Perl Reference Guide

for Perl version 4.035

Perl program designed and created by
Larry Wall **<**lwall@netlabs.com**>**

Reference guide designed and created by
Johan Vromans **<**jv@mh.nl**>**

Contents

1. Command line options
2. Literals
3. Variables
4. Statements
5. Flow control
6. Operators
7. File test operators
8. Arithmetic functions
9. Conversion functions
10. Structure conversion
11. String functions
12. Array and list functions
13. File operations
14. Directory reading routines
15. Input / Output
16. Search and replace functions
17. System interaction
18. Networking
19. SystemV IPC
20. Miscellaneous
21. Formats
22. Info from system files
23. Regular expressions
24. Special variables
25. Special arrays
26. The perl debugger
27. Environment variables

Rev. 4.035.1

Conventions

fixed	denotes literal text.
THIS	means variable text, i.e. things you must fill in.
THIS†	means that THIS will default to **$_** if omitted.
word	is a keyword, i.e. a word with a special meaning.
RET	denotes pressing a keyboard key.
[. . .]	denotes an optional part.
(. . .)*	means that the parentheses may be omitted.

1. Command line options

-a turns on autosplit mode when used with **-n** or **-p**. Splits to **@F**.

-c checks syntax but does not execute.

-d runs the script under the debugger. Use **-de 0** to start the debugger without a script.

-D NUMBER
sets debugging flags.

-e COMMANDLINE
may be used to enter one line of script. Multiple **-e** commands may be given to build up a multi-line script.

-i EXT
files processed by the **<>** construct are to be edited in-place.

-I DIR
with **-P**: tells the C preprocessor where to look for include files. The directory is prepended to **@INC**.

-l [OCTNUM]
enables automatic line ending processing, e.g. **-l013**.

-n assumes an input loop around your script. Lines are not printed.

-p assumes an input loop around your script. Lines are printed.

-P runs the C preprocessor on the script before compilation by perl.

-s interprets "**-xxx**" on the command line as switches and sets the corresponding variables **$xxx** in the script.

-S uses the **PATH** environment variable to search for the script.

-u dumps core after compiling the script. To be used with the *undump* program (where available).

-U allows perl to do unsafe operations.

-v prints the version and patchlevel of your perl executable.

-w prints warnings about possible spelling errors and other error-prone constructs in the script.

-x extracts perl program from input stream.

-0 VAL
(that's the number zero) designates an initial value for the record terminator **$/**. See also **-l**.

2. Literals

Numeric: **123 1_234 123.4 5E-10 0xff** (hex) **0377** (octal).

String: **'abc'** literal string, no variable interpolation nor escape characters.
Also: **q/abc/**.
(Almost any pair of delimiters can be used instead of **/.../**.)

"abc" Variables are interpolated and escape sequences are processed.
Also: **qq/abc/**.
Escape sequences: **\t** (Tab), **\n** (Newline), **\r** (Return), **\f** (Formfeed), **\b** (Backspace), **\a** (Alarm), **\e** (Escape), **\033**(octal), **\x1b**(hex), **\c[** (control).
\l and **\u** lowcase/upcase the following character;
\L and **\U** lowcase/upcase until a **\E** is encountered.

`COMMAND` evaluates to the output of the COMMAND.
Also: **qx/**COMMAND**/**.

Array: **(1,2,3)**. **()** is an empty array.
Also: **($a,$b,@rest) = (1,2,...);**
(1..4) is the same as **(1,2,3,4)**. Likewise **('abc'..'ade')**

Associative array: **(**KEY1**,**VAL1**,**KEY2**,**VAL2**,...)**

Filehandles:
Pre-defined: **<STDIN>**, **<STDOUT>**, **<STDERR>**, **<ARGV>**, **<DATA>**;
User-specified: **<**HANDLE**>**, **<$**VAR**>**.
<> is the input stream formed by the files specified in **@ARGV**, or standard input if no arguments are supplied.

Globs: **<**PATTERN**>** evaluates to all filenames according to the pattern.
Use **<${**VAR**}>** to glob from a variable.

Here-Is: **<<**IDENTIFIER
See the manual for details.

Special tokens:
__FILE__: filename; **__LINE__**: line number.
__END__: end of program; remaining lines can be read using **<**DATA**>**.

3. Variables

$var	a simple scalar variable
$var[28]	29th element of array **@var** (the **[]** are part of it)
$var{'Feb'}	one value from associative array **%var**
$#var	last index of array **@var**
@var	the entire array; in scalar context: the number of elements in the array
@var[3,4,5]	a slice of the array **@var**
@var{'a','b'}	a slice of **%var**; same as **($var{'a'},$var{'b'})**
%var	the entire associative array; in scalar context: TRUE if the array has elements
$var{'a',1,...}	emulates a multi-dimensional array
('a'..'z')[4,7,9]	a slice of an array literal
***NAME**	refers to all objects represented by NAME. "***name1 = *name2**" makes **name1** a reference to **name2**.

4. Statements

Every statement is an expression, optionally followed by a modifier, and terminated by a semicolon. The semicolon may be omitted if the statement is the final one in a BLOCK.

Execution of expressions can depend on other expressions using one of the modifiers **if**, **unless**, **while** or **until**, e.g.:

EXPR1 **if** EXPR2 **;**
EXPR1 **until** EXPR2 **;**

Also, by using one of the logical operators **||**, **&&** or **? :**, e.g.:

EXPR1 **||** EXPR2 **;**
EXPR1 **?** EXPR2 **:** EXPR3 **;**

Statements can be combined to form a BLOCK when enclosed in **{}**.

Compound statements may be used to control flow:

if (EXPR**)** BLOCK [[**elsif (**EXPR**)** BLOCK ...] **else** BLOCK]
unless (EXPR**)** BLOCK [**else** BLOCK]
[LABEL**:**] **while (**EXPR**)** BLOCK [**continue** BLOCK]
[LABEL**:**] **until (**EXPR**)** BLOCK [**continue** BLOCK]
[LABEL**:**] **for (**EXPR**;** EXPR**;** EXPR**)** BLOCK
[LABEL**:**] **foreach** VAR† **(**ARRAY**)** BLOCK
[LABEL**:**] BLOCK [**continue** BLOCK]

Special forms are:

do BLOCK **while** EXPR **;**
do BLOCK **until** EXPR **;**

which are guaranteed to perform BLOCK once before testing EXPR.

5. Flow control

do BLOCK

Returns the value of the last command in the sequence of commands indicated by BLOCK. **next**, **last** and **redo** cannot be used here.

do SUBROUTINE**(LIST)**

Executes a SUBROUTINE declared by a **sub** declaration, and returns the value of the last expression evaluated in SUBROUTINE .
Preferred form is: **&**SUBROUTINE .

do FILENAME

Executes the contents of FILENAME as a perl script. Errors are returned in **$@**.
Preferred form is: **require** FILENAME .

goto LABEL

Continue execution at the specified label.

last [LABEL]

Immediately exits the loop in question. Skips continue block.

next [LABEL]

Starts the next iteration of the loop.

redo [LABEL]

Restarts the loop block without evaluating the conditional again.

return EXPR

Returns from a subroutine with the value specified.

6. Operators

+ – * /	Addition, subtraction, multiplication, division.
%	Modulo division.
| & ^	Bitwise or, bitwise and, bitwise exclusive or.
>> <<	Bitwise shift right, bitwise shift left.
******	Exponentiation.
.	Concatenation of two strings.
x	Returns a string or array consisting of the left operand (an array or a string) repeated the number of times specified by the right operand.

All of the above operators also have an assignment operator, e.g. "**.=**".

++ --	Auto-increment (magical on strings), auto-decrement.
? :	Alternation (if-then-else) operator.
|| &&	Logical or, logical and.
== !=	Numeric equality, inequality.
eq ne	String equality, inequality.
< >	Numeric less than, greater than.
lt gt	String less than, greater than.
<= >=	Numeric less (greater) than or equal to.
le ge	String less (greater) than or equal.
<=>	Numeric compare. Returns -1, 0 or 1.
cmp	String compare. Returns -1, 0 or 1.
=~ !~	Search pattern, substitution, or translation (negated).
..	Enumeration, also input line range operator.
,	Comma operator.

7. File test operators

These unary operators takes one argument, either a filename or a filehandle, and tests the associated file to see if something is true about it. If the argument is omitted, tests **$_** (except for **-t,** which tests **STDIN**). If the special argument **_** (underscore) is passed, uses the info of the preceding test.

-r -w -x	File is readable/writable/executable by effective uid/gid.
-R -W -X	File is readable/writable/executable by real uid/gid.
-o -O	File is owned by effective/real uid.
-e -z	File exists / has zero size.
-s	File exists and has non-zero size. Returns the size.
-f -d	File is a plain file, a directory.
-l -S -p	File is a symbolic link, a socket, a named pipe (FIFO).
-b -c	File is a block/character special file.
-u -g -k	File has setuid/setgid/sticky bit set.
-t	Tests if filehandle (**STDIN** by default) is opened to a tty.
-T -B	File is a text/non-text (binary) file. **-T** and **-B** return TRUE on a null file, or a file at EOF when testing a filehandle.
-M -A -C	File creation / access / inode change time. Measured in days since this program started. See also **$^T** in section "Special Variables".

A LIST is a (possibly parenthesised) list of expressions, variables or LISTs. An array variable or an array slice may always be used instead of a LIST.

8. Arithmetic functions

atan2(Y,X**)**
: Returns the arctangent of Y/X in the range -π to π.

cos(EXPR†**)***
: Returns the cosine of EXPR (expressed in radians).

exp(EXPR†**)***
: Returns **e** to the power of EXPR.

int(EXPR†**)***
: Returns the integer portion of EXPR.

log(EXPR†**)***
: Returns natural logarithm (base **e**) of EXPR.

rand[**(**EXPR**)***]
: Returns a random fractional number between 0 and the value of EXPR. If EXPR is omitted, returns a value between 0 and 1.

sin(EXPR†**)***
: Returns the sine of EXPR (expressed in radians).

sqrt(EXPR†**)***
: Return the square root of EXPR.

srand[**(**EXPR**)***]
: Sets the random number seed for the rand operator.

time Returns the number of seconds since January 1, 1970. Suitable for feeding to **gmtime** and **localtime**.

9. Conversion functions

gmtime(EXPR**)***
: Converts a time as returned by the **time** function to a 9-element array (**$**sec, **$**min, **$**hour, **$**mday, **$**mon, **$**year, **$**wday, **$**yday, **$**isdst) with the time analyzed for the Greenwich timezone. **$**mon has the range 0..11 and **$**wday has the range 0..6.

hex(EXPR†**)***
: Returns the decimal value of EXPR interpreted as an hex string.

localtime(EXPR**)***
: Converts a time as returned by the **time** function to a 9-element array with the time analyzed for the local timezone.

oct(EXPR†**)***
: Returns the decimal value of EXPR interpreted as an octal string. If EXPR starts off with **0x**, interprets it as a hex string instead.

ord(EXPR†**)***
: Returns the ascii value of the first character of EXPR.

vec(EXPR,OFFSET,BITS**)**
: Treats EXPR as a string of unsigned ints, and yields the bit at OFFSET. BITS must be between 1 and 32. May be used as an lvalue.

10. Structure conversion

pack(TEMPLATE,LIST**)**

Packs the values into a binary structure using TEMPLATE.

unpack(TEMPLATE,EXPR**)**

Unpacks the structure EXPR into an array, using TEMPLATE.

TEMPLATE is a sequence of characters as follows:

a / **A**	Ascii string, null / space padded
b / **B**	Bit string in ascending / descending order
c / **C**	Native / unsigned char value
f / **d**	Single / double float in native format
h / **H**	Hex string, low / high nybble first.
i / **I**	Signed / unsigned integer value
l / **L**	Signed / unsigned long value
n / **N**	Short / long in network (big endian) byte order
s / **S**	Signed / unsigned short value
u / **p**	Uuencoded string / Pointer to a string
v / **V**	Short / long in VAX (little endian) byte order
x / **@**	Null byte / null fill until position
X	Backup a byte

Each character may be followed by a decimal number which will be used as a repeat count, an ***** specifies all remaining arguments.
If the format is preceded with **%**N, **unpack** returns an N-bit checksum instead.
Spaces may be included in the template for readability purposes.

11. String functions

chop(LIST†**)**

Chops off the last character on all elements of the list; returns the last chopped character. The parentheses may be omitted if LIST is a single variable.

crypt(PLAINTEXT,SALT**)**

Encrypts a string.

eval(EXPR†**)***

EXPR is parsed and executed as if it were a perl program. The value returned is the value of the last expression evaluated. If there is a syntax error or runtime error, an undefined string is returned by **eval**, and **$@** is set to the error message.

eval{EXPR**;...}**

Executes the code between **{** and **}**. Traps runtime errors as described above.

index(STR,SUBSTR[,OFFSET]**)**

Returns the position of SUBSTR in STR at or after OFFSET. If the substring is not found, returns **$[-1**.

length(EXPR†**)***

Returns the length in characters of the value of EXPR.

rindex(STR,SUBSTR[,OFFSET]**)**

Returns the position of the last SUBSTR in STR at or before OFFSET.

substr(EXPR,OFFSET[,LEN]**)**

Extracts a substring out of EXPR and returns it. If OFFSET is negative, counts from the end of the string. May be used as an lvalue.

12. Array and list functions

delete **$**ARRAY**{**KEY**}**

Deletes the specified value from the specified associative array. Returns the deleted value.

each(%ARRAY**)***

Returns a 2-element array consisting of the key and value for the next value of an associative array. Entries are returned in an apparently random order. When the array is entirely read, a null array is returned. The next call to **each** after that will start iterating again.

grep(EXPR,LIST**)**

Evaluates EXPR for each element of the LIST, locally setting **$_** to refer to the element. Modifying **$_** will modify the corresponding element from LIST. Returns array of elements from LIST for which EXPR returned true.

join(EXPR,LIST**)**

Joins the separate strings of LIST into a single string with fields separated by the value of EXPR, and returns the string.

keys(%ARRAY**)***

Returns an array with of all the keys of the named associative array.

pop(@ARRAY**)***

Pops and returns the last value of the array, shortens the array by 1.

push(@ARRAY,LIST**)**

Pushes the values of LIST onto the end of ARRAY. The length of the array increases by the length of LIST.

reverse(LIST**)***

In array context: returns the LIST in reverse order.
In scalar context: returns the first element of LIST with bytes reversed.

scalar(@ARRAY**)**

Returns the number of elements in the array.

scalar(%ARRAY**)**

Returns TRUE if the associative array has elements defined.

shift[(@ARRAY)*]

Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down. If @ARRAY is omitted, shifts **@ARGV** in main and **@_** in subroutines.

sort([SUBROUTINE] LIST**)***

Sorts the LIST and returns the sorted array value. If SUBROUTINE is specified, gives the name of a subroutine that returns less than zero, zero, or greater than zero, depending on how the elements of the array, available to the routine as **$a** and **$b**, are to be ordered.
SUBROUTINE may be the name of a user-defined routine, or a BLOCK (see "Statements" and "Miscellaneous").

splice(@ARRAY,OFFSET[,LENGTH[,LIST]]**)**

Removes the elements of @ARRAY designated by OFFSET and LENGTH, and replaces them with LIST (if specified).
Returns the elements removed.

split[(PATTERN[,EXPR†[,LIMIT]])]

Splits a string into an array of strings, and returns it. If LIMIT is specified, splits in no more than that many fields. If PATTERN is also omitted, splits on whitespace. If not in array context: returns number of fields and splits to **@_**. See also: "Search and Replace Functions".

unshift(@ARRAY**,**LIST**)**

Prepends list to the front of the array, and returns the number of elements in the new array.

values(%ARRAY**)***

Returns a normal array consisting of all the values of the named associative array.

13. File operations

Functions operating on a list of files return the number of files successfully operated upon.

chmod(LIST**)***

Changes the permissions of a list of files. The first element of the list must be the numerical mode.

chown(LIST**)***

Changes the owner and group of a list of files. The first two elements of the list must be the numerical uid and gid.

truncate(FILE**,**SIZE**)**

truncates FILE to SIZE. FILE may be a filename or a filehandle.

link(OLDFILE**,**NEWFILE**)**

Creates a new filename linked to the old filename.

lstat(FILE**)**

Like stat, but does not traverse a final symbolic link.

mkdir(DIR**,**MODE**)**

Creates a directory with given permissions. Sets **$!** on failure.

readlink(EXPR†**)***

Returns the value of a symbolic link.

rename(OLDNAME**,**NEWNAME**)**

Changes the name of a file.

rmdir(FILENAME†**)***

Deletes the directory if it is empty. Sets **$!** on failure.

stat(FILE**)**

Returns a 13-element array (**$**dev, **$**ino, **$**mode, **$**nlink, **$**uid, **$**gid, **$**rdev, **$**size, **$**atime, **$**mtime, **$**ctime, **$**blksize, **$**blocks). FILE can be a filehandle, an expression evaluating to a filename, or **_** to refer to the last file test operation.

Returns a null list if the **stat** fails.

symlink(OLDFILE**,**NEWFILE**)**

Creates a new filename symbolically linked to the old filename.

unlink(LIST**)***

Deletes a list of files.

utime(LIST**)***

Changes the access and modification times. The first two elements of the list must be the numerical access and modification times.

14. Directory reading routines

closedir(DIRHANDLE**)***
: Closes a directory opened by opendir.

opendir(DIRHANDLE**,**DIRNAME**)**
: Opens a directory on the handle specified.

readdir(DIRHANDLE**)***
: Returns the next entry (or an array of entries) in the directory.

rewinddir(DIRHANDLE**)***
: Positions the directory to the beginning.

seekdir(DIRHANDLE**,**POS**)**
: Sets position for readdir on the directory.

telldir(DIRHANDLE**)***
: Returns the postion in the directory.

15. Input / Output

In input/output operations, FILEHANDLE may be a filehandle as opened by the **open** operator, or a scalar variable which evaluates to the name of a filehandle to be used.

binmode(FILEHANDLE**)***
: Arranges for the file opened on FILEHANDLE to be read in "binary" mode as opposed to "text" mode (MS-DOS only).

close(FILEHANDLE**)***
: Closes the file or pipe associated with the file handle.

dbmclose(%ARRAY**)***
: Breaks the binding between the array and the dbm file.

dbmopen(%ARRAY**,**DBMNAME**,** MODE**)**
: Binds a dbm or ndbm file to the associative array. If the database does not exist, it is created with the indicated mode.

eof(FILEHANDLE**)**
: Returns 1 if the next read will return end of file, or if the file is not open.

eof
: Returns the eof status for the last file read.

eof( )
: Indicates eof on the pseudo file formed of the files listed on the command line.

fcntl(FILEHANDLE**,**FUNCTION**,$**VAR**)**
: Implements the *fcntl*(2) function. This function has non-standard return values. See the manual for details.

fileno(FILEHANDLE**)***
: Returns the file descriptor for a given (open) file.

flock(FILEHANDLE**,**OPERATION**)**
: Calls *flock*(2) on the file. OPERATION adds from 1 (shared), 2 (exclusive), 4 (non-blocking) or 8 (unlock).

getc[**(**FILEHANDLE**)***]
: Yields the next character from the file, or **""** on EOF. If FILEHANDLE is omitted, reads from **STDIN**.

ioctl(FILEHANDLE**,**FUNCTION**,$**VAR**)**
: performs *ioctl*(2) on the file. This function has non-standard return values. See the manual for details.

open(FILEHANDLE[**,**FILENAME]**)**

Opens a file and associates it with FILEHANDLE. If FILENAME is omitted, the scalar variable of the same name as the FILEHANDLE must contain the filename.

The following filename conventions apply when opening a file.

"FILE**"**	open FILE for input. Also **"<**FILE**"**.
">FILE**"**	open FILE for output, creating it if necessary.
">>FILE**"**	open FILE in append mode.
"+>FILE**"**	open FILE with read/write access.
"|CMD**"**	opens a pipe to command CMD.
"CMD**|"**	opens a pipe from command CMD.

FILE may be **&**FILEHND in which case the new file handle is connected to the (previously opened) filehandle FILEHND.

open returns 1 upon success, **undef** otherwise, except for pipes. The parentheses may be omitted, if only a FILEHANDLE is specified.

pipe(READHANDLE**,**WRITEHANDLE**)**

Returns a pair of connected pipes.

print[**(**[FILEHANDLE]LIST†**)***]

Prints a string or a comma-separated list of strings. If FILEHANDLE is omitted, prints by default to standard output (or to the last selected output channel - see **select**).

printf[**(**[FILEHANDLE] LIST**)***]

Equivalent to **print** FILEHANDLE **sprintf(**LIST**)**.

read(FILEHANDLE**,$**VAR**,**LENGTH[**,**OFFSET]**)**

Read LENGTH binary bytes from the file into the variable at OFFSET. Returns number of bytes actually read.

seek(FILEHANDLE**,**POSITION**,**WHENCE**)**

Arbitrarily positions the file. Returns 1 upon success, 0 otherwise.

select[**(**FILEHANDLE**)**]

Returns the currently selected filehandle. Sets the current default filehandle for output operations if FILEHANDLE is supplied.

select(RBITS**,**WBITS**,**NBITS**,**TIMEOUT**)**

Performs a *select*(2) system call with the same parameters.

sprintf(FORMAT**,**LIST**)**

Returns a string formatted by (almost all of) the usual printf conventions.

sysread(FILEHANDLE**,$**VAR**,**LENGTH[**,**OFFSET]**)**

Reads LENGTH bytes into **$**VAR at OFFSET.

syswrite(FILEHANDLE**,**SCALAR**,**LENGTH[**,**OFFSET]**)**

Writes LENGTH bytes from SCALAR at OFFSET.

tell[**(**FILEHANDLE**)**]*

Returns the current file position for the file. If FILEHANDLE is omitted, assumes the file last read.

write[**(**FILEHANDLE**)**]*

Writes a formatted record to the specified file, using the format associated with that file. See “Formats”.

16. Search and replace functions

[EXPR **=~**] [**m**]**/**PATTERN**/**[**g**][**i**][**o**]

Searches EXPR (default: **$_**) for a pattern. If you prepend an **m** you can use almost any pair of characters as delimiters. If used in array context, an array is returned consisting of the subexpressions matched by the parentheses in pattern, i.e. **($1,$2,$3,...)**.

Optional modifiers: **g** matches as many times as possible; **i** searches in a case-insensitive manner; **o** interpolates variables only once.

If PATTERN is empty, the most recent pattern from a previous match or replacement is used.

With **g** the match can be used as an iterator in scalar context.

?PATTERN**?**

This is just like the **/**PATTERN**/** search, except that it matches only once between calls to the reset operator. If PATTERN is empty, the most recent pattern from a previous match or replacement is used.

[**$**VAR **=~**] **s/**PATTERN**/**REPLACEMENT**/**[**g**][**i**][**e**][**o**]

Searches a string for a pattern, and if found, replaces that pattern with the replacement text and returns the number of substitutions made. Otherwise it returns false.

Optional modifiers: **g** replaces all occurrences of the pattern; **e** interprets the replacement string as an expression; **i** and **o** as with **/**PATTERN**/** matching. Almost any delimiter may replace the slashes; if single quotes are used, no interpretation is done on the replacement string.

If bracketing quotes are used, PATTERN and REPLACEMENT may have their own delimiters, e.g. **s(foo)[bar]**.

If PATTERN is empty, the most recent pattern from a previous match or replacement is used.

study[**(**$VAR†**)***]

Study the contents of **$**VAR in anticipation of doing many pattern matches on the contents before it is next modified.

[**$**VAR **=~**] **tr/**SEARCHLIST**/**REPLACEMENTLIST**/**[**c**][**d**][**s**]

Translates all occurrences of the characters found in the search list with the corresponding character in the replacement list. It returns the number of characters replaced. **y** may be used instead of **tr**.

Optional modifiers: **c** complements the SEARCHLIST; **d** deletes all characters not found in SEARCHLIST; **s** squeezes all sequences of characters that are translated into the same target character into one occurrence of this character.

17. System interaction

alarm(EXPR**)***

Schedules a **SIGALRM** to be delivered after EXPR seconds.

chdir [**(**EXPR**)***]

Changes the working directory, **$ENV{"HOME"}** if EXPR is omitted.

chroot(FILENAME†**)***

Changes the root directory for the process and its children.

die[(LIST)*]
: Prints the value of LIST to **STDERR** and exits with the current value of **$!** (errno). If **$!** is 0, exits with the value of **($? >> 8)**. If **($? >> 8)** is 0, exits with 255. LIST defaults to **"Died."**.

exec(LIST**)***
: Executes the system command in LIST; does not return.

exit(EXPR**)***
: Exits immediately with the value of **EXPR**.

fork
: Does a *fork*(2) system call. Returns the child pid to the parent process and zero to the child process.

getlogin
: Returns the current login name as known by the system.

getpgrp[(PID)*]
: Returns the process group for process PID (0, or omitted, means the current process).

getppid
: Returns the process id of the parent process.

getpriority(WHICH**,**WHO**)**
: Returns the current priority for a process, process group, or user.

kill(LIST**)***
: Sends a signal to a list of processes. The first element of the list must be the signal to send (numeric, or its name as a string).

setpgrp(PID**,**PGRP**)**
: Sets the process group for the PID (0 = current process).

setpriority(WHICH**,**WHO**,**PRIO**)**
: Sets the current priority for a process, process group, or a user.

sleep[(EXPR)*]
: Causes the script to sleep for EXPR seconds, or forever if no EXPR. Returns the number of seconds actually slept.

syscall(LIST**)***
: Calls the system call specified in the first element of the list, passing the rest of the list as arguments to the call.

system(LIST**)***
: Does exactly the same thing as **exec** LIST except that a fork is done first, and the parent process waits for the child process to complete.

times
: Returns a 4-element array (**$**user, **$**system, **$**cuser, **$**csystem) giving the user and system times, in seconds, for this process and the children of this process.

umask[(EXPR)*]
: Sets the umask for the process and returns the old one. If EXPR is omitted, returns current umask value.

wait
: Waits for a child process to terminate and returns the pid of the deceased process (-1 if none). The status is returned in **$?**.

waitpid(PID**,**FLAGS**)**
: Performs the same function as the corresponding system call.

warn(LIST**)***
: Prints the message on **STDERR** like **die**, but doesn't exit.

18. Networking

accept(NEWSOCKET**,**GENERICSOCKET**)**
Accepts a new socket.

bind(SOCKET**,**NAME**)**
Binds the NAME to the SOCKET.

connect(SOCKET**,**NAME**)**
Connects the NAME to the SOCKET.

getpeername(SOCKET**)**
Returns the socket address of the other end of the SOCKET.

getsockname(SOCKET**)**
Returns the name of the socket.

getsockopt(SOCKET**,**LEVEL**,**OPTNAME**)**
Returns the socket options.

listen(SOCKET**,**QUEUESIZE**)**
Starts listening on the specified SOCKET.

recv(SOCKET**,**SCALAR**,**LENGTH**,**FLAGS**)**
Receives a message on SOCKET.

send(SOCKET**,**MSG**,**FLAGS[**,**TO]**)**
Sends a message on the SOCKET.

setsockopt(SOCKET**,**LEVEL**,**OPTNAME**,**OPTVAL**)**
Sets the requested socket option.

shutdown(SOCKET**,**HOW**)**
Shuts down a SOCKET.

socket(SOCKET**,**DOMAIN**,**TYPE**,**PROTOCOL**)**
Creates a SOCKET in DOMAIN with TYPE and PROTOCOL.

socketpair(SOCKET1**,**SOCKET2**,**DOMAIN**,**TYPE**,**PROTOCOL**)**
As socket, but creates a pair of bi-directional sockets.

19. SystemV IPC

The following functions all perform the same action as the corresponding system calls.

msgctl(ID**,**CMD**,**ARGS**)**
msgget(KEY**,**FLAGS**)**
msgsnd(ID**,**MSG**,**FLAGS**)**
msgrcv(ID**,$**VAR**,**SIZE**,**TYPE**,**FLAGS**)**
semctl(ID**,**SEMNUM**,**CMD**,**ARG**)**
semget(KEY**,**NSEMS**,**SIZE**,**FLAGS**)**
semop(KEY**,...)**
shmctl(ID**,**CMD**,**ARG**)**
shmget(KEY**,**SIZE**,**FLAGS**)**
shmread(ID**,$**VAR**,**POS**,**SIZE**)**
shmwrite(ID**,**STRING**,**POS**,**SIZE**)**

20. Miscellaneous

caller[(EXPR)]
: Returns an array (**$**package,**$**file,**$**line,...) for a specific subroutine call. "**caller**" returns this info for the current subroutine, "**caller(1)**" for the caller of this subroutine etc..

defined(EXPR**)***
: Tests whether the lvalue EXPR has a real value.

dump [LABEL]
: Immediate core dump. When reincarnated, starts at LABEL.

local(LIST**)**
: Creates a scope for the listed variables local to the enclosing block, subroutine or eval.

package NAME
: Designates the remainder of the current block as a package.

require(EXPR†**)***
: Includes the specified file from the perl library. Does not include more than once, and yields a fatal error if the file does not include OK.

reset [(EXPR)*]
: Resets **??** searches so that they work again. EXPR is a list of single letters. All variables and arrays beginning with one of those letters are reset to their pristine state. Only affects the current package.

scalar(EXPR**)**
: Forces evaluation of EXPR in scalar context.

sub NAME **{** EXPR **;** ... **}**
: Designates NAME as a subroutine. Parameters are passed by reference as array **@_**. Returns the value of the last expression evaluated.

undef[**(**LVALUE**)***]
: Undefines the LVALUE. Always returns the undefined value.

wantarray
: Returns true if the current context expects an array value.

21. Formats

format [NAME] **=**
FORMLIST
.

FORMLIST pictures the lines, and contains the arguments which will give values to the fields in the lines. Picture fields are:

@<<<...	left adjusted field, repeat the **<** to denote the desired width;
@>>>...	right adjusted field;
@|||...	centered field;
@#.##...	numeric format with implied decimal point;
@*	a multi-line field.

Use **^** instead of **@** for multi-line block filling.

Use **~** at the beginning of a line to suppress unwanted empty lines.

Use **~~** at the beginning of a line to have this format line repeated until all fields are exhausted.

Use **$–** to zero to force a page break.

See also **$^**, **$~**, **$–** and **$=** in section "Special Variables".

22. Info from system files

See the manual about return values in scalar context.

passwd

Returns (**$**name, **$**passwd, **$**uid, **$**gid, **$**quota, **$**comment, **$**gcos, **$**dir, **$**shell).

endpwent	Ends lookup processing.
getpwent	Gets next info.
getpwnam(NAME**)**	Gets info by name.
getpwuid(UID**)**	Gets info by uid.
setpwent	Resets lookup processing.

group

Returns (**$**name, **$**passwd, **$**gid, **$**members).

endgrent	Ends lookup processing.
getgrgid(GID**)**	Gets info by group id.
getgrnam(NAME**)**	Gets info by name.
getgrent	Gets next info.
setgrent	Resets lookup processing.

hosts

Returns (**$**name, **$**aliases, **$**addrtype, **$**length, @addrs).

endhostent	Ends lookup processing.
gethostbyaddr(ADDR**,**ADDRTYPE**)**	Gets info by address.
gethostbyname(NAME**)**	Gets info by name.
gethostent	Gets next info.
sethostent(STAYOPEN**)**	Resets lookup processing.

networks

Returns (**$**name, **$**aliases, **$**addrtype, **$**net).

endnetent	Ends lookup processing.
getnetbyaddr(ADDR**,**TYPE**)**	Gets info by address and type.
getnetbyname(NAME**)**	Gets info by name.
getnetent	Gets next info.
setnetent(STAYOPEN**)**	Resets lookup processing.

services

Returns (**$**name, **$**aliases, **$**port, **$**proto).

endservent	Ends lookup processing.
getservbyname(NAME**,** PROTO**)**	Gets info by name.
getservbyport(PORT**,** PROTO**)**	Gets info by port.
getservent	Gets next info.
setservent(STAYOPEN**)**	Resets lookup processing.

protocols

Returns (**$**name, **$**aliases, **$**proto).

endprotoent	Ends lookup processing.
getprotobyname(NAME**)**	Gets info by name.
getprotobynumber(NUMBER**)**	Gets info by number.
getprotoent	Gets next info.
setprotoent(STAYOPEN**)**	Resets lookup processing.

23. Regular expressions

Each character matches itself, unless it is one of the special characters **+?.*()[]{}|**.

. matches an arbitrary character, but not a newline.

(...) groups a series of pattern elements to a single element.

+ matches the preceding pattern element one or more times.

? matches zero or one times.

***** matches zero or more times.

{N,M**}** denotes the minimum N and maximum M match count. **{**N**}** means exactly N times; **{**N**,}** means at least N times.

[...] denotes a class of characters to match. **[^...]** negates the class.

(...|...|...) matches one of the alternatives.

Non-alphanumerics can be escaped from their special meaning using a ****.

\w matches alphanumeric, including "**_**", **\W** matches non-alphanumeric.

\b matches word boundaries, **\B** matches non-boundaries.

\s matches whitespace, **\S** matches non-whitespace.

\d matches numeric, **\D** matches non-numeric.

\n, **\r**, **\f**, **\t** etc. have their usual meaning.

\w, **\s** and **\d** may be used within character classes, **\b** denotes backspace in this context.

\1...**\9** refer to matched sub-expressions, grouped with **()**, inside the match.

\10 and up can also be used if the pattern matches that many sub-expressions.

See also **$1**...**$9**, **$+**, **$&**, **$`** and **$'** in section "Special Variables".

24. Special variables

The following variables are global and should be localized in subroutines:

$_ The default input and pattern-searching space.

$. The current input line number of the last filehandle that was read.

$/ The input record separator, newline by default. May be multi-character.

$, The output field separator for the print operator.

$" The separator which joins elements of arrays interpolated in strings.

$ The output record separator for the print operator.

$# The output format for printed numbers. Initial value is "**%.20g**".

$* Set to 1 to do multiline matching within a string, 0 to assume strings contain a single line. Default is 0.

$? The status returned by the last **`**COMMAND**`**, pipe close or **system** operator.

$] The perl version string (as displayed with **perl -v**), or version number.

$[The index of the first element in an array, and of the first character in a substring. Default is 0.

$; The subscript separator for multi-dimensional array emulation. Default is **"\034"**.

$! If used in a numeric context, yields the current value of errno. If used in a string context, yields the corresponding error string.

$@ The perl error message from the last eval or **do** EXPR command.

$: The set of characters after which a string may be broken to fill continuation fields (starting with "^") in a format.

$0 The name of the file containing the perl script being executed. May be assigned to.

$$ The process number of the perl running this script. Altered (in the child process) by **fork**.

$< The real uid of this process.

$> The effective uid of this process.

$(The real gid of this process.

$) The effective gid of this process.

$^D The debug flags as passed to perl using **-D** .

$^F The highest system file descriptor, ordinarily 2.

$^I In-place edit extension as passed to perl using **-i** .

$^L Formfeed character used in formats.

$^P Internal debugging flag.

$^T The time (as delivered by **time**) when the program started. This value is used by the file test operators "**-M**", "**-A**" and "**-C**".

$^W The value if the **-w** option as passed to perl.

$^X The name by which this perl was invoked.

The following variables are context dependent and need not be localized:

$% The current page number of the currently selected output channel.

$= The page length of the current output channel. Default is 60 lines.

$- The number of lines left on the page.

$~ The name of the current report format.

$^ The name of the current top-of-page format.

$| If set to nonzero, forces a flush after every write or print on the currently selected output channel. Default is 0.

$ARGV The name of the current file when reading from **<>** .

The following variables are always local to the current block:

$& The string matched by the last successful pattern match.

$` The string preceding what was matched by the last successful match.

$' The string following what was matched by the last successful match.

$+ The last bracket matched by the last search pattern.

$1...$9...
Contains the subpattern from the corresponding set of parentheses in the last pattern successfully matched. **$10...** and up are only available if the match contained that many sub-expressions.

25. Special arrays

@ARGV Contains the command line arguments for the script (not including the command name).

@INC Contains the list of places to look for perl scripts to be evaluated by the **do** FILENAME and **require** commands.

@_ Parameter array for subroutines. Also used by **split** if not in array context.

%ENV Contains the current environment.

%INC List of files that have been **require**d or **do**ne.

%SIG Used to set signal handlers for various signals.