausgeliefert und haben daher einen sehr hohen Verbreitungsgrad. Einige ältere Module haben nicht die heute übliche Dateierdung „.pm“, sondern noch „.pl“. Bei einer typischen Perl-Installation werden die Standardmodule innerhalb des Installationsverzeichnis im Verzeichnis *lib* abgelegt.

Die folgenden Tabellen erheben keinen Anspruch auf Vollständigkeit, sondern listet nur die wichtigsten Module auf. Eine umfassendere Übersicht der Standardmodule ihrer Distribution erhalten Sie mittels `perldoc perlmodlib`.

### 2.1 Allgemeines

<code>Attribute::Handlers</code>	Handler für Parameter von Funktionen
<code>Benchmark</code>	Messen von Ausführungszeit für Perl-Codes
<code>Carp</code>	Funktionen zur Fehlerbehandlung
<code>CPAN</code>	Laden/Installieren von Modulen aus dem CPAN
<code>Config</code>	Zugriff auf die Konfigurationsdaten von Perl
<code>Data::Dumper</code>	Darstellen der internen Struktur von Variablen
<code>DB</code>	Ermöglicht den Zugriff auf die Debug-API von Perl
<code>Devel::DProf</code>	Profiling
<code>Digest</code>	Routinen der <code>Digest</code> -Module
<code>Digest::MD5</code>	MD5-Prüfsummen berechnen
<code>Dumpvalue</code>	Perl-Daten darstellen
<code>English</code>	Klartextnamen der vordefinierten Variablen
<code>Env</code>	Importiert Umgebungsvariablen mit einzelnen Namen
<code>Filter::Simple</code>	Einfache Variante von <code>Filter::Util::Call</code>
<code>Getopt::Long</code>	Interpretation von Kommandozeilenoptionen
<code>Getopt::Std</code>	Interpretation von Kommandozeilenoptionen
<code>Hash::Util</code>	Funktionen zum Arbeiten mit Hashes
<code>List::Util</code>	Funktionen zum Arbeiten mit Listen
<code>Scalar::Util</code>	Funktionen zum Arbeiten mit Skalaren
<code>Memoize</code>	Funktionen beschleunigen
<code>POSIX</code>	POSIX-Bibliothek von Funktionen (IEEE 1003.1)
<code>Shell</code>	Standardausgabe von Shell-Kommandos abfangen
<code>Storable</code>	Datenstrukturen auf der Platte speichern

Switch	switch-Anweisungen in Perl
Sys::Syslog	Aufrufe des Unix-Programms <code>syslog</code>
Unicode::Collate	Klasse zum Arbeiten mit Unicode-Strings
Unicode::Normalize	Funktionen zum Normalisieren von Unicode
Unicode::UCD	Schnittstelle zur Unicode-Zeichendatenbank

## 2.2 Bearbeitung von Text

MIME::Base64	Umwandeln von Zeichenketten in das Base64-Format
MIME::QuotedPrint	Umwandeln von Zeichenketten in das Quoted-Printable-Format
Search::Dict	Wörterbuch nach einem Wort durchsuchen
Term::ANSIColor	Text-Bildschirm Ausgaben einfärben
Term::Cap	Zugriff auf <code>termcap</code> (Unix)
Term::Complete	Automatische Vervollständigung von Wordeingaben
Text::ParseWords	Zerlegt einen Text in die einzelnen Wörter
Text::Soundex	Erzeugt für ein Wort den Soundex-Code
Text::Tabs	Ersetzt Tabulatoren durch Leerzeichen
Text::Wrap	Bricht Text automatisch um

## 2.3 Berechnungen

Math::BigFloat	Rechnen mit großen Fließkommazahlen
Math::BigInt	Rechnen mit großen Ganzzahlen
Math::BigRat	Rechnen mit großen rationalen Zahlen
Math::Complex	Rechnen mit komplexen Zahlen
Math::Trig	Trigonometrische Funktionen und Pi

## 2.4 Dateiverwaltung

Cwd	Ermittelt das aktuelle Arbeitsverzeichnis
DirHandle	Objektorientierte Directory-Funktionen
File::Basename	Zerlegt einen Pfad in Pfadnamen/Dateinamen
File::CheckTree	Prüft mehrere Dateien/Verzeichnisse auf Existenz
File::Compare	Vergleicht zwei Dateien
File::Copy	Kopieren von Dateien
File::Find	Durchsucht Verzeichnisbäume
File::Path	Bearbeiten mehrerer Verzeichnisse
File::Spec	Arbeiten mit syntaktisch korrekten Dateinamen
File::stat	Objektorientierte Variante von <code>stat()</code>
File::Temp	Arbeiten mit temporären Dateien (eindeutige Namen)
FileCache	Viele Dateien gleichzeitig öffnen
FileHandle	Objektorientierte Handles für Dateien, Positionierung
IO	lädt die IO-Module
IO::Dir	Objektorientierte Verzeichnishandles
IO::File	Objektorientierten Ein-/Ausgabe-Handles
IO::Pipe	Objektorientierte Pipes zwischen Prozessen

---

IO::Poll	Objektorientierter Zugriff auf <code>poll()</code>
IO::Seekable	Objektorientierte Front-Ends für <code>seek()</code> , <code>tell()</code> und andere Funktionen
IO::Select	Objektorientierte Verwaltung von Ein- und Ausgabe-Handles

## 2.5 Datum und Uhrzeit

Time::HiRes	Millisekunden bei <code>time()</code> und <code>sleep()</code>
Time::Local	Umkehrfunktionen für <code>gmtime()</code> und <code>localtime()</code>
Time::gmtime	Objektorientierte Variante von <code>gmtime()</code>
Time::localtime	Objektorientierte Variante von <code>localtime()</code>

## 2.6 Prozesskommunikation

IPC::Msg	Arbeiten mit System V Message Queues
IPC::Open2	Öffnet einen Unix-Prozess zum Lesen und Schreiben
IPC::Open3	Öffnet einen Unix-Prozess zum Lesen, Schreiben und für die Fehlerausgabe
IPC::Semaphore	System-V-Semaphor-Funktionen
IPC::SysV	IPC-Konstanten von Unix-System V
Thread	Arbeiten mit Multithreading

## 2.7 Variablenbindung

Tie::Array	Bindet eine Liste an eine Objektklasse
Tie::File	Zeilen einer Datei als Array ansprechen
Tie::Handle	Bindet ein Handle an eine Objektklasse
Tie::Hash	Bindet einen Hash an eine Objektklasse
Tie::Memoize	Fügt Daten einem Hash hinzu
Tie::RefHash	Referenzen als Schlüssel für Hashes
Tie::Scalar	Bindet einen Skalar an eine Objektklasse

## 2.8 Modulverwaltung und objektorientierte Programmierung

Class::ISA	Stellt die Werte der @ISA-Listen bereit
Exporter	Gezieltes Exportieren von Variablen und Funktionen
UNIVERSAL	Basis-Objektklasse für objektorientiertes Programmieren

## 2.9 Systemnahe Module

Errno	Importiert die Konstanten aus <code>errno.h</code>
Fcntl	Übersetzung der C-Header-Datei <code>fcntl.h</code>

Test	Stellt Test-Typen und Fehlerreaktionen bereit
Test::Builder	Eigene Bibliotheken für Tests erstellen
Test::Harness	Standardisierte Tests
Test::More	Erweiterung von Test::Simple
Test::Simple	Einfaches Testsystem
Win32	Zugriff auf einige interne Windows-Funktionen

## 2.10 Netzwerk und CGI

Socket	Verwaltung von Sockets (prozedural)
IO::Socket	Objektorientierte Verwaltung von Sockets
Net::Cmd	Basisklasse für befehlsbasierte Protokolle
Net::Domain	Nameserver-Abfrage
Net::FTP	Kommunikation mit FTP-Servern
Net::Netrc	Objektorientierter Zugriff auf die <code>.netrc</code> -Datei
Net::NNTP	Kommunikation mit News-Servern
Net::Ping	Ping-Funktion
Net::POP3	Kommunikation mit POP3-Servern
Net::SMTP	Kommunikation mit SMTP-Servern
Net::Time	Zugriff auf Zeit-Server
Net::hostent	Ersetzt <code>gethostbyname</code> durch ein Objekt
Net::netent	Ersetzt <code>getnetbyname</code> durch ein Objekt
Net::protoent	Ersetzt <code>getprotobyname</code> durch ein Objekt
Net::servent	Ersetzt <code>getservbyname</code> durch ein Objekt
User::grent	Ersetzt <code>getgrnam</code> durch ein Objekt
User::pwent	Ersetzt <code>getpwnam</code> durch ein Objekt
CGI	Formulardaten verarbeiten und Erzeugen von HTML-Ausgaben
CGI::Carp	Leitet Fehlermeldungen auf den Browser
CGI::Cookie	Verarbeiten von Cookies
CGI::Push	Schnittstelle für Server-Push

## 2.11 Verwalten von Perl-Erweiterungen

AutoLoader	Lädt Unterprogramme nur, wenn erforderlich
AutoSplit	Splittet ein Programm in mehrere Einzeldateien auf
DynaLoader	Lädt dynamisch Routinen aus den C-Bibliotheken
ExtUtils	Viele Untermodule für die Programmentwicklung in C
XS::APItest	Prüft, ob die C-API von Perl korrekt funktioniert
XS::Typemap	Prüft die XS-Typemaps von Perl
XSLoader	Lädt dynamisch Routinen aus den C-Bibliotheken

## 2.12 Dokumentation mit POD

Pod::Checker	Prüft die Syntax von POD-Dokumenten
Pod::Find	Verzeichnisbaum nach POD-Dokumenten durchsuchen
Pod::Functions	Liste von Perl-Standardfunktionen

<code>Pod::Html</code>	Generiert aus dem POD-Format HTML-Code
<code>Pod::LaTeX</code>	Konvertiert POD-Dokumente ins LaTeX-Format
<code>Pod::Man</code>	Konvertiert POD-Dokumente ins Manual-Format
<code>Pod::PlainText</code>	Konvertiert POD-Dokumente in formatierten Text
<code>Pod::Text</code>	Konvertiert POD-Dokumente in reinen Text
<code>Pod::Parser</code>	Basisklasse zum Parsen des POD-Formats
<code>Pod::ParseLink</code>	Parst Links in einem POD-Dokument
<code>Pod::ParseUtils</code>	Enthält verschiedene Klassen zum Parsen von POD-Dokumenten
<code>Pod::Select</code>	Extrahiert eine bestimmte Stelle aus einem POD-Dokument
<code>Pod::Usage</code>	Erzeugt Usage-Info aus POD-Dokumenten