



Perl 5






See also: <ul style="list-style-type: none"> Perl - Perl Perl @ Wikipedia perl.org PerlMonks.org O'Reilly Books Perl mailing lists 	<ul style="list-style-type: none"> Perl Intro - a quick introduction to Perl. PerlCheat , Learn Perl in Y minutes, or in 2 hours 30 minutes Online Perl books and <i>tutorials</i> : Beginning Perl , Modern Perl (html) , Perl Maven Tutorial, Intro to Perl-old Perl Cookbook ♫ (PLEAC Perl: <i>list of Perl code solutions</i>) Learning Perl LP♫, Intermediate Perl ♫ , Mastering Perl ♫ , Effective Perl Programming ♫ Other exist but are not recommended for various reasons.	perl , Perl command line options , perlrun , perlivp , perldoc , perlbug / perlthanks perlsec	<ul style="list-style-type: none"> Online Perl Interpreter perl-live-coding out/in Emacs Online PerlTidy option info.
Perl Guidelines and tools	Perl Style Guide, 10 Essential Development Practices. <ul style="list-style-type: none"> Books: Perl Best Practices ♫, Modern Perl Best Practices (course) ♫ perlcritic script uses Perl::Critic to scan Perl code. The pel-perl-critic command invokes it to check code in buffer. The perltidy application reformats Perl code. Older perltidy home page. PerlTidy @ Wikipedia, PBP recommended .perltidyrc 		
perldoc browser <ul style="list-style-type: none"> In Emacs: C-c C-h F 	<ul style="list-style-type: none"> perldoc : about perldoc itself perltoc : table of content: of all pages perlsyn : Perl syntax perlfunc : Perl built-in functions 	 Use perldoc to find if a Perl module is installed, as in: perldoc local::lib <ul style="list-style-type: none"> perldoc local::lib prints the documentation of local::lib if it is installed. perl -Mlocal::lib is useful to get modules installed in your home directory ♫ 	
CPAN (@ Wikipedia) <ul style="list-style-type: none"> Search CPAN — meta::cpan 	<ul style="list-style-type: none"> The Zen of Comprehensive Archive Networks PAUSE - Perl Authors Upload Server Installing Local Perl Modules with CPAN 	Command line tools interacting with CPAN to install Perl modules ♫ : (see also this StackOverflow Q/A): <ul style="list-style-type: none"> cpan: (requires config, but has defaults). Use local::lib; cpan will be able to install into your ~/perl5 tree. <ul style="list-style-type: none"> Type cpan to open the cpan shell, then type install The::Module to install packages. cpanplus, or cpanminus : cpanm :(no config required). cpanm: cpanm -S The::Module 	

Last updated on: 2025-02-07

Perl scripts

Writing Perl scripts		Impose strictures in Perl files to prevent errors by adding one of the following use lines. Also see the strictures package .	
Use the following at the beginning of Perl script files. perldiag @ perldoc	<pre>#!/usr/bin/env perl use strict; use warnings; # for testing only: use diagnostics;</pre>	<pre>#!/usr/bin/perl -w use v5.12; # loads strict ... use v5.35; # &loads warnings ! use diagnostics produces more info but increases startup time.</pre> <p>Alternative: perl -Mdiagnostics . Emacs pel-perl-critic command can report diagnostic.</p>	Executable Perl script should have a valid shebang line identifying the appropriate location of the Perl interpreter. It may have to be modified at installation time (OpenGroup/SUS).  It's best to: use warnings ; perl -w generates warning for all Perl code in the program including modules used by the program. Also use the -c option to check syntax. But most Perl code should also activate the strict Perl rules and warnings to detect warnings. See: Barewords in Perl
	use version/features	<pre>use v5.36;</pre> <p>This can be used to enable both the strict and warning pramas as well as several named features.</p> <ul style="list-style-type: none">• See the table listing the feature bundles per Perl versions.	

Perl 5 Operators


Perl 5 Operators	Perl operators, listed below with their precedence and associativity .			C Operators missing from Perl : unary &, unary * and (type)
Note:	<ul style="list-style-type: none"> Quote and Quote-like operators : in Perl quotes are operators and they provide various kind of interpolating and pattern matching capabilities. 			
Associativity: one of: <ul style="list-style-type: none"> right left NA : not associative: cannot use more than one of these operators in sequence. CH: chained To get this information, use: perldoc perlop Note: ♫ The Bitwise String Operators are : <pre>~. &. . ^. &.= .= ^=</pre>	<div> <div>left</div> <div>terms and list operators (leftward)</div> <div>()</div> </div> <div> <div>left</div> <div>Arrow Operator:</div> <div>-></div> </div> <div> <div>NA</div> <div>Auto-increment and Auto-decrement:</div> <div>++ --</div> </div> <div> <div>right</div> <div>Exponentiation:</div> <div>**</div> </div> <div> <div>right</div> <div>Symbolic Unary Operators:</div> <div>! ~ -. \ and unary + and -</div> </div> <div> <div>left</div> <div>Binding operators:</div> <div>== !=</div> </div> <div> <div>left</div> <div>Multiplicative Operators:</div> <div>* / % x</div> </div> <div> <div>left</div> <div>Additive Operators:</div> <div>+ - .</div> </div> <div> <div>left</div> <div>Shift Operators:</div> <div><< >></div> </div> <div> <div>NA</div> <div>named unary operators</div> <div></div> </div> <div> <div>NA</div> <div>Class instance Operator:</div> <div>isa</div> </div> <div> <div>CH</div> <div>Relational Operators:</div> <div>as numbers: < > <= >= as strings: lt gt le ge</div> </div> <div> <div>CH/NA</div> <div>Equality Operators:</div> <div>as numbers: == != <=> as strings: eq ne cmp --</div> </div> <div> <div>left.</div> <div>Bitwise And:</div> <div>& &.</div> </div> <div> <div>left</div> <div>Bitwise Or and Exclusive Or:</div> <div> . ^ ^.</div> </div> <div> <div>left</div> <div>C-style Logical And:</div> <div>&&</div> </div> <div> <div>left</div> <div>Logical Defined-Or:</div> <div> ^^ //</div> </div> <div> <div>NA</div> <div>Range Operators:</div> <div>.. ...</div> </div> <div> <div>right</div> <div>Conditional Operator:</div> <div>? :</div> </div> <div> <div>right</div> <div>Assignment Operators:</div> <div>=</div> <div> <div>**= += *= &= &.= <<= &&=</div> <div>-= /= = .= >>= =</div> <div>.= %= ^= ^.= // =</div> </div> <div>goto last next redo dump</div> </div> <div> <div>left</div> <div>Comma, fat-comma Operators:</div> <div>, =></div> </div> <div> <div>NA</div> <div>list operators (rightward)</div> <div></div> </div> <div> <div>right</div> <div>Logical Not:</div> <div>not</div> </div> <div> <div>left</div> <div>Logical And:</div> <div>and</div> </div> <div> <div>left</div> <div>Logical or and Exclusive or:</div> <div>or xor</div> </div>			Note: The operator \ creates a reference. See example .
trick operators  Do not use in production code! But understanding how these work does help understand Perl. These are not real Perl operators; they are concatenation of other operators that achieve a specific effect.	<div> <div>+-</div> <div>0+</div> </div> <div> <div>=()</div> </div> <div> <div>@{[]}</div> </div> <div> <div>--</div> </div>	Converts a string that starts with digits into a number. 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. Interpolate an array in a string: <pre>"@{[something]}" is the same as: join \$", something</pre> Force scalar context.	In scalar context localtime returns human readable time, but in list context it returns a 9-tuple with date elements.	<pre>print +- '22les poulets!'; # prints 22</pre> <pre>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</pre> <pre>print "these people @{get_names()} get promoted"</pre> <pre>\$ perl -le 'print ~~localtime' Mon Nov 30 09:06:13 2009</pre>
Truth and falsehood  Remember that the strings '0' and " mean false. The output of glob() may return a file named '0' !  a bareword false has a truth value of true!!!!	<ul style="list-style-type: none"> False in a boolean context: <ul style="list-style-type: none"> the number 0, the strings '0' and ' ' , the empty list () , "undef" All other values are true. 	<ul style="list-style-type: none"> 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 : <ul style="list-style-type: none"> undef - the undefined value 0 the number 0, even if you write it as 000 or 0.0 " the empty string. '0', a single 0 in the string. 	All other scalar values are true , such as: <ul style="list-style-type: none"> 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' . Even the string 'false' evaluates to true.
	 One way to define valid true and false <i>constant symbols</i> that can be used in assignments (but see ♫):		<pre>use constant { true => 1, false => 0 };</pre>	
File test operators See filetest -X	File tests can be stacked (~r ~w -e \$fname) or combined as in the following example ♫ :  Notice the underscore in the example: it's the virtual filehandle _ accessing the last stat or lstat result :			<pre>if (-e \$fname && -f _ && -r _) { print("\$fname exists, is readable\n"); }</pre>
The operators check if the file... See also: <ul style="list-style-type: none"> File Tests ♫ File test operators @ perl tutorial See also: <ul style="list-style-type: none"> localtime File::stat IO::Interactive 	<div> <div>-r</div> <div>is readable <i>by effective uid/gid</i></div> </div> <div> <div>-w</div> <div>is writable <i>by effective uid/gid</i></div> </div> <div> <div>-x</div> <div>is executable <i>by effective uid/gid</i></div> </div> <div> <div>-o</div> <div>is owned <i>by effective uid</i></div> </div> <div> <div>-R</div> <div>is readable <i>by real uid/gid</i></div> </div> <div> <div>-W</div> <div>is writable <i>by real uid/gid</i></div> </div> <div> <div>-X</div> <div>is executable <i>by real uid/gid</i></div> </div> <div> <div>-O</div> <div>file is owned <i>by real uid</i>.</div> </div> <div> <div>-M</div> <div>Days between start time and file modification time</div> </div>	<div> <div>-e</div> <div>exists.</div> </div> <div> <div>-z</div> <div>is empty.</div> </div> <div> <div>-s</div> <div>has nonzero size (returns size in bytes).</div> </div> <div> <div>-f</div> <div>is a plain file.</div> </div> <div> <div>-d</div> <div>is a directory.</div> </div> <div> <div>-l</div> <div>is a symbolic link.</div> </div> <div> <div>-p</div> <div>is a named pipe (FIFO) or Filehandle is a pipe.</div> </div> <div> <div>-S</div> <div>is a socket.</div> </div> <div> <div>-A</div> <div>Days between start time and file access time</div> </div>		<div> <div>-b</div> <div>is a block special file.</div> </div> <div> <div>-c</div> <div>is a character special file.</div> </div> <div> <div>-t</div> <div>handle is opened to a tty.</div> </div> <div> <div>-u</div> <div>has setuid bit set.</div> </div> <div> <div>-g</div> <div>has setgid bit set.</div> </div> <div> <div>-k</div> <div>has sticky bit set.</div> </div> <div> <div>-T</div> <div>is an ASCII text file (heuristic guess).</div> </div> <div> <div>-B</div> <div>is a "binary" file (opposite of -T).</div> </div> <div> <div>-C</div> <div>Days between start time and node change time (in Unix).</div> </div>

Perl 5 Constants and Variables

Perl Constants					<ul style="list-style-type: none">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		All: underscore or letter of the first character.			
Case is significant in all names. ASCII by default, UTF-8 if the utf8 pragma is used.		<ul style="list-style-type: none">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 . <ul style="list-style-type: none">@locals@Global_Arrays@CONSTANT_ARRAYS		<ul style="list-style-type: none">Module names are MixedCaseNoUnderscoresConstants are UPPERCASE_WITH_UNDERSCORESPackage wide vars are Mixed_Case_With_UnderscoresFunctions/methods are lowercase_with_underscoresAvoid ALLUPPERCASE: used by Perl special variables.		
Perl types		Sigil	Examples	Meaning			Extra Info			
Scalar		\$	\$foo \$days[28] \$days{'Feb'} \${days} \$Dog::days \$Dog' days \$#days \$days->[28] \$days[0][2] \$d{99}{'Feb'} \$d{99, 'Feb'}	Simple scalar value 29 th element of array @days Value associated with the <i>Feb</i> key of hash %days Same as \$days, but unambiguous before alphanumerics. Useful inside strings <u>for interpolation of variables followed by other letters</u> . The \$days variable inside the Dog package. Same as above. However this is an archaic use of the single quote. Last index of array @days . 29 th element of array pointed to by reference \$days. Multi-dimensional array Multi-dimensional hash Multi-dimensional hash emulation						
list and Array <ul style="list-style-type: none">0-based indexed (first index is 0).Last index of array @name is \$#name		@	@days @days[3,4,5] @days[3..5]	Array containing (\$days[0], \$days[1], ... \$#days[\$#days]) . Array slice containing (\$days[3], \$days[4], \$days[5]) . Array slice containing (\$days[3], \$days[4], \$days[5]) .			<ul style="list-style-type: none">A <i>list</i> is an ordered collection of scalars (of any type).An <i>array</i> is a variable that contains a list.Reading beyond the end of array returns undef			
<ul style="list-style-type: none">slices Simple explanation		<ul style="list-style-type: none">Use a slice to select multiple elements from a list, array, or hash.Don't use a slice when you know you need exactly one element.An lvalue slice imposes list context on the righthand side.			my @digits = (0..9); my @one2five = @digits[1..5]; my @premiers = @digit[1, 2, 3, 5, 7];			my @extracted = (7, 2, 0, 9); my @choices = @digits[@extracted] my \$mod_time = (state \$filename)[9];		
<ul style="list-style-type: none">Anonymous arrays		<ul style="list-style-type: none"><u>What are the advantages of anonymous array? @ StackOverflow</u><u>Perldref @ Perldoc</u>, Perl reference tutorial @ Perldoc			<ul style="list-style-type: none">Anonymous array := a type of array reference.Array reference allows Perl to treat the array as a single item.<ul style="list-style-type: none">This can be used to build, nested data structures.					
Hash/associative array		%	%days @days{'J','F'}	Associative array (hash): keys-value pairs. Can be initialized as: <ul style="list-style-type: none">%days = (Jan => 31, Feb => \$leap? 29 : 28, ...)%days = ("Jan", 31, 'Feb', \$leap? 29 : 28, ...) Hash slice containing (\$days{'J'}, \$days{'F'}) .			Initialize a hash slice with array context: @char_to_num{'A' .. 'Z'} = 1 .. 26;			
Subroutine		&	&foo	& is needed to create reference to subroutine.						
Typeglob		*	*foo	See: <u>Advanced Perl Programming</u> , 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 <ul style="list-style-type: none">how to format output in Perl?, Perl-FormatsSee write and select		6. file handles 7. directory handles		
Scalar values		Numeric		literals examples:		Note: leading 0 work only for literals, not for string-to-number conversions.		Useful related builtin functions		
<ul style="list-style-type: none">numeric:		<ul style="list-style-type: none">integer : using the system's native format.<ul style="list-style-type: none">bigint - transparent big integer support.bignum - transparent big number support.floating-point : using the system's native format.<ul style="list-style-type: none">bigrat - transparent big rational number support. <p><i>A variable holding an integer can be converted to floating-point if the operation done to it requires it (such as dividing 1 by 2).</i></p>		my \$x = 12345; my \$x = 12345.67; my \$x = 6.02e23; my \$x = 0x1f.0p3; my \$x = 4_294_967_296; my \$x = 0x1234_5678; my \$x = 0377; my \$x = 0o377; my \$x = 0xffff; my \$x = 0b1100_0010;		# integer # floating point # scientific notation # power ² exponent: <i>Perl >= v5.22</i> # underline for legibility # underline in hex is also OK # octal # octal also <i>Perl >= v5.34</i> # hexadecimal # binary with underlines		<ul style="list-style-type: none">oct - supports binary, octal, hexhexPOSIX::ceilPOSIX::floorabs		
<ul style="list-style-type: none">string		<ul style="list-style-type: none">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.Single quote and double quote strings can spread multiple lines: it embeds the newline character on each new line.But \a is only expanded in double quoted strings! In single quote string it is treated as two characters; no substitution is done (as explained above).								
<ul style="list-style-type: none">Unicode support		Use Unicode literally in a program; add the utf8 pragma : use utf8 ; See: Perl Unicode Tutorial , Perl Unicode Introduction , Perl Unicode Support @ perldoc								
<ul style="list-style-type: none">Quote constructs		Customary	Generic	Meaning	Interpolates?	Notes				
See: <ul style="list-style-type: none">Strings in Perl: quoted, interpolated and escaped	''	q//	Literal string	No	<ul style="list-style-type: none">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:<pre>my \$chuck_of_code = q { if (\$condition) { print "Salut!"; } };</pre>					
	""	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						
<ul style="list-style-type: none">Character escapes (only inside double quoted strings)		\a	Alert (bell)	\e	ESC character	Any Unicode code point, by name:				
	\b	Backspace	\033	ESC in octal						
	\e	ESC character	\o{33}	ESC in octal			\N{LATIN SMALL LETTER E WITH ACUTE} é			
	\f	Form feed	\x7f	DEL in hexadecimal			\N{ U+E9 } é			
	\n	Newline (usually LF)	\x{263a}	Character number 0x263A						
	\r	Carriage return (Usually CR)	\cC	Control-C						
	\t	Horizontal tab								
<ul style="list-style-type: none">translation escapes (inside double quoted strings)		\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 Unicode fold case. Ends at \E						
			\Q	Backslash all following non alphanumeric characters. Ends at \E						
<ul style="list-style-type: none">bareword		In Perl, a <i>bareword</i> refers to a sequence of characters suitable for an identifier. It's not quoted. By default Perl allows barewords to behave like strings. <ul style="list-style-type: none">This is not allowed when any of use strict; or use strict "subs"; or use v5.12; is specified.								
<ul style="list-style-type: none">Here documentsHere docs @ Perl mavenPerl 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: <ul style="list-style-type: none">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.								
<ul style="list-style-type: none">Perl Regexp		Regexp Tutorial, Learn PCRE in X minutes, PCRE cheatsheet,			Debuggex regexp tester, regex101, RegEx Pal					
<ul style="list-style-type: none">index/substr		\$pos = index (\$page, \$line);		\$last_slash = rindex ("usr/bin/ls", "/");		\$part = substr (\$text, \$pos, \$len)		A value of -1 in pos identifies last character.		
<ul style="list-style-type: none">Replacementmanipulate strings with substr LPo		my \$pref = "I like awk and erlang"; substr (\$pref, index (\$pref, "awk"), length ("awk")) = "Perl"; substr (\$pref, 0, 0) = "Sally and "; # insert text anywhere			substr (\$pref, -15) =~ s/Perl/Perl5/g; # replace text inside a restricted portion of the string.					

Current value of warning switch	<ul style="list-style-type: none"><code>\$WARNING</code><code>\$^W</code>		Current set of warning checks enabled by the use warnings pragma	<code>\${^WARNING_BITS}</code>	
<ul style="list-style-type: none">Variables related to the interpreter state	These variables provide information about the current interpreter state.				
Flag associated with the -c switch	<ul style="list-style-type: none"><code>\$COMPILING</code><code>\$^C</code>		The current value of the debugging flags	<ul style="list-style-type: none"><code>\$DEBUGGING</code><code>\$^D</code>	
Current phase of the perl interpreter	<code>\${^GLOBAL_PHASE}</code>		Debugging support. Internal variable.	<ul style="list-style-type: none"><code>\$PERLDB</code><code>\$^P</code>	
Compile-time hints for the perl interpreter. Internal use only	<code>\$^H</code>		Values of compiled statements	<code>%^H</code>	
Taint mode	<code>\${^TAINT}</code>		Safe locale operations availability	<code>\${^SAFE_LOCALES}</code>	
Input/Output Layers. Internal use by PerlIO only.	<code>\${^OPEN}</code>		Unicode Settings of Perl	<code>\${^UNICODE}</code>	
Internal UTF-8 offset caching code state	<code>\${^UTF8CACHE}</code>		State of UTF-8 locale detected by perl at startup.	<code>\${^UTF8LOCALE}</code>	
<ul style="list-style-type: none">File handle Variables	See also: <u>Perl File Handles</u> The following variables are used in the Input/Output handling as well as program arguments.				
Name of current file read from <>	<code>\$ARGV</code>	Command line arguments of the script ← See <u>diamond operator</u> <>. →	<code>@ARGV</code>	Number of arguments minus one	<code> \$#ARGV</code>
Special file handle that iterates over command-line filenames in @ARGV	<code>ARGV</code>	Special file handle that points to currently open output file when doing edit-in-place processing	<code>ARGVOUT</code>		
Output field separator for the print operator	<ul style="list-style-type: none"><code>IO::Handle->output_field_separator(EXPR)</code><code>\$OUTPUT_FIELD_SEPARATOR</code><code>\$OFS</code><code>\$,</code>		Current line number for the last file handled accessed	<ul style="list-style-type: none"><code>HANDLE->input_line_number(EXPR)</code><code>\$INPUT_LINE_NUMBER</code><code>\$NR</code><code>\$.</code>	
Input record separator (newline by default)	<ul style="list-style-type: none"><code>IO::Handle->input_record_separator(EXPR)</code><code>\$INPUT_RECORD_SEPARATOR</code><code>\$RS</code><code>\$/</code>		Output record separator	<ul style="list-style-type: none"><code>IO::Handle->output_record_separator(EXPR)</code><code>\$OUTPUT_RECORD_SEPARATOR</code><code>\$ORS</code><code>\$\</code>	
Auto-flush control <ul style="list-style-type: none">order of output @ Perl MavenSuffering from Buffering?	<ul style="list-style-type: none"><code>HANDLE->autoflush(EXPR)</code><code>\$OUTPUT_AUTOFLUSH</code><code>\$!</code>	Perl activates file buffering by default. Assign 1 to <code>\$!</code> to activate auto-flush.	<u>Last read file handle</u>	<code>\${^LAST_FH}</code>	

Perl 5 Input/Output 🚧

References	<ul style="list-style-type: none">• open @ perldoc browser• Writing to files with Perl @ Perl Maven• open file in-memory @ stackOverflow• Stupid open() tricks @Perl.com:<ul style="list-style-type: none">• No explicit filename• create an anonymous temporary file• print to a string• read lines from a string				
print, printf, sprintf	print, printf, sprintf (which describes the format) . Note: print is more efficient than printf . print and printf output to stdout by default, but accept a file handle as the first argument if it is NOT followed by a separating comma! (a ' , puts it in the list to print!)				
diamond operator <>	Both <> and <<>> operators read the content of files listed on the command line via @ARGV . Nothing or - on the command line identifies stdin. The <> operator supports shell redirection and pipe operations which <<>> does not allow (for security reasons).				
The double diamond, a more secure <> (Perl >= v5.22)	print <>;	← Simple implementation of /bin/cat	print <<>>;	← safer one	Redirection cannot be forced via file names embedding them with. the <<>> operator.
	print sort <>;	← Simple implementation of /bin/sort	print sort <<>>;	← safer one	
 In-place-editing ↔ The <> operator tries to duplicate the original file's permission and ownership.	Set \$^I to a backup file extension (such as Emacs "~" or ".bak") to change the behaviour of the <> and <<>> operators and print. In a while (<>) { ... } loop, when \$^I is not undef (its default), Perl: <ul style="list-style-type: none">• renames currently processed file with the specified extension added,• opens a new file with the original name• prints into the new file.• Any modification goes into the new file: in-place-editing it!		<pre>use strict; \$^I = "~"; # rename old file: add '~' to it's name (Emacs-style backup) while (<>) { s/something/Something else/; # perform any substitution print; }</pre>		
perl -i cmdline option	It's also possible to do this on the command line!		For example:	perl -p -i- -w -e 's/something/Something else/g' data*.dat	
Special filehandle names	ARGV	The special filehandle that iterates over command-line filenames in @ARGV. Usually written as the null filehandle in the angle operator <> (or <<>>)			
Also See: • File handle Variables section above.	ARGVOUT	The special filehandle that points to the currently open output file when doing edit-in-place processing with _i . <ul style="list-style-type: none">• Useful when you have to do a lot of inserting and don't want to keep modifying \$_			
	STDIN	<STDIN> : line input operator for the STDIN filehandle (for the standard input). <ul style="list-style-type: none">• Each time <STDIN> is used in scalar context, Perl reads 1 complete line of the standard input and uses it as the value of <STDIN>.<ul style="list-style-type: none">• The string includes a line termination character. Use the chomp built-in function to strip it off the variable.• If <STDIN> is read in list context, it returns all lines inside a list! For example, foreach (<STDIN>) { ... } reads the entire stdin in 1 step: \$_ holds it all!			
		<pre>while (<STDIN>) { # print all print; # lines of # stdin }</pre>	<pre>while (defined(\$_ = <STDIN>)) { print \$_; }</pre>	The code in the left-most cell is the shortest form. It is equivalent to the code beside it; each line of stdin is stored in the default variable \$_ and the loop stops on end at which time <STDIN> returns undef .	
	STDOUT	standard output			
	STDERR	standard error Note: generally STDERR is not buffered, while STDOUT is buffered by default. Text sent on STDERR may show up before STDOUT. <ul style="list-style-type: none">• Print a new line on STDOUT to help flushing it or assign 1 to \$ to activate auto-flush.			
	DATA				
say	• say <code>use feature qw(say);</code> or <code>use v5.10;</code> (or higher). Like print, but implicitly appends a newline at the end of the list.				
open					


Perl 5 Statements ⚠️

Loop control	See perlsyn for more information on Perl syntax which includes declarations, blocks, loops, labels, subroutines, etc...		
		The last , next , and redo loop control keywords work in the following constructs:	Notes:
	loop control keywords: <ul style="list-style-type: none">last g: exits the loop.next g: starts the next iteration of the loop.redo g: restarts the loop block without evaluating the condition again.	<ul style="list-style-type: none">while (condition) { ... }until (condition) { ... }for (init; condition; continue) { ... }foreach array { ... }naked block: { ... }	<ul style="list-style-type: none">The while and foreach loops may have a continue block: executed before evaluating condition again, which corresponds to the 3rd part of a for loop statement. See this @ stackOverflow.Blocks can be labelled g as targets to last, next, and redo
Statement modifiers	<ul style="list-style-type: none">if EXPRunless EXPRwhile EXPRuntil EXPRfor LISTforeach LISTwhen EXPR	The for and foreach statements impose a list context ; the complete list is processed. Therefore a loop like the following trying to stop on a line that has " __END__ " on it will not work since it reads all of STDIN: <pre>foreach (<STDIN>) { last if ?__END__ /; ...; }</pre>	The while statement imposes a scalar context ; it takes one line at a time from <STDIN> and the following code works properly: <pre>while (<STDIN>) { last if /__END__ /; ...; }</pre>
	<ul style="list-style-type: none">do block		
Conditional statements			


Perl 5 Subroutines ⚠️

Perl subroutines			
subroutine &	<ul style="list-style-type: none">Why we teach the subroutine ampersandWhy should I use the & to call a Perl subroutine? @ StackOverflow		Another point of view: Subroutines and Ampersands
Subroutine Prototypes	An older Perl feature. Clashes with subroutine signatures as of Perl v5.20. In <i>Perl >= v5.20</i> put the :prototype attribute before subroutine prototype parenthesis.		
Subroutine signatures • <i>Perl >=5.36</i> : Stable • <i>Perl >= 5.20</i> : Experimental See: Use v5.20 subroutine signatures	Exactly zero arguments	()	Zero or 1 argument, no default, unnamed: (\$=)
	Zero or 1 argument, no default, named	(\$val=)	Zero or 1 argument, named, with default (\$val=1)
	exactly 1 named argument:	(\$val)	Exactly 2 arguments (\$v1, \$v2)
	2, 3 or 4 arguments no defaults:	(\$v1, \$v2, \$=, \$=)	2,3 or 4 arguments, 1 default: (\$v1, \$v2, \$v3='a', \$=)
	Two or more, any number of arguments.	(\$v1, \$v2, @)	Two or more arguments, remainders into a named array: (\$v1, \$v2, @rest)
	Two or more arguments: an even number	(\$v1, \$v2, %)	Two or more arguments, remainders into a named hash: (\$v1, \$v2, %rest)
	Class method	(\$class, ...)	Object method (\$self, ...)
Variables in subroutines	global by default		
	my	local, lexical scope, non persistent	
	state	Local, lexical scope, persistent <i>Perl >= v5.10</i>	Restriction: in <i>Perl < v5.28</i> : array and hashes state cannot be initialized in list context.
	our	creates a lexical scoped alias to a package variable	
	local	Localizes an existing package variable to the current scope. It's not a declaration. The variable previous value is restored when leaving the scope.	
Returned value	<ul style="list-style-type: none">The result of the last evaluated expression is implicitly returnedThe return operator can be used but it's not required unless used to change execution flow (return immediately from the subroutine).The subroutine can return a scalar in scalar context or a list if called in list context.<ul style="list-style-type: none">Inside the subroutine, use the wantarray function to determine the context of the subroutine call.		

Perl 5 Built-in Functions ⚠️

Perl Functions Perl syntax	 To get information about a Perl function from the command line use the perldoc -f command. <ul style="list-style-type: none">To get information about print use: perldoc -f print		
⚠️ Cautionary notes			
<ul style="list-style-type: none">each keyword is brokenUse 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. <ul style="list-style-type: none">each is not re-entrant:<ul style="list-style-type: none">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.<ul style="list-style-type: none">If you use each on the same hash later it will resume from where it left, it will not start form the beginning.		

Perl 5 Modules ⚠️

Perl Modules			
Perl core modules	<ul style="list-style-type: none">How to detect where a module is installed : perldoc -l ModuleHow to check if a module is part of Perl core : corelist Module (<i>Perl >= v5.9.2</i>)		
Modules @perltutorial Modules Using simple modules g	do	Looks for the module file by searching the @INC path. Performed at run time (and therefore can be done conditionally). <ul style="list-style-type: none">If Perl finds the file, it places the code inside the calling program and executes it. Otherwise, Perl will skip the do statement silently. The "included" code does not have access to the lexical variables from the main program.Skip the @INC path lookup if given a file path starting with ./, ../, or /	
	require	Loads the module file once, also searching the @INC path. Performed at run time (and therefore can be done conditionally). <ul style="list-style-type: none">If the require for the same file appears twice, Perl ignores it. Perl will issue an error message if it cannot find the file (as opposed to do).Skip the @INC path lookup if given a file path starting with ./, ../, or /	
The <i>normal</i> way to access Perl modules ➡	use	Similar to require except that Perl applies it before the program starts: it's done at compile time . <ul style="list-style-type: none">Therefore the use statement cannot be invoked inside conditional statements such as if-else. Used often to include a module in a program.	
Error handling for: Can't locate in @INC • How to fix that	For the above statements to work Perl must be able to identify the location of the requested module(s). <ul style="list-style-type: none">Perl looks for a module code inside the directories identified by the @INC array. if you have. use The::Module; inside your code, Perl looks for a sub-directory named 'The' containing a file named 'Module.pm' inside each @INC directory. If it does not find it, there are multiple ways to solve the problem : <ul style="list-style-type: none">Add the required directory to the list of directories identified in the ':' separated list in the PERL5LIB environment variable. (use ';' as separators in Windows).Add a use lib 'path/to/the/directory'; statement inside your Perl file to dynamically add the required directory when executing a specific piece of Perl code.Run Perl with the -I(capital i) option to run the code with the extra directory added to @INC array.		

Topic: Directory Operations🚧

Directory Operations	In Books: LPo		
Opening Files	All file open operations are relative to the <i>current working directory</i> (for relative file names)		<code>open my \$filehandle, '<:utf8', 'a_relative/path.txt'</code>
Creating temporary files	File::Temp (Perl >= v5.6.1). Using File::Temp <ul style="list-style-type: none">Also see IO::File		
Built-in Functions	Related Functions/Packages / Descriptions		Notes
Getting file names by: <ul style="list-style-type: none">Globbering :<ul style="list-style-type: none">with globwith the glob operator <>	File::Glob (Perl >= v5.6.0) - provides more control.		Example: <code>my @all_files = glob '*';</code> <code>my @perl_files = glob '*.pm *.pl'; # 2 globs, space-separated</code>
	The <> operator is identifying: <ul style="list-style-type: none">a filehandle, when: the item inside <> is a Perl identifier or an indirect file handle read scalar,a glob expression otherwise.		Glob examples: <code>my @all_files = <'*>;</code> <code>my @all_files = <*>; # 1 glob: no space, no need for string</code> <code>my @perl_files = <'*.pm *.pl*>; # 2 globs, space-separated</code>
	See: readline		<code>my \$etc_dir = '/etc';</code> <code>my @etc_dir_files = <\$etc_dir/* \$etc_dir/*>;</code>
			<code>my @files = <LARRY/*>; # a glob</code>
with a directory handle LPo	Example: iterate explicitly over a list of file names extracted from the directory using these 3 functions.		<code>my \$dir = '/usr/bin';</code> <code>opendir my \$dh, \$dir or die "Failed opening \$dir: \$!";</code> <code>foreach \$file (readdir \$dh) {</code> <code> print "File \$file is inside \$dir\n"; # ⚠️ no path in name!</code> <code>}</code> <code>closedir \$dh;</code>
			<code>my \$name = 'LARRY';</code> <code>my @his_lines = <\$name>; # indirect filehandle read of LARRY handle</code> <code>my @same_lines = readline LARRY; # another way to write above</code> <code>my @same_lines = readline \$name;</code>
Creating directory	<ul style="list-style-type: none">mkdir		Example: <code>mkdir \$dir_name, oct(\$permissions); # octal for permissions</code> <code>mkdir \$dir_name, 0700; # do not use "0700", it's 700 decimal!</code>
Removing directory	<ul style="list-style-type: none">rmdir Removes an empty directory.File::Path remove_tree , rmtree remove dir & files (Perl >= v5.0.1)		
Removing files	<ul style="list-style-type: none">unlink a list or \$_		<code>unlink 'file1.txt', 'file2.txt';</code> <code>unlink qw(file1.txt file2.txt);</code> <code>unlink glob 'file?.txt'</code>
Renaming files	<ul style="list-style-type: none">rename an old file name to a new one.<ul style="list-style-type: none">The fat comma operator is sometimes used to highlight what is the old and the new name.		As in here: <code>rename 'old_name' , 'new_name';</code> <code>rename old_name => new_name; # using fat comma (which quotes)</code>
Changing permissions	<ul style="list-style-type: none">chmod changes file permissions		
Changing ownership	<ul style="list-style-type: none">chown changes file ownership		
Creating Hard link	<ul style="list-style-type: none">link to create a hard link		
Creating symbolic link	<ul style="list-style-type: none">symlink to create a symbolic link		
chdir Change current working directory	<ul style="list-style-type: none">File::chdirFile::HomeDir		<ul style="list-style-type: none">Change the current working directory.chdir without argument attempt to change to user home directory using the \$ENV{HOME} and \$ENV{LOGDIR} environment values if ⚠️ they are set. The File::HomeDir module helps in setting them.The built-in chdir is global ⚠️ for the entire program. Use File::chdir facilities for localized operations.
Modules	Functions Legend: Exported by default , exported on request, <i>Win32 specific</i>		Extra Information
Cwd	<ul style="list-style-type: none">getcwd, cwd, fastcwd, fastgetcwd, <i>getdcwd</i>abs_path, realpath, fast_abs_path		<code>use Cwd;</code> <code>my \$curdir = getcwd;</code> <code>print "cwd is \$curdir\n";</code>

Topic: Process control🚧

Process Control	In Books: LPo		Important security information: perldoc perlsec	
Environment Variables	Inside the %ENV hash.		Perl %Config hash: Perl configuration information. For example, whether it support threads, what are path separators, etc... <ul style="list-style-type: none">To use it: <code>use Config;</code>	
Built-in Functions	Example		Description/ Notes	
system (2 functions) <ul style="list-style-type: none">using the shell<ul style="list-style-type: none">security risk?avoiding the shell<ul style="list-style-type: none">other syntax	system 'ls -l \$HOME';		Run child process asynchronously using parent's stdin, stdout and stderr, using the OS native command shell.	
	system "cd \$project; make &;"		Use the Unix shell to execute a long running build asynchronously. 🙌 However: avoid using the shell like this . <ul style="list-style-type: none">Using the shell to build commands from unvalidated user input data may lead to security issues.	
	system 'tar', 'cvf', \$tarfile, @directories;		No shell invoked when more than 1 argument is passed to system. No shell interpretation, piping, re-direction done.	
	system ('tar', @arguments);		0 means success: <code>unless (system 'tar', arguments) { print "tar command success\n"; }</code>	
	system ({ \$prog }, \$arg0, @args);			
	👉 Note that if the string contain no shell metacharacters it is executed directly (not through a shell).			
system return value: <ul style="list-style-type: none">A value of 0 usually means all was OK.	2 bytes:	MSByte: child program exit code.	<code>my \$retval = system(...);</code>	
		LSByte: system-specific information bits: <ul style="list-style-type: none">0x80 : set on core dump.0x7f : signal number	<code>my \$childp_exitcode = \$retval >> 8;</code> <code>my \$had_core_dump = (\$retval & 0x80) == 0x80? 1 : 0;</code> <code>my signal_number = \$retval & 0x7f;</code> <div>← shift most significant byte</div> <div>← use least significant byte</div>	
exec	Unlike system, exec does not return to the parent Perl process. Use: <code>exec 'the_program' or die 'Could not run: \$!'; #or warn or exit</code>			
backquotes ``	Use backquotes to capture the stdout of a program. That's the main point of using it. <ul style="list-style-type: none">The trailing newline is not filtered out; it can be filter by chomp.			<code>chomp(my \$current_date = `date`);</code>
	<ul style="list-style-type: none">The value inside the backquotes is treated like the single double quote string argument of system: it will invoke the shell if there are any shell meta-characters and supports interpolation.<ul style="list-style-type: none">The following example builds a dictionary (hash) of topics with the text extracted from perldoc.Note that <code>`...`</code> is also written as qx/ ... /backquote operation in scalar context returns 1 string. In list context it returns a list of strings (1 per line).			<code>my @topics = qw(die warn exit);</code> <code>my %info;</code> <code>foreach (@topics) {</code> <code> \$info{\$_} = `perldoc -t -f \$_`;</code> <code>}</code>
Modules				
Capture streams	<ul style="list-style-type: none">Capture::Tiny		Can be used to capture the stdout and stderr streams for various ways if executing other programs	
Inter-process support	<ul style="list-style-type: none">IPC::System::Simple		Can also be used to capture streams and provide more inter-process support. <ul style="list-style-type: none">It provides systemx which never uses the shell, along with other useful functions.	

Processes as filehandles	In Books: LPα		
Perl ← program	Launching a process that pipes into the Perl process	<code>open DATE, 'date '</code> or die "Cannot pipe from date: \$!";	Use a bare word to define the DATE file handle.
		<code>open my \$date_fh, '- ', 'date'</code> or die "Cannot pipe from date: \$!";	This one and the others define a local file handle variable. The file handle variable can later be used to read, as the above one, but is not global.
		<code>open my \$ps_fh, '- ', 'ps', 'aux'</code> or die "Cannot pipe from ps: \$!";	
		<code>open my \$find_fh, '- ', 'find', qw(. -name '*.p[lm]' -print)</code> or die "Cannot pipe from find: \$!";	
Perl → program	Launching a process that the Perl process pipes into.	<code>open my \$dispatcher_fh, ' -', 'dispatcher', qw('—to-perl-groups' 'Help!')</code> or die "Cannot pipe to the dispatcher: \$!";	
Forking	In Books: LPα . See also: Linux fork(2) system call, QA: Why do we need fort to create new processes? Why fork woks the way it does?		
fork with exec and waitpid See also: <ul style="list-style-type: none">Other IPC functionsPerl IPC	<ul style="list-style-type: none">fork the process into parent and child.in the child process start the program with <code>exec</code>In the parent process wait for the program termination with <code>waitpid</code>	<pre>defined(my \$process_id = fork) or die "Fork failed: \$!"; unless (\$process_id) { # Inside the child process (created by fork) exec 'long_running_process' or die "Failed starting long_running_process: \$!"; } # Inside the parent process, wait for completion of long_running_process. waitpid(\$process_id, 0);</pre>	
Signals	In Books: LPα		
kill	Sends a signal to a list of processes. <ul style="list-style-type: none">The signal may be identified by number or name (string), which is more portable.The <code>%Config{sign_name}</code> provides the supported signal names.	kill 'INT', \$pid or die "Can't signal \$pid with SIGINT: \$!";	
	<ul style="list-style-type: none">Note that the <i>fat comma</i> operator (<code>=></code>) can be used to automatically quote signal name:	kill INT => \$pid or die "Can't signal \$pid with SIGINT: \$!";	
	<ul style="list-style-type: none">If the signal is 0 or "ZERO" no signal is sent to the process; instead Perl checks if it's possible to send a signal to the process: ie: if the process exists.	<pre>unless (kill 0, \$process_id) { warn "Process \$process_id is no longer running!"; }</pre>	
Signal handlers	<ul style="list-style-type: none">If the signal is a negative number or a string that starts with '-' the signal is sent to the process group identified by the process scalar argument.Set the signal handler by setting <code>%SIG</code> for the signal name (with no 'SIG' prefix) to a string holding the name of the subroutine.	<ul style="list-style-type: none">kill '-KILL', \$process_groupkill -9, \$process_group	
		<code>\$SIG{'INT'} = 'dispatcher_int_handler';</code>	

PerlTidy formatting control 🚧

perltidy option	Option	Impact
indentation style	<ul style="list-style-type: none"> <code>-bl</code>, <code>--opening-brace-on-new-line</code> <code>--brace-left</code> 	<ul style="list-style-type: none"> Without this option (the default) the code indentation style selected is K&R style. With this option, the indentation style is Allman/BSD style.