Perl Memoire

[Installing Modules 2](#_Toc74075334)

[strings 3](#_Toc74075335)

[string functions 3](#_Toc74075336)

[arrays 4](#_Toc74075337)

[syntax 4](#_Toc74075338)

[array functions 4](#_Toc74075339)

[hashes 5](#_Toc74075340)

[syntax 5](#_Toc74075341)

[hash functions 5](#_Toc74075342)

[files 6](#_Toc74075343)

[file functions 6](#_Toc74075344)

[maths 7](#_Toc74075345)

[math functions 7](#_Toc74075346)

[Module Data 8](#_Toc74075347)

[Data::Dumper 8](#_Toc74075348)

# types

## numbers

<All numbers in perl are handled as doubles?>

## scalar vs list context

## boolean truthiness

Perl does not have a specific boolean type.

The following evaluate to *false*:

An empty string

Numerical zero

An empty list

The literal string *“0”* *(note that “0.0”, “00”, “+0”, ect. are true as strings)*

An undefined value

An object with an overloaded Boolean operator that evaluates to one of the above

A magical variable that evaluates to one of the above

Anything else evaluates to *true*.

# perlrun

## Executing Perl

Specify the source for the perl shell command:

1. Line-by-line, as text argument to ‘-e’ or ‘-E’
2. Contained in the first filename in the command, only if *1)* not specified first
3. Via standard-input, only if neither *1)* or *2)* are given.

## Options

-a Split line (specify delimiter -F’delim’)

-F <>

<…>

# one-liner core tools

# installing modules

cpan <module>

# [special variables](https://perldoc.perl.org/perlvar)

Enable long variable names with:

use English;

Behaviour of the default variable – the following pairs are all equivalent:

while (<>) {...}

while (defined($\_ = <>)) {...}

/REGEX/

$\_ =~ /REGEX/

tr/a-z/A-Z/

$\_ =~ tr/a-z/A-Z/

chomp

chomp($\_)

## global scalars

$\_ $ARG

The default input and pattern-searching space.

@\_ @ARG

In a subroutine, contains parameters passed to said subroutine, and is the default array for *pop()* and *shift()*

$” $LIST\_SEPARATOR

When an array or array slice is interpolated, its elements are separated by this value.

$ARGV

Name of current file when reading from <>

$, $OFS $OUTPUT\_FIELD\_SEPERATOR

If defined, value is printed between each argument given to *print()*. Default is *undef*.

$. $NR $INPUT\_LINE\_NUMBER

Current line-number for the last filehandle accessed

$/ $RS $INPUT\_RECORD\_SEPARATOR

Influences perl’s idea of what a ‘line’ is. Default is *‘\n’*. Set to *undef* to read to the end of the file as a line. Set to *“”* to read until blank line (and treat consecutive empty lines as a single empty line) as a line. Set to reference to number to read that many bytes as a line. Value is a string, not a regex.

$\ $ORS $OUTPUT\_RECORD\_SEPERATOR

If defined, value is printed after last argument given to *print()*. Default is *undef*.

$$ $PID $PROCESS\_ID

The process number of the perl running this script. Can be set (but doing so is discouraged)

$0 $PROGRAM\_NAME

Contains the name of the program being executed

$( $GID $REAL\_GROUP\_ID

The real gid of this process. If membership of multiple groups is supported, is a space separated list.

$) $EGID $EFFECTIVE\_GROUP\_ID

The effective gid of this process. If membership of multiple groups is supported, is a space separated list.

$< $UID $REAL\_USER\_ID

The real uid of this process

$> $EUID $EFFECTIVE\_USER\_ID

The effective uid of this process

$; $SUBSEP $SUBSCRIPT\_SEPERATOR

The subscript separator for multidimensional array emulation

$a $b

Special variables used by *sort()*

$] $OLD\_PERL\_VERSION

Version of perl interpreter, represented as a decimal *5.XXXYYY*, where *XXX* and *YYY* are the version/subversion

$^F $SYSTEM\_FD\_MAX

The maximum (currently used) system file descriptor (typically 2)

$^I $INPLACE\_EDIT

Current value of the inplace-edit ‘-i’ option. Set to *undef* to disable inplace editing.

$^M

Emergency memory pool

$^O $OSNAME

The name of the OS under which this copy of perl was built

$^T $BASETIME

The time (in seconds since epoch) at which program began running

$^V $PERL\_VERSION

The revision, version, and subversion of the perl interpreter, as a version object (use *$]* for decimal representation)

$^X $EXECUTABLE\_NAME

The (relative or absolute) path of the perl executable, or alternatively the string used to invoke perl

$| $OUTPUT\_AUTOFLUSH

If nonzero, forces a flush after every write or print for the currently selected output channel. Default is 0 (note that a value of zero doesn’t necessarily mean output channel is buffered). Does not effect input buffering

${^LAST\_FH}

Reference to the last read filehandle. Set by *<HANDLE>*, *readline()*, *tell()*, *eof()*, and *seek()*.

## global arrays

@F

Contains the fields of each line when autosplit ‘-a’ is enabled

@INC

List of places that *do*, *require*, and *use* operators look for their library files. Consists of any arguments given with -I, followed by default perl library.

@ARGV

Command line arguments intended for script. Note that *$ARGV[0]* is the first argument, not the program name.

## global hashes

%ENV

Contains current shell environment. Any changes are inherited by child processes created by *fork()*

%INC

Entries for each filename included via *do*, *require*, or *use* operators, with the filenames as keys, and path of corresponding file as values. For files loaded via a hook, the <name of the?> hook is used as the key instead.

%SIG

Contains signal handlers

## global filehandles

ARGV

Filehandle that iterates over command line arguments in *@ARGV*

ARGVOUT

Filehandle that points to the currently open output file when doing inplace-editing ‘-i’

STDIN

STDOUT

STDERR

Filehandles for standard input/output/error

DATA

Filehandle for anything following *\_\_END\_\_* or *\_\_DATA\_\_*

\_ (underscore)

Filehandle used to cache the information from the last *stat*, *lstat*, or file test operator

## global constants

\_\_END\_\_

Indicates logical end of program. Any following text is ignored, but readable via DATA filehandle.

\_\_FILE\_\_

Represent the filename at the point in program where it is used. Not interpolated into strings.

\_\_LINE\_\_

Represents the current line number. Not interpolated into strings.

\_\_PACKAGE\_\_

Represents the current package name at compile time, or undefined if there is no current package. Not interpolated into strings.

## format variables

$^A $ACCUMULATOR

Current value of the *write()* accumulator for *format()* lines

$^L $FORMAT\_FORMFEED

What formats output as a form feed. Default is ‘\f’.

$% $FORMAT\_PAGE\_NUMBER

The current page number of currently selected output channel

$- $FORMAT\_LINES\_LEFT

Number of lines left of the page of the currently selected output channel

$: $FORMAT\_LINE\_BREAK\_CHARACTERS

The set of characters after which a string may be broken to fill continuation fields (starting with *^*) in a format. Default is *‘ \n-‘* (space, newline, hyphen).

$= $FORMAT\_LINES\_PER\_PAGE

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

$^ $FORMAT\_TOP\_NAME

The name of the current top-of-page format for the currently selected output channel. Default is the name of the filehandle with *‘\_TOP’* appended.

$~ $FORMAT\_NAME

The name of the current report format for the currently selected output channel. Default is the name of the filehandle.

## error variables

$! $ERRNO $OS\_ERROR

Current value of the C *errno* integer value. When referenced as a string, yields the message corresponding to *errno*.

%! %ERRNO %OS\_ERROR

Each element of *%!* Is true only if *$!* is set to that value

$? $CHILD\_ERROR

The status returned by the last pipe close, backtick command, successful call to *wait()* or *waitpid()*, or from *system()*. The exit value of the subprocess is really (*$? >> 8*), and *$? & 127* gives which signal, if any, the process died from, and *$? & 128* reports whether there was a core dump.

$@ $EVAL\_ERROR

The perl error from the last *eval* operator, i.e: the last exception that was caught.

${^CHILD\_ERROR\_NATIVE}

The native status returned by the last pipe close, backtick command, successful call to *wait()* or *waitpid()*, or from *system()*.

$^E $EXTENDED\_OS\_ERROR

Error information specific to the current OS

$^S $EXCEPTIONS\_BEING\_CAUGHT

Current state of the interpreter

$^W $WARNING

The current value of the warning switch, true if ‘-w’ was used, modifiable

${^WARNING\_BITS}

The current set of checks enabled by the *use warnings* pragma

## interpreter state variables

$^C $COMPILING

Current value of flag associated with ‘-c’

$^D $DEBUGGING

Current value of debugging flags ‘-D’

${^GLOBAL\_PHASE}

Current pause of the perl interpreter. Possible values are: CONSTRUCT, START, CHECK, INIT, RUN, END, DESTRUCT.

$^H

For internal use. Contains compile-time hints for the perl interpreter.

%^H

Provides the same scoping semantics as *$^H* – can be used for implementation of lexically scoped pragmas

${^OPEN}

For internal use. String, describing input and output layers, separated by *\0* byte.

$^P $PERLDB

For internal use. Variable for debugging support.

${^TAINT}

One if taint mode ‘-T’ is on, zero if it off, minus-one for only taint warnings ‘-t’

${^SAFE\_LOCALES}

One if safe locale operations are available, zero otherwise

${^UNICODE}

Perl Unicode settings ‘-C’

${^UTF8CACHE}

State of internal UTF-8 offset caching code. One if on (default), zero if off, minus-one for debugging.

${^UTF8LOCALE}

Whether UTF-8 locale was detected by perl at startup ‘-CL’

## deprecated/removed variables

$#

Format printed numbers Deprecated in Perl 5, removed in 5.10.

$\*

Enable multiline matching. Deprecated in Perl 5, removed in 5.10.

$[

What is the index of the first element of an array (0 for zero indexing, 1 for one indexing. Deprecated in perl 5.12, has no effect as of 5.30.

# input/output

console output

standard input/output

# error handling

# strings

## string functions

chomp()

length()

substr()

uc()

ucfirst()

lc()

lcfirst()

chr()

chop()

index()

rindex()

sprint()

ord()

quotemeta()

split()

# arrays

## syntax

Length – array returns its length when used in a scalar context

Is Empty:

if (!@var\_array)

if (@var\_array == 0)

if (scalar @var\_array == 0)

## array functions

push(*Array, List*)

Pushes the values in *List* onto the end of *Array*. Returns the number of elements in resulting array.

pop(*Array*)

Remove the last element of *Array* and return it, or return *undef* if *Array* is empty.

shift(*Array*)

Remove the first element of *Array* and return it, or return *undef* if *Array* is empty. Remaining elements are all shifted left.

unshift(*Array, List*)

Places the values of *List* onto the beginning of *Array*, shifting existing values right. Returns number of elements in resulting array.

sort(*List*)

sort(*Subroutine, List*)

sort(*Block, List*)

Sort *List* and return result. Sorting order may be specified by *Subroutine*, or *Block.* Behaviour is undefined in scalar context. *Subroutine* must be a subroutine (or a reference to a subroutine) that returns greater-than zero, less-than zero, or zero, given two arguments, according to how those arguments are to be ordered.

Examples of *Block*:

{ $a cmp $b } # alphabetical

{ $a <=> $b } # numeric

wantarray()

Returns *true* if called within function that was called in list context. Returns *false* if called within function that was called in scalar context. Returns *undef* if called within function that was called in void context.

exists(*Expr*)

If *Expr* specifies an element of a hash, return *true* if the specified element has ever been initialized (even if corresponding value if undefined). Use of *exists()* on list elements is (strongly) discouraged due to often surprising behaviour (use *defined()* instead).

defined(*Var*)

Returns *true* if variable *Var* has value other than *undef*.

grep(*Expression, @Array*)

Extract any elements from the given *Array* which evaluate *true* given the regex *Expression*. Returns list of results in list context, or number of results in scalar context.

split(*Pattern*)

split(*Pattern, Expr*)

split(*Pattern, Expr, Limit*)

Split string *Expr* (use *$\_* if not given) into a list of strings, using *Pattern*. If *Pattern* is an empty string, *Expr* is split between characters. Returns resulting list in list context, or size of this list in scalar context. If *Limit* is given, it specifies the maximum number of fields into which *Expr* may be split (which is 1 greater than the number of splits that will be made). If *Limit* is negative, as many fields as possible will be produced. If it is unspecified, or zero, trailing empty fields are stripped. When assigning result of *split()* to list of variables, *Limit* is implicitly 1 greater than the number of variables in said list.

join(*Delim, List*)

Join the separate strings of *List* into a single string, with fields separated by *Delim*, and return it.

reverse(*List*)

Reverses values in *List*. Returns reversed list in list context, or in scalar context, concatenated string of reversed list with characters of each element also reversed.

# hashes

## syntax

Initialising:

%var\_hash = ('Welcome' => 10, 'to' => 20, 'Geeks' => 80);

Access element:

print “$var\_hash{‘Welcome’}”

Nested hash

Access element:

print “$var\_hash{‘key\_outer’}{‘key\_inner’}”

Set element:

$var\_hash{‘key\_outer’}{‘key\_inner’} = $value

## printing a hash

use Data::Dumper;

print Dumper(\%hash\_var);

or

print “@{[%hash\_var]}”;

or

while ( my($k, $v) = each(%hash\_var) ) { print “k=($k), v=($v)\n”; }

## hash functions

values()

keys()

each()

delete()

# regex

match operator

m//

substitute operator

s///

transliterate

tr///

## binding operator

By default, these regex operators are matched against *$\_* the default variable. That is:

m/regex/

is equivalent to:

$\_ =~ m/regex/

When a variable other than *$\_* is used, the regex is matched against that instead.

The match operator does not modify the variable being compared. The substitution and transliterate operators however store the operation result in the same variable being compared.

## modifiers

match

i case insensitive

m match ^$ against newlines instead of against string boundry

o evaluate expression only once

s allows '.' to match newline

x Allows whitespace in the expression for clarity

g globally find all matches

cg allow search to contine even after a global match fails

a use only ASCII versions of character classes

c don't reset pos on failed matches when using /g

substitute

i case insensitive

m match ^$ against newlines instead of against string boundry

o evaluate expression only once

s allows '.' to match newline

x Allows whitespace in the expression for clarity

g Replace all occurences of the found expression

e Evaluate the replacement as a perl statement, and use the return value as replacement text

r Leave origional string alone and return modified copy

transliterate

c Complements SEARCHLIST

d Delete found-but-unreplaced characters

s Squashes duplicate replaced characters

## capture groups

Named capture group

(?<name>REGEX)

(?P<name>REGEX)

Backreference to group (use name *n* where *n* is the number of the group for unnamed groups, or just *\n*)

\g{name}

\k<name>

(?P=name)

## all matches as list

Result of match operator, with *g* (match all) modifier, when evaluated in list context.

## regex

[...] any single character in ...

[^...] any single character not in ...

\* 0 or more occurences

+ 1 or more occurences

? 0 or 1 occurences

{n} exactly n occurences

{n,} n or more occurences

{,m} at most m occurences

{n,m} n to m occurences

a|b a or b

\w word character

\W non-word character

\s whitespace [\t\n\r\f]

\S non-whitespace

\d digit [0-9]

\D non-digit

^ or \A match beginning of string

$ or \Z match end of string (before newline)

\z end of string

\b{} match at Unicode boundary of specified type

\B{} Match where corresponding \b{} doesn’t match

\b word bountry when outside brackets, backspace (0x08) inside brackets

\B non word boundry

\G match only after ‘*pos*’ position of last match

\n newline

\t tab

\1 ... \9 n-th grouped subexpression

\10 10th grouped subexpression if matched, otherwise octal representation of char code

Non-greedy quanitifiers

?? 0 or 1

\*? 0 or more

+? 1 or more

{m,n}? specific number

## regex variables

$<digit>

Subpattern from the n-th set of capturing parentheses from the last successful pattern match

@{^CAPTURE}

Contents of the capture buffers (if any) of the last successful pattern match. 0-th element is equivalent to $1, ect.

$& $MATCH

String matched by the last successful pattern match (excluding matches hidden within a BLOCK or *eval()*)

${^MATCH}

Same as $MATCH, without associated performance penalty

$` $PREMATCH

String preceding whatever was matched by the last successful pattern match (excluding matches hidden within a BLOCK or *eval()*)

${^PREMATCH}

Same as $PREMATCH, without associated performance penalty

$’ $POSTMATCH

String following whatever was matched by the last successful pattern match (excluding matches hidden within a BLOCK or *eval()*)

${^POSTMATCH}

Same as $POSTMATCH, without associated performance penalty

$+ $LAST\_PAREN\_MATCH

The text matched by the highest used capture group of the last successful search pattern, equal to the highest numbered captured variable with a defined value

$^N $LAST\_SUBMATCH\_RESULT

The text matched by the used group most recently closed (the group with the rightmost closing parenthesis) of the last successful search pattern

$#+

Number of subgroups in last successful match

@+ @LAST\_MATCH\_END

Offsets of the ends of the last successful submatches in the currently active dynamic scope. $+[0] is the offset of the end of the entire match (what *pos()* returns when called on variable that was matched against), $+[1] is the offset past where $1 ends, ect.

@- @LAST\_MATCH\_START

As per @LAST\_MATCH\_END, but with the offsets of the beginnings of the last successful submatches

%+ ${^CAPTURE} $LAST\_PAREN\_MATCH

Named capture groups in the last successful match in the currently active dynamic scope. Keys list only the names of groups that have been captured, value is text matched by capture group. If multiple capture groups have the same name, the leftmost group is kept.

%- %{^CAPTURE\_ALL}

Named capture groups in the last successful match in the currently active dynamic scope. Keys are the names of the capture groups (whether they were found or not), and corresponding values are a list of all matches from groups of said name.

$^R $LAST\_REGEXP\_CODE\_RESULT

The result of evaluation of the last successful *(?{code})* regex assertion.

${^RE\_COMPILE\_RECURSION\_LIMIT}

The maximum number of open but unclosed parenthetical groups there may be at any point in a regex. Default is 1000.

${^RE\_DEBUG\_FLAGS}

Regex debugging flags

${^RE\_TRIE\_MAXBUF}

Controls memory usage of regex optimisations. Default is 65536 which corresponds to 512kB. Higher value trades memory for speed, negative value conserves as much memory as possible.

# files

## file functions

open()

glob()

tell()

getc()

reverse()

rename()

print()

# maths

## math functions

exp()

hex()

srand()

sqrt()

oct()

rand()

log()

int()

sin()

cos()

atan2()

abs()

# packages

@ISA

Array containing list of the package (class) parent classes (if any)

# Module Data

## Data::Dumper

# Features by version

v5.10

State (persistent) variables

Named capture groups