## Regular expresions

Standard UNIX regular expressions, with the following additions:

( ) | { }
: do not have to be escaped with a \ .

\+ : matches the preceding pattern element one or more times.

? : matches zero or one times.

{*nn*,*mm*}
: denote the minimum (*nn*) and maximum (*mm*) match count.

\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 and \NNN have their usual meaning.

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

$1 ... $9 can be used to refer to matched sub-expressions outside the matching.

$+ : returns whatever the last bracket match matched.

$& : returns the entire matched string.

$` : returns everything before the matched string.

$´ : returns everything after the matched string.

## 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, the '<' is repeated to denote the length of the field;

@>>>… right adjusted field;

@|||… centered field;

@* 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.

Perl version 3.0 Reference Card © 1989 Johan Vromans
Reference card version 3.0.1.1, november 1989.

## Special variables

$_ : 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.

$ , : The output field separator for the print operator.

$" : The separator which is used to join arrays which are interpolated in string context.

$\ : The output record separator for the print operator.

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

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

$= : The current page length (printable lines) of the currently selected output channel. Default is 60.

$– : 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.

$$ : The process number of the perl running this script.

$? : The status returned by the last backtick ( ` ` ) command, pipe close or system operator.

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

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

$´ : The string following what was matched by the last pattern match.

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

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

$0 : Contains the name of the file containing the perl script being executed. Any pattern matching clobbers $0.

$<digit>
: Contains the subpattern from the corresponding set of parentheses in the last pattern matched.

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

$] : The perl version string as displayed with "perl -v".

$; : 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 system error string.

$@ : The error message from the last eval command.

$< : The real uid of this process.

$> : The effective uid of this process.

$ ( : The real gid of this process.

$ ) : The effective gid of this process.

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

## Special arrays

@ARGV : Contains the command line arguments for the script. $ARGV[0] is the first argument, not the command name. See $0 for the command name.

@INC : Contains the list of places to look for perl scripts to be evaluated by the "do EXPR" command.

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

%ENV : Contains the current environment.

%SIG : Used to set signal handlers for various signals.

## The perl debugger

h : Prints out a help message.

s : Single steps.

c : Continues (until breakpoint or exit).

RET : Repeats last 's' or 'c'.

n : Single steps around subroutine call.

l [RANGE]
: lists a range of lines. RANGE may be a number, start-end, start+amount, or a subroutine name. If omitted, continues last 'l'.

L : Lists lines that have breakpoints or actions.

t : Toggles trace mode.

b [LINE]
: Sets breakpoint. If LINE is omitted, sets breakpoint at current line.

b SUBNAME
: Sets breakpoint at the subroutine.

S : Lists names of all subroutines.

d LINE
: Deletes breakpoint at the given line.

D : Deletes all breakpoints.

A : Deletes all line actions.

V [PACKAGE]
: Lists all variables in a package. Default package is main.

a LINE COMMAND
: Sets an action for line.

< COMMAND
: Sets an action to be executed before every debugger prompt.

> COMMAND
: Sets an action to be executed before every 's', 'c' or 'n' command.

! [ [–]NUMBER]
: Redo a debugging command. Default is previous command.

H [–NUMBER]
: Displays the last –number commands of more than one letter.

q : Quits. You may also use your EOF character.

COMMAND
: Executes command as a perl statement.

p EXPR
: Prints expr.

## System interaction

chdir ([EXPR])*
: Changes the working directory, $HOME if EXPR is omitted.

chroot(FILENAME†)*
: Changes the root for the following commands.

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) ( `command ` status). If ($? >> 8) is 0, exits with 255.

exec(LIST)*
: Executes the system command in LIST.

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

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

getlogin
: Returns the current login name from /etc/utmp.

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.

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. The status is returned in $? .

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

## Networking

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)
: 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,FLAGS)
: Sets the requested socket option.

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

socket(SOCKET,DOMAIN,TYPE,PROTOCOL)
: Creates a SOCKET in the DOMAIN with a given TYPE using the indicated PROTOCOL.

socketpair(SOCKET1,SOCKET2,DOMAIN,TYPE,PROTOCOL)
: As socket, but creates a pair of sockets.

## Miscellaneous

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

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

local(LIST)
: Declares the listed (scalar) variables to be local to the enclosing block, subroutine or eval.

package NAME
: Designates the current file (which must contain a perl sub-program) as a package, having a separate namespace.

reset [(EXPR)*]
: Clears variables and resets ?? searches so that they work again. The expression is interpreted as a list of single characters (hyphens allowed for ranges). All variables and arrays beginning with one of those letters are reset to their pristine state. If the expression is omitted, one-match searches (?pattern?) are reset to match again.

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.

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

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

## Info from system files

### /etc/passwd

Info is ($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. |

### /etc/group

Info is a 4-item array: ($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. |

### /etc/hosts

Info is a 5-element array: ($name, $aliases, $addrtype, $length, @addrs).

| | |
|---|---|
| endhostent( ) | Ends lookup processing. |
| gethostbyname(NAME) | Gets info by name. |
| gethostent( ) | Gets next info. |
| sethostent(STAYOPEN) | Resets lookup processing. |

### /etc/networks

Info is ($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. |

### /etc/services

Info is ($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. |

### /etc/protocols

Info is ($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. |

## File operations

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, in that order.

link(OLDFILE,NEWFILE)
: Creates a new filename linked to the old filename.

lstat(FILEHANDLEorEXPR)*
: Like stat, but does not traverse symbolic links.

mkdir(DIR,MODE)
: Creates a directory with given permissions. Sets $! on failure.

select(RBITS,WBITS,NBITS,TIMEOUT)
: Performs a select(2) system call with the same parameters.

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(FILEHANDLEorEXPR)*
: Returns a 13-element array ($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime, $ctime, $blksize, $blocks).

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 on each file of a list of files. The first two elements of the list must be the numerical access and modification times, in that order.

## 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.

## Input / Output

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

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 exists, 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 NULL 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 the file and associates it with FILEHANDLE. If FILEHANDLE is an expression, its value is used as the name of the real filehandle wanted. If FILENAME is omitted, the scalar variable of the same name as the FILEHANDLE contains the filename. Normally the file is opened for input, but the filename may be preceeded with '>' to select output mode, or with '>>' to select append mode. Selecting input mode with '<' is allowed. '+>' or '+<' selects read/write access.
  If the filename starts with '|' it is interpreted as a pipe to the command following the '|'-symbol. Likewise, a filename which ends with '|' is taken to be a pipe from the command.
  Open returns 1 upon success, undef otherwise, except for pipes.
  The parentheses may be omitted, if only a FILEHANDLE is specified.

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)".

*... continued ...*

*... Input / Output, continued*

read(FILEHANDLE,VAR,LENGTH)
: Read LENGTH bytes from the file into the variable. Returns number of bytes actually read.

seek(FILEHANDLE,POSITION,WHENCE)
: Randomly positions the file. Returns 1 upon success, 0 otherwise.

select(FILEHANDLE)*
: Sets the current default filehandle for output operations. Returns the previously selected filehandle.

sprintf(FORMAT,LIST)
: Returns a string formatted by (almost all of) the usual printf conventions.

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.

## Search and replace functions

[VAR =~] [m]/PATTERN/[i][o]
: Searches a string for a pattern. If no string is specified via the =~ or !~ operator, the $_ string is searched.
  If you prepend an "m" you can use any pair of characters as delimiters. If the final delimiter is followed by the optional letter "i", the matching is done in a case-insensitive manner. If you append "o", variables are interpolated only once. If used in a context that requires an array value, a pattern match returns an array consisting of the subexpressions matched by the parentheses in pattern, i.e. ($1, $2, $3...).

?PATTERN?
: This is just like the /pattern/ search, except that it matches only once between calls to the reset operator.

[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 (0).
  Optional modifiers: "g" indicates that all occurrences of the pattern are to be replaced; "e" indicates that the replacement string is to be interpreted as an expression, "i" and "o" as with /PATTERN/ matching.
  Any delimiter may replace the slashes; if single quotes are used, no interpretation is done on the replacement string.

study[(SCALAR†)*]
: Takes extra time to study SCALAR in anticipation of doing many pattern matches on the string before it is next modified.

[VAR =~] tr/SEARCHLIST/REPLACEMENTLIST/
: 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".

An * after the parameter list indicates that this function may be used both as a function or as a unary operator, without the parentheses.

The symbol † after a parameter indicates that this parameter will default to $_ if omitted.

## Arithmetic functions

atan2(X,Y)
: Returns the arctangent of X/Y 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 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( ).

## 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 as string of unsigned ints, and yields the bit at OFFSET. BITS must be between 1 and 32. May be used as an lvalue.

## Converting binary data

pack(TEMPLATE,LISTorARRAY)
: 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 |
| c / C | Native/unsigned char value |
| s / S | Signed/unsigned short value |
| i / I | Signed/unsigned integer value |
| l / L | Signed/unsigned long value |
| n / N | Short/long in network byte order |
| p | Pointer to a string |
| x | Null byte |

Each character may be followed by a decimal number which will be used as a repeat count.

## String functions

chop(VARIABLE†)*
: Chops off the last character of a string and returns it.

chop(LIST)
: Chops off the last character on all elements of the list, returns the last.

crypt(PLAINTEXT,SALT)
: Encrypts a string.

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

index(STR,SUBSTR)
: Returns the position of SUBSTR in STR,. If the substring is not found, returns $[ – 1.

length(EXPR)*
: Returns the length in characters of the value of EXPR.

rindex(SR,SUBSTR)
: Returns the position of the last occurrence of SUBSTR in STR.

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.

## 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 the element). Returns array of elements from LIST for which EXPR returned true.

join(EXPR,LISTorARRAY)
: Joins the separate strings of LIST or ARRAY into a single string with fields separated by the value of EXPR, and returns the string.

keys(%ARRAY)*
: Returns a normal array consisting of all the keys of the named associative array.

pop(@ARRAY)*
: Pops and returns the last value of the array, shortening the array by 1.

push(@ARRAY,LIST)
: Treats ARRAY as a stack, and pushes the values of LIST onto the end of ARRAY. The length of ARRAY increases by the length of LIST.

reverse(LIST)*
: Returns the LIST in reverse order.

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 a -1, 0, or 1, depending on how the elements of the array, available to the routine as $a and $b, are to be ordered.

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 (/[ \t\n]+/).
If not in array context: returns number of fields and splits into @_ .

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.

# Perl Reference Card

for perl version 3.0, patchlevel 1.

Perl program designed and created by Larry Wall.

Reference card designed and created by Johan Vromans.

## Command line options

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

**–d** runs the script under the debugger.

**–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***extension*
specifies that files processed by the < > construct are to be edited in-place.

**–I***directory*
tells the C preprocessor where to look for include files. Use with **–P**.

**–n** causes perl to assume an input loop around your script. Lines are not printed.

**–p** causes perl to assume an input loop around your script. Lines are printed.

**–P** causes your script to be run through the C preprocessor before compilation by perl.

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

**–S** makes perl use the PATH environment variable to search for the script.

**–u** dumps core after compiling the script.

**–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.

## Literals

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

String: "abc" 'abc' . Variables are interpolated when "..." are used.
q/…/ can be used instead of '…', qq/…/ instead of "…",
(any delimiter instead of /.../).

Array: (1,2,3) . Also: ($a,$b,@rest) = (1,2,...);

Here-Is: <<identifier # the identifier must follow << immediately
input text
identifier

Filehandles:
<STDIN>, <STDOUT>, <STDERR>, <$var>.
< > are the lines read from standard input, or the input stream formed by the files specified in @ARGV.

Globs:
<pattern> evaluates to all filenames according to the pattern.

Commands:
` command ` evaluates to the output of the command.

## 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 |
| $#var | last index of array @var |
| @var | the entire array |
| @var[3,4,5] | a slice of the array @var |
| @var{'a','b'} | same as ($var{'a'},$var{'b'}) |
| %var | the entire associative array |

## Statements

Every statement is an expression, optionally followed by a modifier, and terminated by a semi-colon.

Execution of expressions can depend on other expressions using modifiers:

```
EXPR1 if EXPR2;       # expr1 only if expr2 evaluates false
EXPR1 unless EXPR2;   # expr1 only if expr2 evaluates true
EXPR1 while EXPR2;    # expr1 as long as expr2 evaluates false
EXPR1 until EXPR2;    # expr1 until expr2 evaluates true
```

Also, by using logical operators:

```
EXPR1 || EXPR2;       # expr2 only if expr1 evaluates false
EXPR1 && EXPR2;       # expr2 only if expr1 evaluates true
```

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

Compound statements may be used to control flow ( [ ] denote optional parts):

```
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.

## Flow control

do BLOCK
Returns the value of the last command in the sequence of commands indicated by BLOCK.

do SUBROUTINE (LIST)
Executes a SUBROUTINE declared by a sub declaration, and returns the value of the last expression evaluated in SUBROUTINE. Also: &SUBROUTINE.

do EXPR
Uses the value of EXPR as a filename and executes the contents of the file as a perl script.

goto LABEL
Continue execution at the specified label

*... continued ...*

*... Flow control, continued*

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.

## Operators

| | |
|---|---|
| + – * / | Addition, subtraction, multiplication, division. |
| % | Modulo division. |
| \|\| && | Logical or, logical and. |
| \| & ^ | Bitwise or, bitwise and, bitwise exclusive or. |
| >> << | Bitwise shift right, bitwise shift left. |
| ** | The exponentiation operator. |
| . | Concatenation of two strings. |
| x | The repetition operator. Returns a string consisting of the left operand repeated the number of times specified by the right operand. |

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

| | | |
|---|---|---|
| ?: | | Alternation operator. |
| ( ) | | The null list, used to initialize an array to null. |
| == | != | 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. |
| =~ | !~ | Search pattern, substitution, or translation (negated). |
| .. | | Enumeration, or the range operator, which is bistable. |

## File test operators

This unary operator 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 /-o | File is readable/writable/executable/owned by effective uid. |
| -R /-W /-X /-O | File is readable/writable/executable/owned by real uid. |
| -e /-z /-s | File exists / has zero/non-zero 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. Both -T and -B return TRUE on a null file, or a file at EOF when testing a filehandle. |