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.

-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**.

26. The perl debugger

The perl symbolic debugger is invoked with **perl -d**.

h	Prints out a help message.
T	Stack trace.
s	Single steps.
n	Single steps around subroutine call.
r	Returns from the current subroutine.
c [LINE]	Continues (until LINE, or another breakpoint or exit).
RET	Repeats last **s** or **n**.
l [RANGE]	Lists a range of lines. RANGE may be a number, start-end, start+amount, or a subroutine name. If omitted, lists next window.
f FILE	Switches to FILE and start listing it.
-	Lists previous window.
w	Lists window around current line.
l SUB	Lists the named SUBroutine.
/PATTERN**/**	Forward search for PATTERN.
?PATTERN**?**	Backward search for PATTERN.
L	Lists lines that have breakpoints or actions.
S	List the names of all subroutines.
t	Toggles trace mode.
b [LINE [CONDITION]]	Sets breakpoint at LINE, default: current line.
b SUBNAME [CONDITION]	Sets breakpoint at the subroutine.
S	Lists names of all subroutines.
d [LINE]	Deletes breakpoint at the given line.
D	Deletes all breakpoints.
a LINE COMMAND	Sets an action for line.
A	Deletes all line actions.
< 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.
V [PACKAGE [VARS]]	Lists all variables in a package. Default package is main.
X [VARS]	Like **V**, but assumes current package.
! [[-]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.
= [ALIAS VALUE]	Sets alias, or lists current aliases.

4. 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 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.

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.

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 $-\pi$ 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.

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 include not 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".

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.

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.

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.

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.

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**)**
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.

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".