















Description	Keystroke		Function	Note
Comments	Perl comments start with #. See <a href="#">ℹ Comments</a> for the generic commands that manage and control comments			
Toggle display of comments in buffer or active region	<f11> ; ;	(hide/show-comments-toggle &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. <ul style="list-style-type: none"><li>If the region is active then toggle in the region. Otherwise, in the whole buffer.</li></ul>	
	📦 This requires the <a href="#">hide-comnt.el</a> package (see <a href="#">ℹ Comments</a> ). 🗑️ PEL activates it when the <b>pel-use-hide-comnt</b> user option is t.			
Generic code skeletons <ul style="list-style-type: none"><li><a href="#">tempo skeletons</a></li></ul> See also: <ul style="list-style-type: none"><li><a href="#">ℹ Inserting Text</a></li><li><a href="#">T Templates</a></li></ul>	Several mechanisms have been developed to allow easy insertion of predefined text in Emacs. ⚠ PEL does not yet define skeletons for Perl. You can use the generic one. <ul style="list-style-type: none"><li>Emacs provides the built-in skeleton mechanism and the <b>tempo skeletons</b>.<ul style="list-style-type: none"><li>PEL supports both. They are used a little bit differently. PEL provides <b>generic tempo skeletons you can use for Perl until PEL adds Perl-specific skeletons</b>.<ul style="list-style-type: none"><li>PEL provides key bindings to the tempo skeletons: the generic code templates, accessible via the &lt;f6&gt; prefix key, and the language-specific code templates, accessible via the &lt;f12&gt; key prefix.</li></ul></li></ul></li></ul>			
ℹ Customize PEL Text Insertions control for Perl code skeletons.	<f6> <f2>	(pel-customize-pel &optional OTHER-WINDOW)	Open the customization groups that control the format of the various skeletons including the generic skeleton used by the <f6> h key and the <f12><f12> h key (see below). <ul style="list-style-type: none"><li>If OTHER-WINDOW is non-nil (use C-u), display in other window.</li></ul>	
	<f12> <f12> <f2>	(pel-customize-generic-skels &optional OTHER-WINDOW)		
Insert generic file module header block — Language agnostic	<f6> h	(pel-generic-file-header)	Insert a file header block at the top of the file. Works only for buffer visiting a file.	
	<f12> <f12> h		⚠ The command key binding <f6> h is available only 1 second after Emacs has started. ⚠ As mentioned above PEL does not yet define Perl-specific skeletons, this uses the generic one.	
After inserting the template, navigate though areas that must be filled with: <ul style="list-style-type: none"><li>tempo-forward-mark: C-c .</li><li>tempo-backward-mark: C-c ,</li></ul>	👉 Specify the format of the header via the user-options in the <b>pel-pkg-generic-code-style</b> customization group accessible via <f6> <f2> <ul style="list-style-type: none"><li>Inside a <b>Perl</b> buffer, &lt;f12&gt; &lt;f2&gt; provides access to the following customization groups:</li></ul> 👉 After inserting a template, use <b>tempo-forward-mark</b> and <b>tempo-backward-mark</b> to move to the beginning of each section that must be filled.			
Toggle pel-tempo-mode	<f6> SPC	(pel-tempo-mode &optional ARG)	Toggle PEL tempo mode on/off.	
	<f12> <f12> SPC			
PEL tempo mode activates C-c . and C-c , , as well as to C-c C-. and C-c C-, key bindings to navigate across tempo mark hot-spots. When pel-tempo-mode is active the pel-tempo-mode lighter (⚡) is shown on the status bar. The second set of keys are only available in graphics mode. 👉 The pel-generic-file-header command inserts the text using a tempo skeleton: the PEL tempo mode is automatically activated by typing <f6> h.				
Expand any tag in template	<f6> <f12>	(tempo-complete-tag &optional SILENT)	Look for a tag and expand it. All the tags in the tag lists in <b>'tempo-local-tags'</b> (this includes 'tempo-tags') are searched for a match for the text before the point. The way the string to match for is determined can be altered with the variable 'tempo-match-finder'. If 'tempo-match-finder' returns nil, then the results are the same as no match at all. <ul style="list-style-type: none"><li>If a single match is found, the corresponding template is expanded in place of the matching string.</li><li>If a partial completion or no match at all is found, and SILENT is non-nil, the function will give a signal.</li><li>If a partial completion is found and 'tempo-show-completion-buffer' is non-nil, a buffer containing possible completions is displayed.</li></ul>	
Note: PEL default skeleton does not use tags.	<f12> <f12> <f12>			
Navigation Commands	Some navigation commands have special behaviour for Perl; they are shown below. They are available in <b>perl-mode</b> and <b>cperl-mode</b> . <ul style="list-style-type: none"><li>There's also several generic navigation commands that work in Perl buffer too. See <a href="#">ℹ Navigation</a> for those.</li></ul>			
Move to beginning of function	<ul style="list-style-type: none"><li>C-M-a</li><li>&lt;f12&gt; &lt;up&gt;</li></ul>	(perl-beginning-of-function &optional ARG)	Move backward to next beginning-of-function, or as far as possible. With argument, repeat that many times; negative args move forward.	
To end of function	<ul style="list-style-type: none"><li>C-M-e</li><li>&lt;f12&gt; &lt;down&gt;</li></ul>	(perl-end-of-function &optional ARG)	Move forward to next end-of-function. <ul style="list-style-type: none"><li>The end of a function is found by moving forward from the beginning of one.</li><li>With argument, repeat that many times; negative args move backward.</li></ul>	
Move to next interpolated	C-c C-v	(cperl-next-interpolated-REx &optional SKIP BEG LIMIT)	Move point to next REx which has interpolated parts. <ul style="list-style-type: none"><li>SKIP is a list of possible types to skip, BEG and LIMIT are the starting point and the limit of search (default to point and end of buffer).</li><li>SKIP may be a number, then it behaves as list of numbers up to SKIP; this semantic may be used as a numeric argument.</li><li>Types are 0 for / \$rex /o (interpolated once), 1 for /\$rex/ (if \$rex is a result of qr//, this is not a performance hit), t for the rest</li></ul>	
	C-c C-x	(cperl-next-interpolated-REx-0)	Move point to next REx which has interpolated parts without //o.	
	C-c C-y	(cperl-next-interpolated-REx-1)	Move point to next REx which has interpolated parts without //o. <ul style="list-style-type: none"><li>Skips REXes consisting of one interpolated variable.</li><li>Note that skipped REXen are not performance hits.</li></ul>	
Marking Commands	The following marking commands are specialized for Perl. See <a href="#">ℹ Marking</a> for the generic ones that can also be used in Perl buffers.			
Mark function	C-M-h	(perl-mark-function)	Put mark at end of Perl function, point at beginning	
Indentation Control	The following indentation control commands are specialized for Perl. See <a href="#">ℹ Indentation</a> for the generic ones that can also be used in Perl. <ul style="list-style-type: none"><li>The indentation behaviour of the &lt;tab&gt; key is controlled by the <b>perl-mode</b> and <b>cperl-mode</b> customization user-options. Use &lt;f12&gt; &lt;f3&gt; to open customization.</li></ul>			
Show currently used indentation style	<f12> <f4> s	(pel-perl-show-style)	Show the name of the currently used indentation style.	
Select new/restore old indentation style	<f12> <f4> <TAB>	(pel-perl-set-style)	Set Perl indentation style to named style. Prompt for indentation style name and apply it. ⚠ This change does <b>not</b> persist. Manually access the customized buffer and save it.	
Indent <ul style="list-style-type: none"><li>perl-mode ➡</li></ul>	<tab>	(perl-indent-command &optional ARG)	Indent Perl code in the active region or current line. In Transient Mark mode, when the region is active, reindent the region. <ul style="list-style-type: none"><li>Otherwise with a prefix argument, reindent the current line unconditionally.</li><li>Otherwise if <b>'perl-tab-always-indent'</b> is nil and point is not in the indentation area at the beginning of the line, insert a tab.</li><li>Otherwise, indent the current line. If point was within the indentation area, it is moved to the end of the indentation area. If the line was already indented properly and point was not within the indentation area, and if 'perl-tab-to-comment' is non-nil (the default), then do the first possible action from the following list:<ol style="list-style-type: none"><li>delete an empty comment</li><li>move forward to start of comment, indenting if necessary</li><li>move forward to end of line</li><li>create an empty comment</li><li>move backward to start of comment, indenting if necessary.</li></ol></li></ul>	
		(cperl-indent-command &optional WHOLE-EXP)	Indent current line as Perl code, or in some cases insert a tab character. <ul style="list-style-type: none"><li>If <b>'cperl-tab-always-indent'</b> is non-nil (the default), always indent current line. Otherwise, indent the current line only if point is at the left margin or in the line's indentation; otherwise insert a tab.</li><li>A numeric argument, regardless of its value, means indent rigidly all the lines of the expression starting after point so that this line becomes properly indented. The relative indentation among the lines of the expression are preserved.</li></ul>	
Indent continued expression <ul style="list-style-type: none"><li>cperl-mode ➡</li></ul>	C-M-q	(perl-indent-exp)	Indent each line of the Perl grouping following point.	
		(cperl-indent-exp)	Simple variant of indentation of continued-sexp. <ul style="list-style-type: none"><li>Won't indent comment if it starts at <b>'comment-indent'</b> or looks like continuation of the comment on the previous line.</li><li>If <b>'cperl-indent-region-fix-constructs'</b>, will improve spacing on conditional/loop constructs.</li></ul>	
Indent region	C-M-\	(cperl-indent-region START END)	Indents all the lines whose first character is between START and END inclusive. <ul style="list-style-type: none"><li>Won't indent comment if it starts at <b>'comment-indent'</b> or looks like continuation of the comment on the previous line.</li><li>If <b>'cperl-indent-region-fix-constructs'</b>, will improve spacing on conditional/loop constructs.</li></ul>	
Newline and indent <ul style="list-style-type: none"><li>cperl-mode ➡</li></ul>	C-j	(newline-and-indent)	Insert a newline at point, then indent the newly created line. Use it to split a line. <ul style="list-style-type: none"><li>Indentation is done using the value of <b>'indent-line-function'</b> which is set to <b>cperl-indent-line</b>.</li></ul>	
Insert Perl new line	C-c C-j	(cperl-linefeed)	Go to end of line, open a new line and indent appropriately. If in POD, insert appropriate lines.	
	It's a convenience replacement for typing <b>return</b> . It puts point in the next line with proper indentation, or if you type it inside the inline block of control construct, like <pre>foreach (@lines) {print; print}</pre> and you are on a boundary of a statement inside braces, it will transform the construct into a multiline and will place you into an appropriately indented blank line. <ul style="list-style-type: none"><li>Use C-j for usual 'newline-and-indent' behavior. See <b>'cperl-electric-linefeed'</b> documentation.</li></ul>			

Description	Keystroke	Function	Note
Insert matching parens <ul style="list-style-type: none"><li>toggle with C-c C-e</li></ul>	<ul style="list-style-type: none"><li>(</li><li>&lt;</li><li>[</li><li>{</li></ul>	(cperl-electric-paren ARG)	Insert an opening parenthesis or a matching pair of parentheses. <ul style="list-style-type: none"><li>Controlled by 'cperl-electric-parens' and 'cperl-hairy' user-options.</li></ul>
	<ul style="list-style-type: none"><li>)</li><li>]</li></ul>	(cperl-electric-rparen ARG)	Insert a matching pair of parentheses if marking is active. <ul style="list-style-type: none"><li>If not, or if we are not at the end of marking range, would self-insert.</li><li>Controlled by 'cperl-electric-parens' and 'cperl-hairy' user-options.</li></ul>
Insert : and indent	:	(cperl-electric-terminator ARG)	Insert character and correct line's indentation.
Insert ; and indent	;	(cperl-electric-semi ARG)	Insert character and correct line's indentation.
Insert { and indent	{	(cperl-electric-lbrace ARG &optional END)	Insert character, correct line's indentation, correct quoting by space.
Insert } and indent	}	(cperl-electric-brace ARG &optional ONLY-BEFORE)	Insert character and correct line's indentation. <ul style="list-style-type: none"><li>If ONLY-BEFORE and 'cperl-auto-newline', will insert newline before the place (even in empty line), but not after. If after ")" and the inserted char is "{", insert extra newline before only if 'cperl-extra-newline-before-brace'.</li></ul>
Deleted and possibly untabify	DEL	(cperl-electric-backspace ARG)	Backspace, or remove whitespace around the point inserted by an electric key. <ul style="list-style-type: none"><li>Will untabify if 'cperl-electric-backspace-untabify' is non-nil.</li></ul>
cperl-mode	PEL supports 2 cperl-mode implementations: Emacs' cperl-mode and  HaraldJoerg/cperl-mode (more complete) activated by the pel-use-perl user-option. <ul style="list-style-type: none"><li>The following cperl commands toggle some of its electric behaviour.</li></ul>		
toggle Perl auto-help	C-c C-h a	(cperl-toggle-autohelp)	Toggle the state of Auto-Help on Perl constructs (put in the message area). <ul style="list-style-type: none"><li>Delay of auto-help controlled by 'cperl-lazy-help-time':  By default it is nil, which (like null) prevents activation of the auto-help. To activate it, set cperl-lazy-help-time to an integer value.</li></ul>
Toggle auto-newline	C-c C-a	(cperl-toggle-auto-newline)	Toggle the state of 'cperl-auto-newline'.
Toggle electric mode	C-c C-e	(cperl-toggle-electric)	Toggle the state of parentheses doubling in CPerl mode. When typing an opening parens character the closing one is automatically entered.
Toggle auto-fill mode	C-c C-f <ul style="list-style-type: none"><li>&lt;f11&gt; t f a</li><li>&lt;f11&gt; RET</li></ul>	(auto-fill-mode &optional ARG)	Toggle automatic line breaking (Auto Fill mode). <ul style="list-style-type: none"><li>With a prefix argument, enable Auto Fill mode if the prefix argument is positive, disable it otherwise.</li><li>When Auto Fill mode is enabled, inserting a space at a column beyond 'current-fill-column' automatically breaks the line at a previous space.</li></ul>
Toggle keyword expansion	C-c C-k	(cperl-toggle-abbrev)	Toggle the state of automatic keyword expansion in CPerl mode.
Toggle space fix	C-c C-w	(cperl-toggle-construct-fix)	Toggle whether 'indent-region'/'indent-sexp' fix whitespace too.
here-doc support	The following cperl-mode commands operate on <a href="#">Perl here documents</a> .		
Narrows to here-doc	C-c C-n	(cperl-narrow-to-here-doc &optional POS)	Narrows editing region to the HERE-DOC at POS. POS defaults to the point.
Spell-check here-docs	C-c C-d	(cperl-here-doc-spell)	Spell-check HERE-documents in the Perl buffer. <ul style="list-style-type: none"><li>If a region is highlighted, restricts to the region.</li></ul>
Spell-check POD documentation	C-c C-p	(cperl-pod-spell &optional DO-HERES)	Spell-check POD documentation. <ul style="list-style-type: none"><li>If invoked with prefix argument, will do HERE-DOCs instead.</li><li>If a region is highlighted, restricts to the region.</li></ul>
Verify & refactor Perl code	The following commands provide Perl verification and refactoring facilities.		
<ul style="list-style-type: none"><li>perltidy</li></ul>  currently requires a lperltidy to be present on local host event when processing aremote file.	The following 2 commands run <a href="#">perltidy</a> between areas of the Perl code in the current buffer and the tidied version of that code. All work is done inside Emacs buffers, no file is affected. After generating the tidied code the functions open an <a href="#">ediff session</a> to compare your code and the tidied code, allowing you to decide what to use. Requires  HaraldJoerg/cperl-mode activated by the pel-use-perl user-option.  The perl-tidy-ediff commands execute the perltidy identified by <a href="#">perl-tidy-command</a> user-option with options identified by the <a href="#">perl-tidy-ediff-args</a> user-option. <ul style="list-style-type: none"><li>See <a href="#">Perl::Tidy @metacpan</a> for the perltidy list of options, also <a href="#">Online PerlTidy</a> , which provides help on the various options categorized by feature set.</li></ul>		
Run perltidy on current buffer or region. Ediff changes.	<f12> T	(pel-perl-tidy-ediff)	Run perltidy on current buffer, than start an ediff session, comparing the original source with perltidy output. Error messages are saved in the "perl-tidy-errors" buffer. <ul style="list-style-type: none"><li>If an area is marked, run perltidy on the marked area only.</li></ul>
Run perltidy on current subroutine. Ediff changes.  See <a href="#">ℹ Narrowing</a>	<f12> t	(perl-tidy-ediff-sub)	Run the perltidy program on the subroutine that stats before point and start en ediff session to compare the original code with the tidied version. <ul style="list-style-type: none"><li>Error messages are saved in the "perl-tidy-errors" buffer.</li><li> The buffer is automatically narrowed to the current function.</li><li>When quitting the ediff session it remains narrowed. Use C-x n w to widen the buffer back.</li></ul>
<ul style="list-style-type: none"><li>perlcritic</li></ul>	Perform static code analysis with <a href="#">perlcritic</a> (which must be installed separately).		
Check code with perlcritic  <a href="#">ℹ Tramp</a> -aware : run remote host's <a href="#">perlcritic</a> on remote file.	<f12> c	(pel-perl-critic &optional VERBOSE)	Validate the Perl file visited in current buffer with <a href="#">perlcritic</a> . Report error if it's not installed. <ul style="list-style-type: none"><li>With optional VERBOSE prefix argument, print extra information:<ul style="list-style-type: none"><li>Full name of the Policy module that created the violation</li><li>Full diagnostic discussion of each Perl Best Practice (PBP) violation.</li></ul></li></ul> Show errors in compilation-mode buffer in a format that allows navigation.
<ul style="list-style-type: none"><li>Other refactoring</li></ul>			
Find/fix missing whitespace code	C-c C-b	(cperl-find-bad-style)	Find places in the buffer where insertion of a whitespace may help. <ul style="list-style-type: none"><li>Prompts user for insertion of spaces. Currently it is tuned to C and Perl syntax.</li></ul>
Refactor: if (A) {B}' → 'B if A;'	C-c C-t	(cperl-invert-if-unless)	Change 'if (A) {B}' into 'B if A;' etc (or visa versa) if possible. <ul style="list-style-type: none"><li>If the cursor is not on the leading keyword of the BLOCK flavor of construct, will assume it is the STATEMENT flavor, so will try to find the appropriate statement modifier.</li></ul>
Lining up Perl Code : <a href="#">ℹ Align</a>	It's often best to <b>line up Perl code vertically</b> , arranging elements as tables, to help reader understand the code intent visually. <ul style="list-style-type: none"><li>This technique is also promoted in <a href="#">Damian Conway's Perl Best Practice book</a>, <a href="#">Vertical Alignment</a> section of chapter 2. See <a href="#">pel-align-words-vertically</a> in <a href="#">ℹ Align</a>.</li></ul>		
Lineup	<ul style="list-style-type: none"><li>C-M- </li><li>&lt;f12&gt;  </li></ul>	(cperl-lineup BEG END &optional STEP MINSHIFT)	Lineup construction in a region. <ul style="list-style-type: none"><li>Beginning of region should be at the start of a construction.</li><li>All first occurrences of this construction in the lines that are partially contained in the region are lined up at the same column.</li><li>MINSHIFT is the minimal amount of space to insert before the construction.</li><li>STEP is the tabwidth to position constructions.</li><li>If STEP is nil, 'cperl-lineup-step' will be used (or 'cperl-indent-level', if 'cperl-lineup-step' is nil).</li><li>Will not move the position at the start to the left.</li></ul>
Testing Perl code	The following command provides a way to test Perl code locally. See <a href="#">Perl 5 Syntax</a> for web-sites that provide a Perl interpreter.		
Start Perl REPL  See: <a href="#">ℹ start Shells/REPLs</a>	<f11> z r P  <f12> z	(perl-repl)	Run a Perl REPL in a "Perl-REPL" buffer. <ul style="list-style-type: none"><li> Requires the <a href="#">perl-repl</a> external package  activated by <a href="#">perl-use-perl-repl</a> user-option.</li><li>The <a href="#">perl-repl-file-path</a> user option specifies the name of the Perl REPL program, which may optionally specify the explicit file path.</li><li>PEL provides the <a href="#">perl-repl</a> shell script which uses the Perl command line.</li></ul>
Perl Live Coding ★★	 <a href="#">perl-live-coding</a> external package supported  when <a href="#">pel-use-pel-live-coding</a> user-option is set. The "perl-live" buffer provides a much more powerful Perl interpreter.		
Run perl code in *perl-live*	<f12> 1	(pel-perl-live-run)	Start or open an existing perl-live session buffer. Invokes perl-live-run and open the "perl-live" buffer automatically, a <a href="#">comint</a> mode buffer.  The <b>&lt;f12&gt; 1</b> key binding is always bound in a perl-mode buffer. This is not the case for the following key bindings that become accessible once perl-live-run was first executed (via this command or via M-x <a href="#">perl-live-run</a> )
The *perl-live* buffer operates in comint-mode which provides a set of commands to navigate across lines and history, as well as other. Type <b>&lt;f1&gt; m</b> inside the "perl-live" buffer to see these commands.	C-c C-1	(perl-live-run)	Run perl-live <a href="#">comint</a> session. Once this is done you can also use the "perl live" buffer as a Perl interpreter which runs in <a href="#">comint</a> -mode.
Stop	C-c C-p	(perl-live-stop)	Stop perl-live session. The "perl-live" buffer gets disconnected from the Perl process. <ul style="list-style-type: none"><li>It can be restarted (without closing the buffer) with a new execution of <a href="#">perl-live-run</a>.</li></ul>
Evaluate Perl code line/region	C-c C-c	(perl-live-eval-region-or-line)	Evaluate a line or region of Perl code. <ul style="list-style-type: none"><li>When evaluating a single line, point moves to the next line.</li></ul>
Evaluate Perl code in block	C-M-x	(perl-live-eval-sexp)	Evaluate Perl code between braces

Emacs & Perl — References

Document	Notes
Also see: <a href="#">Perl 5 Syntax</a>	
<a href="#">Perl @ Wikipedia</a>	
<a href="#">perl.org</a>	
<a href="#">Learn Perl @ perl.org</a> • <a href="#">Perl Style Guide</a>	<ul style="list-style-type: none"><li>• <a href="#">Perl Tutorial</a> - a gentle introduction to Perl with several examples over a browsable set of pages.</li><li>• <a href="#">Perl Intro</a> - a quick introduction to Perl</li><li>• <a href="#">Online Perl books</a><ul style="list-style-type: none"><li>• <a href="#">Beginning Perl</a></li></ul></li></ul>
<a href="#">Perl Reference Manuals</a>	<ul style="list-style-type: none"><li>• <a href="#">Perl Keywords</a>.</li><li>• <a href="#">Perl Operators</a>. Also see the <a href="#">Perl ABC Operator</a> page: organizes the information in sections (but has some markup typos).</li><li>• <a href="#">Perl Functions</a></li></ul>
Is Perl still relevant? Most probably.	<ul style="list-style-type: none"><li>• <a href="#">What makes Perl relevant in 2022? @ Stackoverflow blog</a>.</li><li>• <a href="#">Perl is dying quick. Could be extinct by 2023. The HFT Guy, 2019.</a> (which includes several invalid points).<ul style="list-style-type: none"><li>• My point is that Perl was popular, there's a lot of Perl code still being used and it's worth knowing and being able to write and edit Perl code (which was certainly not the first programming language as state by the article). But anyway, the post represents a point of view (and has many people commenting on it).</li></ul></li><li>• <a href="#">Perl is making a comeback: 5 reasons why it's worth learning. Posted January 6, 2023 by Lucas Rees.</a></li><li>• <a href="#">Why Perl Didn't Win on outspeaking.com, updated December 21, 2020</a></li></ul>
Perl and Secure Coding Practice and Tools	<ul style="list-style-type: none"><li>• <a href="#">Security Issues in Perl Scripts - by Jordan Dimov</a> - Discussion of misused and overlooked features of Perl from the security point of view.</li><li>• <a href="#">SEI CERT Perl Coding Standard @ Carnegie Mellon University</a><ul style="list-style-type: none"><li>• <a href="#">The CERT Perl Secure Coding Standard, by David Svoboda . June 25, 2012</a></li></ul></li><li>• <a href="#">perlsec</a> describes Perl security.</li></ul> <b>Tools:</b> <ul style="list-style-type: none"><li>• The <b><a href="#">perlcritic script</a></b> uses <a href="#">Perl::Critic</a> to scan Perl code and provides some listing advices.<ul style="list-style-type: none"><li>• On some Linux distros you can install it with: <code>sudo dnf install perl-Perl-Critic</code></li></ul></li><li>• The <b><a href="#">perltidy</a></b> application reformats Perl code to the promoted format.</li><li>• The <b><a href="#">zarn</a></b> static analyzer, hosted on Github, is another static analyzer for Perl. (Quite immature as of Oct. 2024).</li></ul>
Perl File name extensions	<ul style="list-style-type: none"><li>• <b><a href="#">.pl</a></b> Perl non executable libraries, also used for Perl scripts (but a file with no extension and a shebang line is also fine, and preferable, allowing the invocation of the script without having to type the '.pl' ).</li><li>• <b><a href="#">.pm</a></b> Perl modules.</li><li>• <b><a href="#">.plx</a></b> Used by Active State implementation. Identifies executable Perl scripts. Also used elsewhere to distinguish from Prolog (which also uses the .pl file extension).</li><li>• <b><a href="#">.pls</a></b></li><li>• <b><a href="#">.xs</a></b></li><li>• <b><a href="#">.t</a></b></li><li>• <b><a href="#">.pod</a></b> <b><a href="#">Plain Old Documentation</a></b> files, a lightweight markup language used to document Perl code.<ul style="list-style-type: none"><li>• See also <b><a href="#">perlpod</a></b>.</li></ul></li><li>• <b><a href="#">.ph</a></b></li><li>• <b><a href="#">.cgi</a></b></li><li>• <b><a href="#">.psgi</a></b></li></ul>
Perl programs	<ul style="list-style-type: none"><li>• <a href="#">Perl command line options</a></li></ul>
• <a href="#">perl</a>	Perl language interpreter
• <a href="#">perlbug</a> / <a href="#">perlthanks</a>	Describes how to submit bug report on Perl
• <a href="#">perldoc</a>	Print Perl Documentation, looking it up in the .pod format embedded in perl installation. Support following options: <ul style="list-style-type: none"><li>• -f : <a href="#">built-in functions</a></li><li>• -q : FAQ keyword search</li><li>• -v : <a href="#">variable</a></li><li>• -a perl API</li></ul>
• <a href="#">perlivp</a>	<ul style="list-style-type: none"><li>• Perl Installation Verification Procedure : checks Perl installation.</li><li>• Part of <b><a href="#">perl-level</a></b> package.</li></ul>
• <a href="#">perltidy</a>	<ul style="list-style-type: none"><li>• <a href="#">PeriTidy @ Wikipedia</a> , <a href="#">PeriTidy Home Pages: @SourceForge @GitHub.</a> <a href="#">Perl::Tidy GitHub repo</a>, <b><a href="#">Perl::Tidy @meta::cpan</a></b></li><li>• <a href="#">Online PeriTidy</a> , which provides help on the various options categorized by feature set.</li></ul>
<a href="#">perlsec - Perl security</a>	
Perl Community	Perl has a long history and is quite vast. Here's a collection of links to various web sites that can provide information about it. It is far from complete and collected without much background on what happened in lots of cases and no opinion yet taken on most of this. <ul style="list-style-type: none"><li>• <a href="#">strictures vs Schmorp common::sense</a>, <a href="#">Marc A.Lehman common::sense package</a>.</li></ul>