Perl 5 🚧

See also: 𝔓𝔩 - Perl • **Perl @ Wikipedia** • **perl.org**	**Perl Guidelines** **Tools:**	**Perl Style Guide, 10 Essential Development Practices,** • Books: **Perl Best Practices**, **Modern Perl Best Practices (course)** • **perlcritic script** uses **Perl::Critic** to scan Perl code. • The **perltidy** application reformats Perl code. **Older perltidy home page**. **PerlTidy @ Wikipedia**, **PBP recommended .perltidyrc**		
	Learning Perl	• Perl Intro - a quick introduction to Perl • Effective Perl Programming • Online Perl books • Beginning Perl • Modern Perl (html) • Perl Maven Tutorial	• perl , Perl command line options • perlivp , perldoc , perlbug / perlthanks **perlsec - Perl security**	• Online Perl Interpreter • Online PerlTidy ☝option info.
perldoc browser • `C-c C-h F`	**Topic groups:**	• **perldoc** : about perldoc itself • **perltoc** : table of content: names of all pages • **perlsyn** : Perl syntax • **perlfunc** : Perl built-in functions		
CPAN	• **CPAN @ Wikipedia** • **The Zen of Comprehensive Archive Networks** • **CPAN** • **Search CPAN — meta::cpan** • **PAUSE - Perl Authors Upload Server**	**Command line tools** interacting with CPAN: • **cpan** : install on some Linux with: `sudo dnf install perl-CPAN` • **cpanplus** • cpanminus : **cpanm** : install on some Linux with: `sudo dnf install perl-App-cpanminus`		

Perl scripts

Writing Perl scripts

Use the following at the beginning of Perl script files.	`#!/usr/bin/perl`
	`use strict;`
	`use warnings;`
	`use diagnostics;`
		• The first line of an executable script should be a valid shebang line identifying the appropriate location of the Perl interpreter.
		• Most Perl code should also activate the strict Perl rules and warnings to detect warnings.
		  • See: Barewords in Perl
		• If you want to produce more diagnostics for detected warning or errors then add the '`use diagnostics;`' line.

use version/features	`use v5.36;`	This can be used to enable both the strict and warning pramas as well as several named features. • See the **table listing the feature bundles per Perl versions**.

Perl 5 Operators

Perl 5 Operators

Note:

Perl has a large number of operators, listed below with their **precedence and associativity**.
- C Operators missing from Perl : unary &, unary * and (type)
- Quote and Quote-like operators : in Perl quotes are operators and they provide various kind of interpolating and pattern matching capabilities.

Associativity: one of:
- right
- left
- NA : not associative: cannot use more than one of these operators in sequence.
- CH: chained

To get this information, use:
`perldoc perlop`

left	**terms and list operators (leftward)**	
left	**Arrow Operator:**	`->`
NA	**Auto-increment and Auto-decrement:**	`++ --`
right	**Exponentiation:**	`**`
right	**Symbolic Unary Operators:**	`!  ~  ~.  \` and unary `+` and `-` **Note:** The operator `\` creates a reference. See example.
left	**Binding operators:**	`=~ !~`
left	**Multiplicative Operators:**	`*  /  %  x`
left	**Additive Operators:**	`+  -  .`
left	**Shift Operators:**	`<<  >>`
NA	**named unary operators**	
NA	**Class instance Operator:**	`isa`
CH	**Relational Operators:**	as numbers: `<  >  <=  >=` as strings: `lt  gt  le  ge`
CH/NA	**Equality Operators:**	as numbers: `==  !=  <=>` as strings: `eq  ne  cmp  ~~`
left.	**Bitwise And:**	`&  &.`
left	**Bitwise Or and Exclusive Or:**	`|  |.  ^  ^.`
left	**C-style Logical And:**	`&&`
left	**Logical Defined-Or:**	`||  ^^  //`
NA	**Range Operators:**	`..  …`
right	**Conditional Operator:**	`?:`

right	**Assignment Operators:**	`=`
		`**=  +=  *=  &=  &.=  <<=  &&=`
		`-=  /=  |=  |.=  >>=  ||=`
		`.=  %=  ^=  ^.=  //=`
		`x=`
		`goto last next redo dump`

left	**Comma, fat-comma Operators:**	`,  =>`
NA	**list operators (rightward)**	
right	**Logical Not:**	`not`
left	**Logical And:**	`and`
left	**Logical or and Exclusive or:**	`or xor`

trick operators
These are not real Perl operators, but look like operators: they are concatenation of other operators that achieve a specific effect.
See the link for others.
Understanding these operators helps understand Perl. They should not be used in production code.

`-+-`
`0+`
	Converts a string that starts with digits into a number.	`print -+- '22les poulets!';`
		# prints 22
			-+- is essentially - + - or - - but a + to allow placing them together. The `0+` does the same as `-+-`, but the second has higher precedence.

`=()=`	Called the '**goatse**' operator. It causes the right side expression to be evaluated in array context. Used to assign the array/list size to a scalar.	`my $str = "A 22 before 33 does not make 9, it is 44!";`
		`my $digit_count =()= $str =~ /\d/g;`
		`print "$digit_count";` # prints '7',the number of digits in $str	

`@{[]}`	Useful to interpolate an array inside a string. Note that: `"@{[something]}"` is the same as `join $", something`	`print "these people @{[get_names()]} get promoted"`	

`~~`	Force scalar context.	In scalar context localtime() returns human readable time, but in list context it returns a 9-tuple with various date elements.	`$ perl -le 'print ~~localtime'`
			`Mon Nov 30 09:06:13 2009`

Truth and falsehood ⚠️ Remember that the strings '0' and '' mean false. The output of glob() may return a file named '0' !	• False in a **boolean context**: • the number **0**, • the strings **'0'** and **' '**, • the empty list **()**, • **"undef"** • All other values are true.	• Negation of a true value by "!" or "not" returns a special false value. • When evaluated as a string it is treated as '', but as a number, it is treated as 0.	So the following scalar values are considered false: • undef - the undefined value • 0 the number 0, even if you write it as 000 or 0.0 • '' the empty string. • '0' the string that contains a single 0 digit.	All other scalar values, including the following are true: • 1 any non-0 number • ' ' the string with a space in it • '00' two or more 0 characters in a string • "0\n" a 0 followed by a newline • 'true' • 'false' yes, even the string 'false' evaluates to true.

File test operators

It is possible to combine the file test operator with the AND operator as in the following example:

```
if (-e $fname && -f _ && -r _ ){
   print("$fname exists and is readable\n");
}
```

The most important operators are shown here.
They check if the file…

-r	is readable	**-e**	exists.	**-b**	is a block special file.
-w	is writable	**-z**	is empty.	**-c**	is a character special file.
-x	is executable	**-s**	has nonzero size (returns size in bytes).	**-t**	handle is opened to a tty.
-o	is owned by effective uid.	**-f**	is a plain file.	**-u**	has setuid bit set.
-R	is readable	**-d**	is a directory.	**-g**	has setgid bit set.
-W	is writable	**-l**	is a symbolic link.	**-k**	has sticky bit set.
-X	is executable	**-p**	is a named pipe (FIFO) or Filehandle is a pipe.	**-T**	is an ASCII text file (heuristic guess).
-O	file is owned by real uid.	**-S**	is a socket.	**-B**	is a "binary" file (opposite of -T).

Perl 5 Constants and Variables

Perl Sigils	Sigil	Examples	Meaning	Extra Info
Scalar	$	`$foo`	Simple scalar value	
		`$days[28]`	29th element of array @days	
		`$days{'Feb'}`	Value associated with the *Feb* key of hash %days	
		`${days}`	Same as `$days`, but unambiguous before alphanumerics. Useful inside strings for interpolation of variables followed by other letters.	
		`$Dog::days`	The `$days` variable inside the Dog package.	
		`$Dog'days`	Same as above. However this is an archaic use of the single quote.	
		`$#days`	Last index of array `@days`.	
		`$days->[28]`	29th element of array pointed to by reference `$days`.	
		`$days[0][2]`	Multi-dimensional array	
		`$d{99}{'Feb'}`	Multi-dimensional hash	
		`$d{99, 'Feb'}`	Multi-dimensional hash emulation	
Array	@	`@days`	Array containing `($days[0], $days[1], … #days[$#days])`.	
		`@days[3,4,5]`	Array slice containing `($days[3], $days[4], $days[5])`.	
		`@days[3..5]`	Array slice containing `($days[3], $days[4], $days[5])`.	
		`@days{'J','F'}`	Hash slice containing `($days{'J'}, $days{'F'})`.	
Hash/associative array	%	`%days`	Associative array (hash): keys-value pairs. Can be initialized as: • `%days = (Jan => 31, Feb => $leap? 29 : 28, …)` • `%days = ("Jan, 31, 'Feb', $leap? 29 : 28, … )`	
Subroutine	&	`&foo`	& is needed to create reference to subroutine.	
Typeglob	*	`*foo`		See: Advanced Perl Programming, 1st Edition Section 3.2

7 kinds of package variables or variable-like elements in Perl:			
	1. **scalar variables** 2. **array variables** 3. **hash variables**	4. **subroutine name** 5. **format names** • how to format output in Perl?, Perl-Formats • See **write** and **select**	6. **file handles** 7. **directory handles**

Scalar values		Numeric literals examples	Useful related builtin functions

• **numeric**:	• integer : using the system's native format.
	  • **bigint** - transparent big integer support.
	  • **bignum** - transparent big number support.
	• floating-point : using the system's native format.
	  • **bigrat** - transparent big rational number support.
		`my $x = 12345;          # integer`
		`my $x = 12345.67;       # floating point`
		`my $x = 6.02e23;        # scientific notation`
		`my $x = 4_294_967_296;  # underline for legibility`
		`my $x = 0377;           # octal`
		`my $x = 0xffff;         # hexadecimal`
		`my $x = 0b1100_0010;    # binary`
			• **oct**
			• **hex**
			• **POSIX::ceil**
			• **POSIX::floor**
			• **abs**

• **string**

• double-quoted strings: perform backslash and variable interpolation of expression that begin with **$** (a scalar) or **@** (an array). Hashes cannot be interpolated.
• single-quote strings: only perform `\'` and `\\` substitution (to `'` and `\` respectively), nothing else.

• Quote constructs	Customary	Generic	Meaning	Interpolates?	Notes

See:
• Strings in Perl: quoted, interpolated and escaped
	`''`	`q//`	Literal string	No	• Not all characters can be used as the / separator. `{ }`, `( )` and `< >` can also be used.
					• You can use whitespace between the quote specifier and its initial bracketing character:
					`my $chuck_of_code = q {`
					`    if ($condition) {`
					`       print "Salut!";`
					`    }`
					`};`
					• It's also possible to write: `s<foo>(bar)` and `tr(a-f)[A-F]` as well as:
					`tr (a-f)`
					`   [A-F];`
					• Array variables are interpolated by joining all elements with the separator specified by the $" special variable ($LIST_SEPARATOR) .

	`""`	`qq//`	Literal string	Yes	
	` `` `	`qx//`	Command execution	Yes	
	`()`	`qw//`	World list	No	
	`//`	`m//`	Pattern match	Yes	
	`s///`	`s///`	Pattern substitution	Yes	
	`tr///`	`y///`	Character translation	No	
	`""`	`qr//`	Regular expression	Yes	

• Character escapes						
	\a	Alert (bell)	**\e**	ESC character	`\N{LATIN SMALL LETTER E WITH ACUTE}`	é
	\b	Backspace	**\033**	ESC in octal	`\N{ U+E9 }`	é
	\e	ESC character	**\o{33}**	ESC in octal		
	\f	Form feed	**\x7f**	DEL in hexadecimal		
	\n	Newline (usually LF)	**\x{263a}**	Character number 0x263A		
	\r	Carriage return (Usually CR)	**\cC**	Control-C		
	\t	Horizontal tab				

• translation escapes						
	`\u`	Force next character to titlecase	**\U**	Force all following characters to uppercase. Ends at **\E**	**\E**	Ends **\U**, **\L**, **\F** or **\Q**
	`\l`	Force next character to lowercase	**\L**	Force all following characters to lowercase. Ends at **\E**		
			\F	Force all following characters to fold case. Ends at **\E**		
			\Q	Backslash all following non alphanumeric characters. Ends at **\E**		

• **bareword**

In Perl, a *bareword* refers to a sequence of characters suitable for an identifier. It's not quoted. By default Perl allows barewords to behave like strings.
• This is not allowed when any of `use strict;` or `use strict "subs";` or `use v5.12;` is specified.

• **Here documents**
• Here docs @ Perl maven
• Perl here doc @Wikipedia

Perl here-documents are a form of line oriented quoting. There are several forms of here documents, where the identifier (like **EOF** used below, but can be any word) must be placed at the beginning of the terminating line:

• Default :	**<<EOF;**	Supports variable interpolation.
• Double quotes:	**<<"EOF";**	Supports variable interpolation. Can also be written with whitespace as in **<< "EOF";**
• Single quotes:	**<<'EOF';**	Does not support interpolation. Can also be written with whitespace as in **<< 'EOF';**
• backticks:	**<<\`EOF\`;**	Execute commands in a shell and return text printed on stdout. Can also be written with whitespace as in **<< \`EOF\`;**
• indented:	**<<~EOF;**	Allows indenting the here-doc string. Can also use the ~ with the other forms: **<<~\EOF**, **<<~"EOF"**, **<<~"EOF"**, **<<~\`EOF\`**

• They can also be stacked and text can be transformed. See the documentation.

• **Perl Regexp** info, cheatsheets & regexp testers			
	• **Regexp Tutorial** • **Learn PCRE in X minutes**	• **PCRE cheatsheet**	• Debuggex regexp tester • regex101 • RegEx Pal

Perl Constants

• Perl pragma to declare constants. ⚠️ But be aware that these are still not read-only, that they inject sub-routines and have several limitations. Read the doc!!
• CPAN modules for defining constants by Neil Bowers . Of particular interest: Const::Fast and Attribute::Constant for efficient read-only constants.

Perl Variables Names	Scalar Naming Conventions		Array Naming Conventions
☝ Case is significant in all names.	• Local variables: • Global variables: • Constants: • All variables:	$lowercase $Title_Case $UPPER_CASE words separated by underscores.	Similar conventions, except that array names should be **plural**. • @locals • @Global_Arrays • @CONSTANT_ARRAYS

Perl Special Variables
• **Perl Variables**

☝ To get information about a Perl special variable from the command line use the **perldoc -v** command.
• To get information about **$<** use: `perldoc -v '$<'`

• General variables			
default input and pattern searching space	• $ARG • $_	subroutine parameters	• @ARG • @_
list separator	• $LIST_SEPARATOR • $"	Subscript separator for multidimensional array emulation	• $SUBSCRIPT_SEPARATOR • $SUBSEP • $;
Name of executed program	• $PROGRAM_NAME • $0	Name used to execute the current copy of Perl	• $EXECUTABLE_NAME • $^X
Perl process ID	• $PROCESS_ID • $PID • $$		

Process real GID	• $REAL_GROUP_ID • $GID • $(	Process effective GID	• $EFFECTIVE_GROUP_ID • $EGID • $)		
Process real UID	• $REAL_USER_ID • $UIG • $<	Process effective UID	• $EFFECTIVE_USER_ID$ • $EUID • $>		
Special variables in sort	• $a • $b	Example: by default Perl sort function sorts strings. Pass a sorting function that uses the **<=>** equality operator to force numerical comparisons: `@sorted = sort { $a <=> $b } @unsorted;`			
Current environment	%ENV	Environment variable accessed as an associative array (a hash). • See: Perl: How to access shell environment variables through Perl associative arrays.			
Perl interpreter revision, version and subversion	• $OLD_PERL_VERSION • $]	Perl interpreter revision, version and subversion	• $PERL_VERSION • $^V		
Maximum file descriptor	• $SYSTEM_FD_MAX • $^F				
Fields of each line when auto-split mode is on.	@F				
Include Directories	@INC	Included filenames	%INC	Hook localization (?)	$INC
inplace-edit extension value	• $INPLACE_EDIT • $^I				
Package's class parent classes	@ISA				
Emergency memory pool	$^M				
Maximum block nesting	${^MAX_NESTED_EVAL_BEGIN_BLOCKS}				
Name of OS where this Perl was built	• $OSNAME • $^O				
Signal handlers	%SIG				
Coderefs for various perl keywords	%{^HOOK}				
Time when program began running	• $BASETIME • $^T				
• **Variables related to regular expressions**					
captured sub-patterns	$<digit>($1, $2, …)				
Capture buffer content	@{^CAPTURE}				
String matched	• $MATCH • $&	String matched (compiled regexp)	${^MATCH}		
String preceding match	• $PREMATCH • $`	String preceding match (compiled regexp)	${^PREMATCH}		
String following match	• $POSTMATCH • $'	String following match (compiled regexp)	{^POSTMATCH}		
Last capture group	• $LAST_PAREN_MATCH • $+	Most recently closed capture group	• $LAST_SUBMATCH_RESULT • $^N		
Match capture key values	• %{^CAPTURE} • %LAST_PAREN_MATCH • %+				
Match start offsets	• @LAST_MATCH_START • @-	Match ends offsets	• @LAST_MATCH_END • @+	Named captured groups	• %{^CAPTURE_ALL} • %-
Last successful pattern	${^LAST_SUCESSFUL_PATTERN}				
Result of last successful regexp assertion	• $LAST_REGEXP_CODE_RESULT • $^R				
Maximum regexp nested group	${^RE_COMPILE_RECURSION_LIMIT}				
regexp debug flag	${^RE_DEBUG_FLAG}				
regexp internal optimization/memory	${^RE_TRIE_MAXBUF}				
• **Variables related to file handles**	See also: **Perl File Handles**				
Name of current file read from <>	$ARGV	Command line arguments of the script	@ARGV	Number of arguments minus one	$#ARGV
Special file handle that iterates over command-line filenames in @ARGV	ARGV	Special file handle that points to currently open output file when doing edit-in-place processing	ARGVOUT		
Output field separator for the print operator	• IO::Handle->output_field_separator(EXPR) • $OUTPUT_FIELD_SEPARATOR • $OFS • $,	Current line number for the last file handled accessed	• HANDLE->input_line_number(EXPR) • $INPUT_LINE_NUMBER • $NR • $.		
Input record separator (newline by default)	• IO::Handle->input_record_separator(EXPR) • $INPUT_RECORD_SEPARATOR • $RS • $/	Output record separator	• IO::Handle->output_record_separator(EXPR) • $OUTPUT_RECORD_SEPARATOR • $ORS • $\		
Auto-flush control • See: order of output @ Perl Maven	• HANDLE->autoflush(EXPR) • $OUTPUT_AUTOFLUSH • $|	Last read file handle	${^LAST_FH}		
• **Variables related to format**					
Current value of the write() accumulator for format() lines.	• $ACCUMULATOR • $^A				

Form feed format. defaults to \f	• IO::Handle->format_formfeed(EXPR) • $FORMAT_FORMFEED • $^L	Set of characters after which a string may be broken to fill continuation fields	• IO::Handle->format_line_break_characters EXPR • $FORMAT_LINE_BREAK_CHARACTERS • $:
Number of lines left on the page on currently selected output channel	• HANDLE->format_lines_left(EXPR) • $FORMAT_LINES_LEFT • $-	Current page length of current output channel	• HANDLE->format_lines_per_page(EXPR) • $FORMAT_LINES_PER_PAGE • $=
Name of current top-page format of output channel	• HANDLE->format_top_name(EXPR) • $FORMAT_TOP_NAME • $^	Report format name of output channel	• HANDLE->format_name(EXPR) • $FORMAT_NAME • $~
• **Error Variables**	The variables $@, $!, $^E, and $? contain information about different types of error conditions that may appear during execution of a Perl program. They correspond to errors detected by the Perl interpreter, C library, operating system, or an external program, respectively.		
Perl error from the last eval operator	• $EVAL_ERROR • $@	Current state of interpreter	• $EXCEPTIONS_BEING_CAUGHT • $^S
Current value of C errno integer variable	• $OS_ERROR • $ERRNO • $! **$!** returns the system variable **errno** when used in a numeric context, but returns the string from **perror()** when used in string context.	Hash of error names to 0 or 1, set to 1 if current error is this error.	• %OS_ERROR • %ERRNO • %!
OS detected error	• $EXTENDED_OS_ERROR • $^E		
Status returned by last pipe close, backtick command, wait, waited, or system() call.	• $CHILD_ERROR • $?	native status returned by last pipe close , backtick command, wait() or waitpid() or system() call	${^CHILD_ERROR_NATIVE}
Current value of warning switch	• $WARNING • $^W	Current set of warning checks enabled by the use warnings pragma	${^WARNING_BITS}
• **Variables related to the interpreter state**	These variables provide information about the current interpreter state.		
Flag associated with the -c switch	• $COMPILING • $^C	The current value of the debugging flags	• $DEBUGGING • $^D
Current phase of the perl interpreter	${^GLOBAL_PHASE}		
Compile-time hints for the perl interpreter. Internal use only	$^H	Values of compiled statements	%^H
Input/Output Layers. Internal use by PerlIO only.	${^OPEN}		
Debugging support. Internal variable.	• $PERLDB • $^P		
Taint mode	${^TAINT}	Safe locale operations availability	${^SAFE_LOCALES}
Unicode Settings of Perl	${^UNICODE}		
Internal UTF-8 offset caching code state	${^UTF8CACHE}	State of UTF-8 locale detected by perl at startup.	${^UTF8LOCALE}
• **Deprecated and removed variables:**	$# $* $[${^ENCODING} ${^WIN32_SLOPPY_STAT}		

Perl 5 open file 🚧

References	• open @ perldoc browser • Writing to files with Perl @ Perl Maven • open file in-memory @ stackOverflow	• Stupid open() tricks @Perl.com: • No explicit filename • create an anonymous temporary file • print to a string • read lines from a string

Perl 5 Statements 🚧

Conditional statements			
Loop control	• while (condition) { … } • until (condition) { … }	loop control keywords: • **`next`** : starts the next iteration of the loop. • **`last`** : exits the loop. • **`redo`** : restarts the loop block without evaluating the condition again.	loop control keywords: • **`continue`** block: executed before evaluating condition again.

Perl 5 Functions 🚧

Perl Functions Perl syntax	☝To get information about a Perl function from the command line use the **perldoc -f** command. • To get information about print use: `perldoc -f print`
⚠️**Cautionary notes**	
• **each** keyword is broken • Use **Var::Pairs** instead.	Do NOT use the built-in **each**. It is broken, as described by Damian Conway in his Modern Perl Best Practice O'Reilly course, section control structure. • **each** is not re-entrant: • nested loops of each over the same hash does not work as expected and will create infinite loop since the nested loop each juts iterates from where the first loop each left it. • Exiting the loop leaves the state of the each internal pointer at the current location. • If you use each on the same hash later it will resume from where it left, it will not start form the beginning.

print functions	• print • say `use feature qw(say);` or `use v5.10;` (or higher). Like print, but implicitly appends a newline at the end of the list.

PerlTidy formatting control 🚧

perltidy option	Option	Impact
indentation style	• **-bl**, • **--opening-brace-on-new-line** • **--brace-left**	• Without this option (the default) the code indentation style selected is **K&R style**. • With this option, the indentation style is **Allman/BSD style**.