Perl 5
Quick Reference
Guide

Workbook

October 1999
Perl 5
Quick Reference Guide

October 1999
EUCS Document Number: 3039
Copyright © 2004, 1989 Johan Vromans

It may be reproduced, printed and distributed freely for non-profit purposes, in the exact format as distributed through the Internet.

Where any part of this document is included in another document, due acknowledgement is required.
Programming Perl 5

Johan Vromans
Squirrel Consultancy
Contents

1. Command line options ........................................... 3
2. Syntax .............................................................. 4
3. Variables ........................................................... 4
4. Literals .............................................................. 5
5. Operators and precedence ......................................... 6
6. Statements ......................................................... 7
7. Subroutines, packages and modules .............................. 7
8. Pragmatic modules ................................................ 9
9. Object oriented programming .................................... 10
10. Arithmetic functions ............................................ 10
11. Conversion functions .......................................... 11
12. Structure conversion ............................................ 11
13. String functions ................................................ 12
14. Array and hash functions ..................................... 12
15. Regular expressions ............................................ 14
16. Search and replace functions .................................. 15
17. File test operators .............................................. 16
18. File operations ................................................ 16
19. Input / Output ................................................... 17
20. Formats ............................................................ 19
21. Directory reading routines ..................................... 19
22. System interaction .............................................. 19
23. Networking ........................................................ 21
24. SystemV IPC ...................................................... 21
25. Miscellaneous .................................................... 22
26. Information from system files .................................. 23
27. Special variables ................................................ 24
28. Special arrays .................................................... 25
29. Standard modules ............................................... 26
30. Environment variables ......................................... 30
31. The perl debugger .............................................. 30

Conventions

**fixed** denotes text that you enter literally.

**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.
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. It does run BEGIN and END blocks.
- **-d [ :DEBUGGER ]**
  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 a single line of script. Multiple -e commands may be given to build up a multi-line script.
- **-F REGEXP**
  specifies a regular expression to split on if -a is in effect.
- **-h** prints the Perl usage summary. Does not execute.
- **-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.
- **-m MODULE**
  imports the MODULE before executing the script. MODULE may be followed by a ‘=’ and a comma-separated list of items.
- **-M MODULE**
  Same as -m, but with more trickery.
- **-n** assumes an input loop around the script. Lines are not printed.
- **-P** assumes an input loop around the script. Lines are printed.
- **-p** runs the C preprocessor on the script before compilation by Perl.
- **-s** interprets ‘-xxx’ on the command line as a switch and sets the corresponding variable $xxx in the script.
- **-S** uses the PATH environment variable to search for the script.
- **-T** turns on taint checking.
- **-u** dumps core after compiling the script. To be used with the undump program (where available).
- **-U** allows Perl to perform unsafe operations.
- **-v** prints the version and patchlevel of your Perl executable.
- **-V [ :VAR ]**
  prints Perl configuration information.
- **-w** prints warnings about possible spelling errors and other error-prone constructs in the script.
- **-x [ DIR ]**
  extracts Perl program from the input stream. If DIR is specified, switches to this directory before running the program.
- **-0 [ VAL ]**
  (that’s the number zero) designates an initial value for the record separator $/. See also -l.

Command line options may be specified on the ‘#!’ line of the perl script, except for -M, -m and -T.
2. Syntax

Perl is a free-format programming language. This means that in general it does not matter how the Perl program is written with regard to indentation and lines. An exception to this rule is when the Perl compiler encounters a ‘sharp’ symbol (#) in the input: it then discards this symbol and everything it follows up to the end of the current input line. This can be used to put comments in Perl programs. Real programmers put lots of useful comments in their programs. There are places where whitespace does matter: within literal texts, patterns and formats. If the Perl compiler encounters the special token \_\_END\_\_ it discards this symbol and stops reading input. Anything following this token is ignored by the Perl compiler, but can be read by the program when it is run.

3. Variables

$var a simple scalar variable.
$var[28] 29th element of array @var.
$p = \@var now $p is a reference to array @var.
$$p[28] 29th element of array referenced by $p. Also: $p->[28].
$var[-1] last element of array @var.
$var[$i][$j] $j-th element of $i-th element of array @var.
$var{'Feb'} a value from ‘hash’ (associative array) %var.
$p = \%var now $p is a reference to hash %var.
$$p{'Feb'} a value from hash referenced by $p. Also: $p->{‘Feb’}.
$#var last index of array @var.
@var the entire array; in a scalar context, the number of elements in the array.
@var[3,4,5] a slice of array @var.
@var{‘a’,‘b’} a slice of %var; same as { $var{‘a’}, $var{‘b’} }.
%var the entire hash; in a scalar context, true if the hash has elements.
$var{‘a’,1,...} emulates a multi-dimensional array.
(‘a’..‘z’) [4,7,9] a slice of an array literal.
PKG::VAR a variable from a package, e.g. $pkg::var, @pkg::ary.
\THINGIE reference to a thingie, e.g. $\var, \%hash.
*NAME refers to all thingies represented by NAME.
‘*n1 = *n2’ makes n1 an alias for n2.
‘*n1 = \$n2’ makes $n1 an alias for $n2.

You can always use a { BLOCK } returning the right type of reference instead of the variable identifier, e.g. ${ ..., &{ ... } }. $$p is just a shorthand for ${ $p }.
4. Literals

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

String: 'abc' literal string, no variable interpolation nor escape characters, except \ and \\.
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 lowercase/upcase the following character;
\L and \U lowercase/upcase until a \E is encountered.
\Q quote regexp characters until a \E is encountered.

'COMMAND' evaluates to the output of the COMMAND.
Also: qx/COMMAND/.

Boolean: Perl has no boolean data type. Anything that evaluates to the null string, the number zero or the string "0" is considered false, everything else is true (including strings like "00"!)

Array: (1, 2, 3) a three member array. () is an empty array.
    (1..4) is the same as (1, 2, 3, 4). Likewise ('abc'..'ade').
    qw/foo bar .../ is the same as ('foo', 'bar', ...).
Array reference: [1, 2, 3].

Hash (associative array): (KEY1, VAL1, KEY2, VAL2, ...).
    Also: (KEY1 => VAL1, KEY2 => VAL2, ...).
Hash reference: {KEY1, VAL1, KEY2, VAL2, ...}.

Code reference: sub { STATEMENTS }

Filehandles: STDIN, STDOUT, STDERR, ARGV, DATA.

User-specified: HANDLE, $VAR.

Globs: <PATTERN> evaluates to all filenames according to the pattern.
    Use ‘<$VAR>’ or ‘glob $VAR’ to glob from a variable.

Here-Is: <<IDENTIFIER    Shell-style ‘here document’.

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

5. Operators and precedence

Perl operators have the following associativity and precedence, listed from highest precedence to lowest.

<table>
<thead>
<tr>
<th>Assoc</th>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>terms and list operators</td>
<td>See below.</td>
</tr>
<tr>
<td>left</td>
<td>--&gt;</td>
<td>Infix dereference operator.</td>
</tr>
<tr>
<td></td>
<td>++</td>
<td>Auto-increment (magical on strings).</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Auto-decrement.</td>
</tr>
<tr>
<td>right</td>
<td>**</td>
<td>Exponentiation.</td>
</tr>
<tr>
<td>right</td>
<td>\</td>
<td>Reference to an object (unary).</td>
</tr>
<tr>
<td>right</td>
<td>! ~</td>
<td>Unary negation, bitwise complement.</td>
</tr>
<tr>
<td>right</td>
<td>+ -</td>
<td>Unary plus, minus.</td>
</tr>
<tr>
<td>left</td>
<td>=~</td>
<td>Binds a scalar expression to a pattern match.</td>
</tr>
<tr>
<td>left</td>
<td>!˜</td>
<td>Same, but negates the result.</td>
</tr>
<tr>
<td>left</td>
<td>*/%x</td>
<td>Multiplication, division, modulo, repetition.</td>
</tr>
<tr>
<td>left</td>
<td>+= .</td>
<td>Addition, subtraction, concatenation.</td>
</tr>
<tr>
<td>right</td>
<td>&gt;&gt; &lt;&lt;</td>
<td>Bitwise shift right, bitwise shift left.</td>
</tr>
<tr>
<td>named unary operators</td>
<td>E.g. sin, chdir, -f, -M</td>
<td></td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>Numerical relational operators.</td>
<td></td>
</tr>
<tr>
<td>lt gt le ge</td>
<td>String relational operators.</td>
<td></td>
</tr>
<tr>
<td>== !</td>
<td>= &lt;=&lt;&gt;</td>
<td>Numerical equal, not equal, compare.</td>
</tr>
<tr>
<td>eq ne cmp</td>
<td>Stringwise equal, not equal, compare. Compose operators return -1 (less), 0 (equal) or 1 (greater).</td>
<td></td>
</tr>
<tr>
<td>left &amp;</td>
<td>Bitwise AND.</td>
<td></td>
</tr>
<tr>
<td>left</td>
<td></td>
<td>Bitwise OR, exclusive OR.</td>
</tr>
<tr>
<td>left &amp;&amp;</td>
<td>Logical AND.</td>
<td></td>
</tr>
<tr>
<td>left</td>
<td></td>
<td></td>
</tr>
<tr>
<td>left ..</td>
<td>In scalar context, range operator. In array context, enumeration.</td>
<td></td>
</tr>
<tr>
<td>right ?:</td>
<td>Conditional (if ? then : else) operator.</td>
<td></td>
</tr>
<tr>
<td>right = += -= *= etc.</td>
<td>Assignment operators.</td>
<td></td>
</tr>
<tr>
<td>left ,</td>
<td>Comma operator, also list element separator. Same, enforces the left operand to be a string.</td>
<td></td>
</tr>
<tr>
<td>list operators (rightward)</td>
<td>See below.</td>
<td></td>
</tr>
<tr>
<td>right not</td>
<td>Low precedence logical NOT.</td>
<td></td>
</tr>
<tr>
<td>left and</td>
<td>Low precedence logical AND.</td>
<td></td>
</tr>
<tr>
<td>left or xor</td>
<td>Low precedence logical OR, exclusive OR.</td>
<td></td>
</tr>
</tbody>
</table>

Parentheses can be used to group an expression into a term.

A ‘list’ is a list of expressions, variables or lists, separated by commas. An array variable or an array slice may always be used instead of a list.

All Perl functions can be used as list operators, in which case they have very high or very low precedence, depending on whether you look at the left side of the operator or at the right side of the operator.

Parentheses can be added around the parameter lists to avoid precedence problems.

The logical operators do not evaluate the right operand if the result is already known after evaluation of the left operand.
6. Statements

Every statement is an expression, optionally followed by a modifier, and terminated with 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.:

```plaintext
EXPR1 if EXPR2 ;
EXPR1 until EXPR2 ;
```

The logical operators ||, &&, or ?: also allow conditional execution, e.g.:

```plaintext
EXPR1 || EXPR2 ;
EXPR1 ? EXPR2 : EXPR3 ;
```

Statements can be combined to form a BLOCK when enclosed in {}. BLOCKs may be used to control flow:

```plaintext
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 ([ EXPRESSION ]; [ EXPRESSION ]; [ EXPRESSION ]) BLOCK
[ LABEL: ] foreach VAR | (LIST) BLOCK [ continue BLOCK ]
[ LABEL: ] BLOCK [ continue BLOCK ]
```

Program flow can be controlled with:

```plaintext
goto LABEL
```

Finds the statement labeled with LABEL and resumes execution there. LABEL may be an expression that evaluates to the name of a label.

```plaintext
last [ LABEL ]
```

Immediately exits the loop in question. Skips continue block.

```plaintext
next [ LABEL ]
```

Starts the next iteration of the loop.

```plaintext
redo [ LABEL ]
```

Restarts the loop block without evaluating the conditional again.

Special forms are:

```plaintext
do BLOCK while EXPR ;
do BLOCK until EXPR ;
```

which are guaranteed to perform BLOCK once before testing EXPR, and

```plaintext
do BLOCK
```

which effectively turns BLOCK into an expression.

7. Subroutines, packages and modules

SUBROUTINE [ LIST ]

Executes a SUBROUTINE declared by a preceding sub declaration, and returns the value of the last expression evaluated in SUBROUTINE.

SUBROUTINE can be an expression yielding a reference to code. In this case you can use &$\{EXPR\} ([LIST]) or $\{EXPR\} -> ([LIST])

```plaintext
&SUBROUTINE ([ LIST ])
```

Executes a SUBROUTINE not neccessarily declared before being used.

bless REF [, CLASSNAME ]

Turns the object REF into an object in CLASSNAME. Returns the reference.
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.. Returns false if no caller.

do SUBROUTINE LIST
   Deprecated form of &SUBROUTINE.

goto &SUBROUTINE
   Substitutes a call to SUBROUTINE for the current subroutine.

import MODULE [ VERSION ] [ LIST ]
   Imports the named items from MODULE. Checks the module for the
   required VERSION.

no MODULE [ LIST ]
   Cancels imported semantics. See use.

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

prototype NAME
   Returns the prototype for this function.

require EXPR
   If EXPR is numeric, requires Perl to be at least that version. Otherwise
   EXPR must be the name of a file that is included from the Perl library. Does
   not include more than once, and yields a fatal error if the file does not
   evaluate to a true value.
   If EXPR is a bare word, assumes extension '.pm' for the name of the file.
   This form of loading of modules does not risk altering your namespace.

return EXPR
   Returns from a subroutine with the value specified.

sub NAME [ ( PROTO ) ] { EXPR ; ... }
   Designates NAME as a subroutine. Parameters are passed by reference as
   array @_. Returns the value of the last expression evaluated.
   PROTO can be used to define the required parameters.
   Without a BLOCK it is a forward declaration, without the NAME it is an
   anonymous subroutine. Functions that have an empty prototype and do
   nothing but return a fixed value are inlined.

[ sub ] BEGIN { EXPR ; ... }
   Defines a setup BLOCK to be called before execution.

[ sub ] END { EXPR ; ... }
   Defines a cleanup BLOCK to be called upon termination.

tie VAR, CLASSNAME, [ LIST ]
   Ties a variable to a package class that will handle it. Can be used to bind a
dbm or ndbm file to a hash.

tied VAR
   Returns a reference to the object underlying VAR, or the undefined value if
   VAR is not tied to a package class.

untie VAR
   Breaks the binding between the variable and the package class.

use VERSION
   Requires perl version.

use MODULE [ VERSION ] [ LIST ]
   Imports semantics from the named module into the current package.
Standard methods

The **UNIVERSAL** package contains the following methods that are inherited by all other classes:

**isa CLASS**
- Returns **true** if its object is blessed into a subclass of **CLASS**.

**can METHOD**
- Returns a reference to the method if its object has it, **undef** otherwise.

**VERSION [ NEED ]**
- Returns the version of the class. Checks the version if **NEED** is supplied.

### 8. Pragmatic modules

Pragmatic modules affect the compilation of your program. Pragmatic modules can be activated (imported) with **use**, and deactivated with **no**. These are locally scoped.

**autouse MODULE => SUBS**
- Defers **require** until one of the subs is called.

**blib [DIR]**
- Used for testing of uninstalled packages.

**constant NAME = VALUE**
- Defines **NAME** to have a constant (compile-time) value.

**diagnostics**
- Force verbose warning diagnostics.

**integer**
- Compute arithmetic in integer instead of double precision.

**less**
- Request less of something from the compiler.

**lib**
- Manipulate **@INC** at compile time.

**locale**
- Enable POSIX locales.

**ops**
- Restrict unsafe operations when compiling.

**overload**
- Package for overloading Perl operators.
  Example: **use overload "+" => \my_add;**

**sigtrap**
- Enable simple signal handling.
  Example: **use sigtrap qw(SEGV TRAP);**

**strict**
- Restrict unsafe constructs.
  **use strict "refs"** restricts the use of symbolic references.
  **use strict "vars"** requires all variables to be either local or fully qualified.
  **use strict "subs"** restricts the use of bareword identifiers that are not subroutines.

**subs**
- Predeclare subroutine names, allowing you to use them without parentheses even before they are declared.
  Example: **use subs qw(ding dong);**

**vars**
- Predeclare variable names, allowing you to use them under “use strict”.
  Example: **use vars qw($foo @bar);**

**vmsish**
- Emulate some VMS behaviour.
9. Object oriented programming

Perl rules of object oriented programming:

- An object is simply a reference that happens to know which class it belongs to. Objects are blessed, references are not.
- A class is simply a package that happens to provide methods to deal with object references. If a package fails to provide a method, the base classes as listed in @ISA are searched.
- A method is simply a subroutine that expects an object reference (or a package name, for static methods) as the first argument. Methods can be applied with:
  - METHOD OBJREF PARAMETERS or OBJREF->METHOD PARAMETERS

10. Arithmetic functions

absolute value

abs EXPR
  Returns the absolute value of its operand.

arctangent

atan2 Y, X
  Returns the arctangent of Y/X in the range -π to π.

cosine

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

exponentiation

exp EXPR
  Returns e to the power of EXPR.

integer portion

int EXPR
  Returns the integer portion of EXPR.

natural logarithm

log EXPR
  Returns natural logarithm (base e) of EXPR.

random number

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.

sine

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

square root

sqrt EXPR
  Returns the square root of EXPR.

random number seed

srand [ EXPR ]
  Sets the random number seed for the rand operator.

time

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

**chr** EXPR

Returns the character represented by the decimal value EXPR.

**gmtime** EXPR

Converts a time as returned by the `time` function to a 9-element array (0:$sec, 1:$min, 2:$hour, 3:$mday, 4:$mon, 5:$year, 6:$wday, 7:$yday, 8:$isdst) with the time localized for the standard Greenwich time zone. $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 `ctime(3)` string. In array context, returns a 9-element array (see `gmtime`) with the time localized for the local time zone.

**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 string EXPR as a vector of unsigned integers of BITS bits each, and yields the decimal value of the element at OFFSET. BITS must be a power of 2 between 1 and 32. May be assigned to.

12. 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
- P A pointer to a structure (fixed-length string)
- v / V Short / long in VAX (little endian) byte order
- w / x BER compressed integer / null byte
- X / @ Backup a byte / null fill until position

Each character may be followed by a decimal number which will be used as a repeat count, ‘*’ 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.
13. String functions

chomp LIST
   Removes line endings from all elements of the list; returns the (total) number of characters removed.

chop LIST
   Chops off the last character on all elements of the list; returns the last chopped character.

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, undef is returned by eval, and $@ is set to the error message. See also eval in section ‘Miscellaneous’.

index STR, SUBSTR [, OFFSET ]
   Returns the position of SUBSTR in STR at or after OFFSET. If the substring is not found, returns -1 (but see $ in section ‘Special variables’).

length EXPR
   Returns the length in characters of EXPR.

lc EXPR
   Returns a lower case version of EXPR.

lcfirst EXPR
   Returns EXPR with the first character in lower case.

quotemeta EXPR
   Returns EXPR with all regexp meta-characters quoted.

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. If LEN is negative, leaves that many characters off the end of the string. May be assigned to.

uc EXPR
   Returns an upper case version of EXPR.

ucfirst EXPR
   Returns EXPR with the first character in upper case.

14. Array and hash functions

delete $HASH{KEY}
   Deletes the specified value from the specified hash. Returns the deleted value (unless HASH is tied to a package that does not support this).

each %HASH
   Returns a 2-element array consisting of the key and value for the next value of the hash. After all values of the hash have been returned, an empty list is returned. The next call to each after that will start iterating again.

exists EXPR
   Checks whether the specified hash key exists in its hash array.
grep EXPR, LIST
grep BLOCK LIST
Evaluates EXPR or BLOCK for each element of the LIST, locally setting $__ to refer to the element. Modifying $__ will modify the corresponding element from LIST. Returns the 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 %HASH
Returns an array of all the keys of the named hash.

map EXPR, LIST
map BLOCK LIST
Evaluates EXPR or BLOCK for each element of the LIST, locally setting $__ to refer to the element. Modifying $__ will modify the corresponding element from LIST. Returns the list of results.

pop [ @ARRAY ]
Pops off and returns the last value of the array. If @ARRAY is omitted, pops @ARGV in main and @_ in subroutines.

push @ARRAY, LIST
Pushes the values of the list onto the end of the array.

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 %HASH
Returns a true value if the hash 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 package global variables $a and $b, are to be ordered. SUBROUTINE may be the name of a user-defined routine, or a BLOCK.

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 into at most that number of fields. If PATTERN is omitted, splits on whitespace (after skipping any leading whitespace). If not in array context: returns number of fields and splits to @__.

unshift @ARRAY, LIST
Prepends list to the front of the array, and returns the number of elements in the new array.

values %HASH
Returns a normal array consisting of all the values of the named hash.
15. Regular expressions

Each character matches itself, unless it is one of the special characters 
+?.*\$()\{\}\|. The special meaning of these characters can be escaped using a `\`.

. matches an arbitrary character, but not a newline unless the modifier `/s` is used.

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

^ matches the beginning of the target. In multi-line mode (see m//m) also matches after every newline character.

$ matches the end of the line. In multi-line mode also matches before every newline character.

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

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

(?# TEXT ) Comment.

( ?: REGEXP ) Like ( REGEXP ) but does not make back-references.

( ?= REGEXP ) Zero width positive look-ahead assertion.

( ?! REGEXP ) Zero width negative look-ahead assertion.

(? MODIFIER ) Embedded pattern-match modifier. MODIFIER can be one or more of i, m, s or x.

Quantified subpatterns match as many times as possible. When followed with a `?` they match the minimum number of times. These are the quantifiers:

+ 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.

A `\` escapes any special meaning of the following character if non-alphanumeric, but it turns most alphanumeric characters into something special:

\w matches alphanumeric, including `"`, \W matches non-alphanumeric.

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

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

\A matches the beginning of the string, \Z matches the end.

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

\G matches where the previous m//g search left off.

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

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

Back-references:

\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’.

With modifier x, whitespace and comments can be used in the patterns for readability purposes.
16. Search and replace functions

[ EXPR =~ ] [ m ] /PATTERN/ [ c g i m o s x ]

Searches EXPR (default: $__) for a pattern. If you prepend an m you can use almost any pair of delimiters instead of the slashes. If used in array context, an array is returned consisting of the sub-expressions matched by the parentheses in pattern, i.e. ($1, $2, $3, ...).

Optional modifiers: c continues the previous match (use with g); g matches as many times as possible; i searches in a case-insensitive manner; o interpolates variables only once.

m let ‘^’ and ‘$’ match even at embedded newline characters; s let ‘.’ match even at embedded newline characters; x allows for regular expression extensions.

If PATTERN is empty, the most recent pattern from a previous successful match or replacement is used. With g the match can be used as an iterator in scalar context. The iterator is reset upon failure, unless c is also supplied.

?PATTERN?

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

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

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

Optional modifiers: g replaces all occurrences of the pattern; e evaluates the replacement string as a Perl expression; for the other modifiers, see /PATTERN/ matching. Almost any delimiter may replace the slashes; if single quotes are used, no interpolation is done on the strings between the delimiters, otherwise the strings are interpolated as if inside double quotes.

If bracketing delimiters 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 successful match or replacement is used.

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

Translates all occurrences of the characters found in the search list into 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 found in SEARCHLIST that do not have a corresponding character in REPLACEMENTLIST; s squeezes all sequences of characters that are translated into the same target character into one occurrence of this character.

pos [ SCALAR ]

Returns the position where the last m//g search left off for SCALAR. May be assigned to.

study [ $VAR ]

Studies the scalar variable $VAR in anticipation of performing many pattern matches on its contents before the variable is next modified.
17. File test operators

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

- 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 modification/access/inode-change time. Measured in days. Value returned reflects the file age at the time the script started. See also $^T in section ‘Special variables’.

18. 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 $! on failure.
stat FILE
Returns a 13-element array (0:$dev, 1:$ino, 2:$mode, 3:$nlink, 4:$uid, 5:$gid, 6:$rdev, 7:$size, 8:$atime, 9:$mtime, 10:$ctime, 11:$blksize, 12:$blocks). FILE can be a filehandle, an expression evaluating to a filename, or _ to refer to the last file test operation or stat call. Returns an empty 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.

19. Input / Output

In input/output operations, FILEHANDLE may be a filehandle as opened by the open operator, a pre-defined filehandle (e.g. STDOUT) or a scalar variable that evaluates to a reference to or the name of a filehandle to be used.

<FILEHANDLE>
In scalar context: reads a single line from the file opened on FILEHANDLE.
In array context: reads the whole file.

<>
Reads from the input stream formed by the files specified in @ARGV, or standard input if no arguments were supplied.

binmode FILEHANDLE
Arranges for the file opened on FILEHANDLE to be read or written in binary mode as opposed to text mode (null-operation on UNIX).

close FILEHANDLE
Closes the file or pipe associated with the file handle.

dbmclose %HASH
Depreciated, use untie instead.

dbmopen %HASH, DBMNAME, MODE
Depreciated, use tie instead.

eof FILEHANDLE
Returns true 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
Performs fcntl(2) on the file. This function has non-standard return values.

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

flock FILEHANDLE, OP
Calls a system-dependent locking routine on the file. OP is formed by adding 1 (shared), 2 (exclusive), 4 (non-blocking) or 8 (unlock).

getc [ FILEHANDLE ]
Yields the next character from the file, or an empty string 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.
open FILEHANDLE [ , FILENAME ]

Opens a file and associates it with FILEHANDLE. open returns true upon success. If FILENAME is omitted, uses the scalar variable of the same name as the FILEHANDLE.

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 existing FILE with read/write access.

">>FILE"  create new FILE with read/write access.

">>>FILE"  read/write access in append mode.

"| CMD"  opens a pipe to command CMD; forks if CMD is ‘-‘.

"CMD | "  opens a pipe from command CMD; forks if CMD is ‘-‘.

FILE may be &FILEHND, in which case the new file handle is connected to the (previously opened) filehandle FILEHND. If it is &=N, FILE will be connected to the given file descriptor.

pipe READHANDLE, WRITEHANDLE

Returns a pair of connected pipes.

print [ FILEHANDLE ] [ LIST ];

Prints the elements of LIST, converting them to strings if needed. 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 ];

Reads 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 true upon success.

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 in the style of printf(3) conventions.

sysopen FILEHANDLE, PATH, MODE [ , PERMS ];

Performs an open(2) system call. The possible values and flag bits of MODE are system-dependent; they are available via the standard module Fcntl.

sysread FILEHANDLE, $VAR, LENGTH [ , OFFSET ];

Reads LENGTH bytes into $VAR at OFFSET.

sysseek FILEHANDLE, POSITION, WHENCE

Performs a seek(2) system call.

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.
20. Formats

format PICTURE, LIST
    Formats LIST according to PICTURE and accumulates the result into $^A.
write [ FILEHANDLE ]
    Writes a formatted record to the specified file, using the format associated
    with that file.

Formats are defined as follows:

format [ NAME ] = FORMLIST
    FORMLIST pictures the lines, and contains the arguments which will give values to
    the fields in the lines. NAME defaults to STDOUT if omitted.

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.

Set $- to zero to force a page break on the next write.

See also $^~, $~, $^A, $^F, $- and $= in section ‘Special variables’.

21. 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) from the directory.

rewinddir DIRHANDLE
    Positions the directory to the beginning.

seekdir DIRHANDLE, POS
    Sets position for readdir on the directory.

telldir DIRHANDLE
    Returns the position in the directory.

22. System interaction

alarm EXPR]
    Schedules a SIGALRM to be delivered after EXPR seconds.

chdir [ EXPR ]
    Changes the working directory.

    Uses $ENV{ "HOME" } or $ENV{ "LOGNAME" } 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". Inside an eval, the error message is put into $@, and the eval is terminated
with undef; this makes die the way to raise an exception.

exec LIST
Executes the system command in LIST; does not return.

exit [ EXPR ]
Exits immediately with the value of EXPR, which defaults to 0 (zero). Calls
END routines and object destructors before exiting.

fork
Does a fork(2) system call. Returns the process ID of the child to the parent
process and zero to the child process.

getlogin
Returns the current login name as known by the system. If it returns false,
use getpwuid.

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.

glob PAT
Returns a list of filenames that match the shell pattern PAT.

kill LIST
Sends a signal to a list of processes. The first element of the list must be the
signal to send (either numeric, or its name as a string). Negative signals kill
process groups instead of processes.

setpgrp PID, PGRP
Sets the process group for the PID (0 indicates the current process).

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

sleep [ EXPR ]
Causes the program 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. Returns the
exit status of the child process.

times
Returns a 4-element array (0:$user, 1:$system, 2:$cuser, 3:$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 process ID 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 LIST on STDERR like die, but does not exit.
LIST defaults to "Warning: something’s wrong".

23. 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.

24. SystemV IPC

Depending on your system configuration, certain system files need to be required to access the message and semaphore specific facilities.

msgctl ID, CMD, ARGS
Calls msgctl(2). If CMD is IPC_STAT then ARGS must be a single variable. See the manual for details on the non-standard return values of this function.
msgget KEY, FLAGS
    Creates a message queue for KEY. Returns the message queue identifier.

msgsnd ID, MSG, FLAGS
    Sends MSG to queue ID.

msgrcv ID, $VAR, SIZE, TYPE, FLAGS
    Receives a message from queue ID into VAR.

semctl ID, SEMNUM, CMD, ARG
    Calls semctl(2).
    If CMD is IPC_STAT or GETALL then ARG must be a variable.

semget KEY, NSEMS, SIZE, FLAGS
    Creates a set of semaphores for KEY. Returns the message semaphore identifier.

semop KEY, ...
    Performs semaphore operations.

shmctl ID, CMD, ARG
    Calls shmct1(2). If CMD is IPC_STAT then ARG must be a single variable.

shmget KEY, SIZE, FLAGS
    Creates shared memory. Returns the shared memory segment identifier.

shmread ID, $VAR, POS, SIZE
    Reads at most SIZE bytes of the contents of shared memory segment ID starting at offset POS into VAR.

shmwrite ID, STRING, POS, SIZE
    Writes at most SIZE bytes of STRING into the contents of shared memory segment ID at offset POS.

25. Miscellaneous

defined EXPR
    Tests whether the EXPR has an actual value.

do FILENAME
    Executes FILENAME as a Perl script. See also require in section ‘Subroutines, packages and modules’.

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

eval { EXPR; ... }
    Executes the code between { and }. Traps run-time errors as described with eval(EXPR), section ‘String functions’.

local VAR
    Creates a scope for VAR local to the enclosing block, subroutine or eval.

my VAR
    Creates a scope for the variable lexically local to the enclosing block, subroutine or eval.

ref EXPR
    Returns a true value if EXPR is a reference. Returns the package name if EXPR has been blessed into a package.

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.

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

**wantarray**
Returns **true** if the current context expects an list value. **undef** if the current context does not expect a value at all, **false** otherwise.

### 26. Information 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 look-up processing.

**getpwent**
Gets next user information.

**getpwnam** NAME
Gets information by name.

**getpwuid** UID
Gets information by user ID.

**setpwent**
Resets look-up processing.

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

**endgrent**
Ends look-up processing.

**getgrgid** GID
Gets information by group ID.

**getgrnam** NAME
Gets information by name.

**getgrent**
Gets next group information.

**setgrent**
Resets lookup processing.

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

**endhostent**
Ends look-up processing.

**gethostbyaddr** ADDR, ADDRTYPE
Gets information by IP address.

**gethostbyname** NAME
Gets information by host name.

**gethostent**
Gets next host information.

**sethostent** STAYOPEN
Resets look-up processing.

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

**endnetent**
Ends look-up processing.

**getnetbyaddr** ADDR, TYPE
Gets information by address and type.

**getnetbyname** NAME
Gets information by network name.

**getnetent**
Gets next network information.

**setnetent** STAYOPEN
Resets look-up processing.

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

**endservent**
Ends look-up processing.

**getservbyname** NAME, PROTO
Gets information by service name.

**getservbyport** PORT, PROTO
Gets information by service port.

**getservent**
Gets next service information.
setservent STAYOPEN

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

endprotoent

getprotobynumber NUMBER

getprotobyname NAME

getprotoent

setprotoent STAYOPEN

27. 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. Reset only
        when the filehandle is closed explicitly.
$/      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. Deprecated.
$*     Set to 1 to do multiline matching within strings. Deprecated, see the m and s
        modifiers in section ‘Search and replace functions’.
$?     The status returned by the last ‘...’ command, pipe close or system
        operator.
$]     The Perl version number, e.g. 5.004.
$[     The index of the first element in an array, and of the first character in a
        substring. Default is 0. Deprecated.
$;     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 ‘ˆ ’) in a format.
$0     The name of the file containing the Perl script being executed. May be
        assigned to.
$$     The process ID of the Perl interpreter running this script. Altered (in the
        child process) by fork.
$<     The real user ID of this process.
$>     The effective user ID of this process.
$      The real group ID of this process.
$)     The effective group ID and groups of this process.
$ˆA    The accumulator for formline and write operations.
$ˆD    The debug flags as passed to Perl using ‘-D’.
$ˆE    Extended error message on some platforms.
$ˆF    The highest system file descriptor, ordinarily 2.
$ˆH    Set of syntax checks enabled by ‘use strict’.
In-place edit extension as passed to Perl using ‘-i’.

Formfeed character used in formats.

Out-of-memory emergency pool.

Internal debugging flag.

The time (as delivered by `time`) when the program started. This value is used by the file test operators ‘-M’, ‘-A’ and ‘-C’.

The value of the ‘-w’ option as passed to Perl.

The name by which this Perl interpreter 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 remaining 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 output channel currently selected. Default is 0.

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.

Contain the subpatterns from the corresponding sets of parentheses in the last pattern successfully matched. $10 and up are only available if the match contained that many subpatterns.

### 28. Special arrays

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

@EXPORT Names the methods a package exports by default.

@EXPORT_OK Names the methods a package can export upon explicit request.

@INC Contains the list of places to look for Perl scripts to be evaluated by the `do` FILENAME, `use` and `require` commands. Do not modify directly, but use the ‘`use lib`’ pragma or `-I` command line option instead.

@ISA List of base classes of a package.

@_ 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 included with `use`, `require` or `do`.

%SIG Used to set signal handlers for various signals. `__WARN__` and `__DIE__` are pseudo-signals to attach handlers to Perl warnings and exceptions.
29. Standard modules

AnyDBM_File
   Provides a framework for multiple dbm files.
AutoLoader
   Load functions only on demand.
AutoSplit
   Split a package for autoloading.
Benchmark
   Benchmark running times of code.
CGI  Web server Common Gateway Interface.
CGI::Apache
   Support for Apache’s Perl module.
CGI::Carp
   Log server errors with helpful context.
CGI::Fast
   Support for FastCGI (persistent server process).
CGI::Push
   Support for server push.
CGI::Switch
   Simple interface for multiple server types.
CPAN
   Interface to Comprehensive Perl Archive Network.
CPAN::FirstTime
   Utility for creating CPAN configuration file.
CPAN::Nox
   Run CPAN while avoiding compiled extensions.
Carp  Warn of errors.
Class::Struct
   Declare struct-like datatypes as Perl classes.
Config
   Access to Perl configuration information.
Cwd   Get the pathname of current working directory.
DB_File
   Access to Berkeley DB files.
Devel::SelfStubber
   Generate stubs for a SelfLoading module.
DirHandle
   Supplies object methods for directory handles.
Dynaloader
   Dynamically loads C libraries into Perl code.
English
   Use long English names for punctuation variables.
Env    Imports environment variables.
Exporter
   Implements default import method for modules.
ExtUtils::Embed
   Utilities for embedding Perl in C/C++ applications.
ExtUtils::Install
  Install files from here to there.
ExtUtils::Liblist
  Determine libraries to use and how to use them.
ExtUtils::MakeMaker
  Create an extension Makefile.
ExtUtils::Manifest
  Utilities to write and check a MANIFEST file.
ExtUtils::Miniperl
  Write the C code for perlmain.c.
ExtUtils::Mkbootstrap
  Make a bootstrap file for use by DynaLoader.
ExtUtils::Mksymlists
  Write linker options files for dynamic extension.
ExtUtils::MM_OS2
  Methods to override Unix behaviour in ExtUtils::MakeMaker.
ExtUtils::MM_Unix
  Methods used by ExtUtils::MakeMaker.
ExtUtils::MM_VMS
  Methods to override Unix behaviour in ExtUtils::MakeMaker.
ExtUtils::testlib
  Adds blib/* directories to @INC.
Fatal
  Replaces functions with equivalents which succeed or die.
Fcntl
  Loads the C fcntl.h defines.
File::Basename
  Parse file specifications.
FileCache
  Keep more files open than the system permits.
File::CheckTree
  Run many filetest checks on a tree.
File::Copy
  Copy files or filehandles.
File::Find
  Traverse a file tree.
FileHandle
  Supplies object methods for filehandles.
File::Path
  Create or remove a series of directories.
File::stat
  By-name interface to Perl’s builtin stat.
FindBin
  Locate the directory of the original Perl script.
GDBM_File
  Access to the gdbm library.
Getopt::Long
  Extended handling of command line options. Suits all needs.
Getopt::Std
  Process single-character switches with switch clustering.
I18N::Collate
  Compare 8-bit scalar data according to the current locale.

IO
  Loads various IO modules.

IO::File
  Supplies object methods for filehandles.

IO::Handle
  Supplies object methods for I/O handles.

IO::Pipe
  Supplies object methods for pipes.

IO::Seekable
  Supplies seek based methods for I/O objects.

IO::Select
  Object interface to the select system call.

IO::Socket
  Object interface to socket communications.

IPC::Open2
  Open a pipe to a process for both reading and writing.

IPC::Open3
  Open a pipe to a process for reading, writing, and error handling.

Math::BigFloat
  Arbitrary length float math package.

Math::BigInt
  Arbitrary size integer math package.

Math::Complex
  Complex numbers and associated mathematical functions.

Math::Trig
  Trigoniometric functions.

NDBM_File
  Tied access to ndbm files.

Net::hostent
  By-name interface to Perl’s builtin gethost functions.

Net::netent
  By-name interface to Perl’s builtin getnet functions.

Net::Ping
  Check a host for upness.

Net::protoent
  By-name interface to Perl’s builtin getproto functions.

Net::servent
  By-name interface to Perl’s builtin getserv functions.

Opcode
  Disable named opcodes when compiling Perl code.

Pod::Text
  Convert POD data to formatted ASCII text.

POSIX
  Interface to IEEE Std 1003.1.

Safe
  Compile and execute code in restricted compartments.

SDBM_File
  Tied access to sdbm files.
Search::Dict
  Search for key in dictionary file.
SelectSaver
  Save and restore a selected file handle.
SelfLoader
  Load functions only on demand.
Shell
  Run shell commands transparently within Perl.
Socket
  Load the C **socket.h** defines and structure manipulators.
Symbol
  Manipulate Perl symbols and their names.
Sys::Hostname
  Try every conceivable way to get the name of this system.
Sys::Syslog
  Interface to the Unix **syslog(3)** calls.
Term::Cap
  Perl interface to Unix **termcap(3)**.
Term::Complete
  Word completion module.
Term::ReadLine
  Interface to various readline packages.
Test::Harness
  Run Perl standard test scripts with statistics.
Text::Abbrev
  Create an abbreviation table from a list.
Text::ParseWords
  Parse text into an array of tokens.
Text::Soundex
  Implementation of the Soundex Algorithm as described by Donald Knuth.
Text::Tabs
  Expand and unexpand tabs.
Text::Wrap
  Line wrapping to form simple paragraphs.
Tie::Hash
  Base class definitions for tied hashes.
Tie::RefHash
  Base class for tied hashes with references as keys.
Tie::StdHash
  Basic methods for tied hashes.
Tie::Scalar
  Base class definitions for tied scalars.
Tie::StdScalar
  Basic methods for tied scalars.
Tie::SubstrHash
  Fixed table-size, fixed key-length hashing.
Time::gmtime
  By-name interface to Perl’s builtin **gmtime**.
Time::Local
  Efficiently compute time from local and GMT time.
30. Environment variables

Perl uses the following environment variables.

**HOME**  Used if `chdir` has no argument.

**LOGDIR**  Used if `chdir` has no argument and **HOME** is not set.

**PATH**  Used in executing subprocesses, and in finding the Perl script if ‘-S’ is used.

**PERL5LIB**  A colon-separated list of directories to look in for Perl library files before looking in the standard library and the current directory.

**PERL5DB**  The command to get the debugger code.

Defaults to `BEGIN { require 'perl5db.pl' }`.

**PERLLIB**  Used instead of **PERL5LIB** if the latter is not defined.

**PERL5OPT**  Used to set initial (command line) options for perl.

31. The perl debugger

The Perl symbolic debugger is invoked with ‘perl -d’.

- **h**  Prints out a long help message.
- **h CMD**  Prints out help for the command CMD.
- **h h**  Prints out a concise help message.
- **T**  Prints a stack trace.
- **s [ EXPR ]**  Single steps.
- **n [ EXPR ]**  Single steps around subroutine call.
- **RET**  Repeats last ‘s’ or ‘n’.
- **r**  Returns from the current subroutine.
- **c [ LINE ]**  Continues (until LINE, or another breakpoint, or exit).
- **p EXPR**  Prints EXPR.
- **l [ RANGE ]**  Lists a range of lines. RANGE may be a number, start–end, start+amount, or a subroutine name. If RANGE is omitted, lists next window.
- **w [ LINE ]**  Lists window around the specified line.
Lists previous window.
. Returns to the executed line.
\f FILE Switches to FILE and starts listing it.
1 SUB Lists the named subroutine.
S [ ! ] PATTERN Lists the names of all subroutines [not] matching the pattern.
/PATTERN/ Searches forwards for PATTERN.
? PATTERN? Searches backwards for PATTERN.
b [ LINE [ CONDITION ] ]
Sets breakpoint at LINE, default is the current line.
b SUB [ CONDITION ]
Sets breakpoint at the named subroutine.
d [ LINE ] Deletes breakpoint at the given line.
D Deletes all breakpoints.
L Lists lines that have breakpoints or actions.
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 after every debugger prompt.
V [ PACKAGE [ PATTERN ] ]
Lists variables matching PATTERN in a package. Default package is main.
X [ PATTERN ] Like ‘\v’, but assumes the current package.
! [ [-]NUMBER ]
Re-executes a command. Default is the previous command.
! [ PATTERN ] Re-executes the last command that started with PATTERN.
!! [ COMMAND ]
Runs COMMAND in a sub-process.
H [ -NUMBER ] Displays the last -NUMBER commands.
| CMD Runs debugger command CMD through the current pager.
| | CMD Same, temporarily selects DB: :OUT as well.
t Toggles trace mode.
t EXPR Traces through execution of EXPR.
x EXPR Evals EXPR in list context, dumps the result.
o [ OPT [=VAL] ]
Sets or queries values of debugger options.
= [ ALIAS VALUE ]
Sets alias, or lists current aliases.
R Restarts the debugger.
q Quits. You may also use your EOF character.
COMMAND Executes COMMAND as a Perl statement.