






















## Emacs support for Unix Shell Scripting

Description	Keystroke	Function	Note
<b>UNIX-like Shell Script Editing</b> See: <ul style="list-style-type: none"> <li><a href="#">comparison of command shells</a></li> <li><a href="#">ShellCheck Wiki</a></li> <li><a href="#">ShellCheck on-line</a></li> </ul> <ul style="list-style-type: none"> <li><b>PEL sh support activation</b> </li> <li>Activate sh-mode on files </li>   <li>Make script executable </li> <li>Distinguish script from sourced scripts </li> <li>Script extensions </li> <li><a href="#">ℹ Indentation</a> control </li> <li>shellcheck syntax check </li>   <li>Specialized templates </li> <li>superword-mode on </li> </ul>	Emacs provides the built-in <a href="#">sh-mode</a> to support UNIX-style shell script programming. <ul style="list-style-type: none"> <li>It supports several shell variants including:               <ul style="list-style-type: none"> <li><a href="#">bash</a> - see <a href="#">Bash Reference Manual</a></li> <li><a href="#">csh</a> - see <a href="#">An Introduction to C shell</a> , <a href="#">csh OpenBSD man page</a>, <a href="#">csh NetBSD Man page</a>.</li> <li><a href="#">ksh</a>.</li> <li><a href="#">sh</a>, the Bourne shell</li> <li><a href="#">zsh</a> - see <a href="#">zsh Manual</a> and <a href="#">The Z Shell</a> page</li> </ul> </li> <li>Several other shell types are supported . Use the <a href="#">sh-set-shell</a> command to force the use of a specific shell type, with <b>C-c :</b></li> </ul> PEL activates Unix shell-script support with the  <a href="#">pel-use-sh</a> user-options. <ul style="list-style-type: none"> <li>When <a href="#">pel-use-sh</a> on: the <a href="#">&lt;f11&gt; SPC Z &lt;f1&gt;</a> prefix is made available. In a shell script buffer these commands are accessible via the <a href="#">&lt;f12&gt;</a> key. The <a href="#">auto-mode-alist</a> user-option identifies path patterns files that must use the <a href="#">sh-mode</a> or <a href="#">shell-script-mode</a> (which is an alias for sh-mode).</li> <li><a href="#">pel-auto-mode-alist</a>: identifies extra entries that PEL automatically adds to the auto-mode-alist.               <ul style="list-style-type: none"> <li>Add <code>/bin/[^.]+\`</code> to sh-mode to automatically activate sh-mode for your shell scripts stored inside your <code>~/bin</code> directory.</li> </ul> </li> <li>PEL also activate extra minor modes in shell-script-mode through the PEL <a href="#">pel-sh-activates-minor-modes</a> user-option.</li> </ul> <ul style="list-style-type: none"> <li><a href="#">pel-make-script-executable</a> : when turned on (set to t), Emacs makes the saved shell script file executable.</li> <li>PEL provides the ability to automatically identify shell scripts that must be sourced and are therefore not executables:               <ul style="list-style-type: none"> <li><a href="#">pel-shell-sourced-script-file-name-prefix</a>: use a regexp to identify the base name of files that are meant to be sourced. For example, if all shell files that are sourced have a file name that begins with an underscore, use the following regexp: <code>\`_</code> <ul style="list-style-type: none"> <li><a href="#">pel-shell-script-extensions</a>: identifies file extensions of files that PEL must <b>not</b> identify as sourced files.</li> </ul> </li> </ul> </li> <li>Use of hard tab for indentation is set by <a href="#">pel-sh-use-tabs</a>. The number of columns used for indentation is controlled by <a href="#">pel-sh-tab-width</a>.</li> <li>Set <a href="#">pel-use-shellcheck</a> to activate shellcheck-based syntax checking. Values allow activating flycheck or flymake manually or automatically. Recommendation: select 'use flycheck automatically': it will activate it and will provide key bindings automatically.</li> <li>PEL also provide specialized code templates that are taking the above user-options into account. The commands distinguish a shell script file that must be executable from one that must be sourced and generates different text.</li> <li>PEL activates the <a href="#">superword-mode</a> automatically in shell script buffers. See <a href="#">ℹ Text Modes</a> for more info.</li> </ul>		
<b>Open this PDF file.</b> See also: <a href="#">ℹ Help/Info</a>	<a href="#">&lt;f11&gt; SPC Z &lt;f1&gt;</a> <a href="#">&lt;f12&gt; &lt;f1&gt;</a>	<a href="#">(pel-help-pdf &amp;optional OPEN-WEB-PAGE)</a>	Open the <a href="#">ℹ - UNIX Shell</a> local PDF. If the prefix argument (like <b>C-u</b> or <b>M--</b> ) is used, then it opens the remote GitHub hosted raw PDF instead. If the <a href="#">pel-flip-help-pdf-arg</a> user-option is set it's the other way around.
<a href="#">ℹ Customize</a> PEL UNIX Shell support	<a href="#">&lt;f11&gt; SPC Z &lt;f2&gt;</a> <a href="#">&lt;f12&gt; &lt;f2&gt;</a>	<a href="#">(pel-customize-pel &amp;optional OTHER-WINDOW)</a>	Customize PEL UNIX Shell support. <ul style="list-style-type: none"> <li>If OTHER-WINDOW is non-nil (use <b>C-u</b>), display in another window.</li> </ul>
<a href="#">ℹ Customize</a> Emacs UNIX Shell support	<a href="#">&lt;f11&gt; SPC Z &lt;f3&gt;</a> <a href="#">&lt;f12&gt; &lt;f3&gt;</a>	<a href="#">(pel-customize-library &amp;optional OTHER-WINDOW)</a>	Customize Emacs UNIX Shell support: sh, sh-script, sh-indentation. <ul style="list-style-type: none"> <li>If OTHER-WINDOW is non-nil (use <b>C-u</b>), display in another window.</li> </ul>
<b>Specialized Execution</b>	The following commands can be used to change the scripting dialect and to execute a portion of the code in the buffer.		
<b>Set the buffer shell type</b>	<b>C-c :</b>	<a href="#">(sh-set-shell SHELL &amp;optional NO-QUERY-FLAG INSERT-FLAG)</a>	Set this buffer's shell to SHELL (a string). Prompts, support tab-completion. <ul style="list-style-type: none"> <li>When used interactively, insert the proper starting <code>#!</code>-line, and make the visited file executable via 'executable-set-magic', perhaps querying depending on the value of 'executable-query'.</li> <li>Calls the value of 'sh-set-shell-hook' if set.</li> <li>Shell script files can cause this function be called automatically when the file is visited by having a 'sh-shell' file-local variable whose value is the shell name (don't quote it).</li> </ul>
<b>Execute region in a sub-shell</b>	<b>C-M-x</b>	<a href="#">(sh-execute-region START END &amp;optional FLAG)</a>	Pass optional header and region to a subshell for noninteractive execution. <ul style="list-style-type: none"> <li>The working directory is that of the buffer, and only environment variables are already set which is why you can mark a header within the script.</li> <li>With a positive prefix ARG, instead of sending region, define header from beginning of buffer to point. With a negative prefix ARG, instead of sending region, clear header.</li> <li>Print result on the echo area if it fits, otherwise into the "Shell Command Output" buffer.</li> </ul>
<b>Syntax checking with shellcheck</b>	Emacs shell script buffer syntax checking is done by <a href="#">shellcheck</a> . It can be provided by the built-in <a href="#">flymake</a> or the <a href="#">flycheck</a> external package.  With PEL, the <a href="#">pel-use-shellcheck</a> user-option determines which one is supported, if any. Defaults to no support.		
<b><a href="#">Flycheck</a></b> <b>pel-use-shellcheck</b> := <ul style="list-style-type: none"> <li><a href="#">flycheck-manual</a></li> <li><a href="#">flycheck-automatic</a></li> </ul>	Flycheck is a minor mode for on-the-fly syntax checking.  The <a href="#">flycheck</a> external package  is activated by PEL when <a href="#">pel-use-shellcheck</a> is set to either flycheck-manual or flycheck-automatic. <ul style="list-style-type: none"> <li>It is also activated when the <a href="#">pel-use-flycheck</a> user-option is turned on when another major mode specific user-option requires it.</li> </ul>  Aside from the following 2 key bindings that PEL provides to toggle the flycheck mode, flycheck key prefix is <b>C-c !</b> as set by its <a href="#">flycheck-keymap-prefix</a> user-option. You can change it for a different key prefix.		
<b>Toggle flycheck mode for current buffer</b>	<a href="#">&lt;f11&gt; ! !</a>	<a href="#">(flycheck-mode &amp;optional ARG)</a>	Toggle flycheck minor-mode for the current buffer.
<b>Toggle flycheck mode for all buffers</b>	<a href="#">&lt;f11&gt; ! M-!</a>	<a href="#">(global-flycheck-mode &amp;optional ARG)</a>	Toggle Flycheck mode in all buffers. <ul style="list-style-type: none"> <li>Flycheck mode is enabled in all buffers where 'flycheck-mode-on-safe' would do it.</li> </ul>
<b>• <a href="#">Info about Flycheck</a></b>	The following extra key bindings are available when flycheck is active.		
<b>Open Flycheck manual</b>	<b>C-c ! i</b>	<a href="#">(flycheck-manual)</a>	Open the Flycheck manual.
<b>Display Flycheck version</b>	<b>C-c ! v</b>	<a href="#">(flycheck-version &amp;optional SHOW-VERSION)</a>	Get the Flycheck version as string. <ul style="list-style-type: none"> <li>If called interactively or if SHOW-VERSION is non-nil, show the version in the echo area and the messages buffer.</li> <li>The returned string includes both, the version from package.el and the library version, if both a present and different.</li> <li>If the version number could not be determined, signal an error, if called interactively, or if SHOW-VERSION is non-nil, otherwise just return nil.</li> </ul>
<b>• <a href="#">Flycheck setup</a></b>	The following extra key bindings are available when flycheck is active.		
<b>Display documentation about syntax checker</b>	<b>C-c ! ?</b>	<a href="#">(flycheck-describe-checker CHECKER)</a>	Display the documentation of CHECKER. <ul style="list-style-type: none"> <li>CHECKER is a checker symbol.</li> <li>Pop up a help buffer with the documentation of CHECKER.</li> </ul>
<b>Select Flycheck Checker for current buffer</b>	<b>C-c ! s</b>	<a href="#">(flycheck-select-checker CHECKER)</a>	Select <a href="#">CHECKER</a> for the current buffer. <ul style="list-style-type: none"> <li>CHECKER is a syntax checker symbol (see 'flycheck-checkers') or nil. In the former case, use CHECKER for the current buffer, otherwise deselect the current syntax checker (if any) and use automatic checker selection via 'flycheck-checkers'.</li> <li>If called interactively prompt for CHECKER. With prefix arg deselect the current syntax checker and enable automatic selection again.</li> <li>Set 'flycheck-checker' to CHECKER and automatically start a new syntax check if the syntax checker changed.</li> <li>CHECKER will be used, even if it is not contained in 'flycheck-checkers', or if it is disabled via 'flycheck-disabled-checkers'.</li> </ul>
<b>Verify Flycheck setup</b>	<b>C-c ! v</b>	<a href="#">(flycheck-verify-setup)</a>	Check whether Flycheck can be used in this buffer. <ul style="list-style-type: none"> <li>Display a new buffer listing all syntax checkers that could be applicable in the current buffer. For each syntax checkers, possible problems are shown.</li> </ul>
<b>Disable Flycheck checker</b>	<b>C-c ! x</b>	<a href="#">(flycheck-disable-checker CHECKER &amp;optional ENABLE)</a>	Interactively disable CHECKER for the current buffer. <ul style="list-style-type: none"> <li>Prompt for a syntax checker to disable, and add the syntax checker to the buffer-local value of 'flycheck-disabled-checkers'.</li> <li>With non-nil ENABLE or with prefix arg, prompt for a disabled syntax checker and re-enable it by removing it from the buffer-local value of 'flycheck-disabled-checkers'.</li> </ul>

Description	Keystroke	Function	Note
• <b>Flycheck buffer/file</b>	The following extra key bindings are available when flycheck is active.		
Syntax Check current buffer	<b>C-c ! c</b>	(flycheck-buffer)	Start checking syntax in the current buffer. <ul style="list-style-type: none"> <li>Get a syntax checker for the current buffer with ‘flycheck-get-checker-for-buffer’, and start it.</li> </ul>
Check syntax of current file	<b>C-c ! C-c</b>	(flycheck-compile CHECKER)	Run CHECKER via ‘compile’. <ul style="list-style-type: none"> <li>Prompt for a syntax checker to run.</li> <li>Instead of highlighting errors in the buffer, this command pops up a separate buffer with the entire output of the syntax checker tool, just like ‘compile’.</li> </ul>
• <b>Manage Errors</b>	The following extra key bindings are available when flycheck is active.		
Show error list for current buffer	<ul style="list-style-type: none"> <li><b>C-c ! l</b></li> <li><b>&lt;f12&gt; e</b></li> </ul>	(flycheck-list-errors)	Show the error list for the current buffer.
Display all errors at point	<b>C-c ! h</b>	(flycheck-display-error-at-point)	Display all the error messages at point.
Explain error at point	<ul style="list-style-type: none"> <li><b>C-c ! e</b></li> <li><b>&lt;f12&gt; /</b></li> </ul>	(flycheck-explain-error-at-point)	Display an explanation for the first explainable error at point. <ul style="list-style-type: none"> <li>In a shell script buffer this opens the <b>shellcheck wiki page</b> for the identified error.</li> </ul>
Copy errors	<b>C-c ! C-w</b>	(flycheck-copy-errors-as-kill POS &optional FORMATTER)	Copy each error at POS into kill ring, using FORMATTER. <ul style="list-style-type: none"> <li>FORMATTER is a function to turn an error into a string, defaulting to ‘flycheck-error-message’.</li> <li>Interactively, use ‘flycheck-error-format-message-and-id’ as FORMATTER with universal prefix arg, and ‘flycheck-error-id’ with normal prefix arg, i.e. copy the message and the ID with universal prefix arg, and only the id with normal prefix arg.</li> </ul>
Clear all errors	<b>C-c ! C</b>	(flycheck-clear &optional SHALL-INTERRUPT)	Clear all errors in the current buffer. <ul style="list-style-type: none"> <li>With prefix arg or SHALL-INTERRUPT non-nil, also interrupt the current syntax check.</li> </ul>
Move point to next error	<ul style="list-style-type: none"> <li><b>C-c ! n</b></li> <li><b>M-n</b></li> </ul>	(flycheck-next-error &optional N RESET)	Visit the N-th error from the current point. <ul style="list-style-type: none"> <li>N is the number of errors to advance by, where a negative N advances backwards. With non-nil RESET, advance from the beginning of the buffer, otherwise advance from the current position.</li> </ul>
Move point to prior error	<ul style="list-style-type: none"> <li><b>C-c ! p</b></li> <li><b>M-p</b></li> </ul>	(flycheck-previous-error &optional N)	Visit the N-th previous error. <ul style="list-style-type: none"> <li>If given, N specifies the number of errors to move backwards by.</li> <li>If N is negative, move forwards instead.</li> </ul>
<b>Specialized Navigation</b>	The following commands override normal key bindings and provide specialized navigation key bindings in shell scripts buffers.		
Go to beginning of command	<b>M-a</b>	(sh-beginning-of-command)	Move point to successive beginnings of commands.
Go to end of command	<b>M-e</b>	(sh-end-of-command)	Move point to successive ends of commands.
<b>Backward to beginning of block:</b> <ul style="list-style-type: none"> <li>if* ⇐</li> <li>for   while   until ⇐</li> <li>case ⇐</li> </ul>	<ul style="list-style-type: none"> <li><b>C-M-b</b></li> <li><b>C-M-&lt;left&gt;</b></li> <li><b>C-[ C-b</b></li> <li><b>Esc C-b</b></li> <li><b>Esc C-&lt;left&gt;</b></li> </ul>	(backward-sexp &optional ARG)	Move backward across one balanced expression (sexp). <ul style="list-style-type: none"> <li>With ARG, do it that many times. Negative arg -N means move forward across N balanced expressions. This command assumes point is not in a string or comment.</li> <li><b>C-M-b</b> : ▸ Shift marking is available in graphics mode, <b>not in terminal mode</b>.</li> <li><b>C-M-&lt;left&gt;</b> : ▸ Shift marking works with this command.</li> </ul>
<b>(block backward)</b> See also: <a href="#">Navigation</a>	<ul style="list-style-type: none"> <li>⚠ With PEL: if you want to use <b>Esc C-&lt;left&gt;</b> binding you must ensure that <b>pel-windmove-on-esc-cursor</b> user option is set to nil.</li> <li>❖ <b>C-M-&lt;left&gt;</b> does not work on Windows, but <b>H-&lt;left&gt;</b> works.</li> <li>🔊 Several Linux distros map <b>C-M-&lt;left&gt;</b> to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems-&gt;settings-&gt;keyboard-&gt;shortcuts to prevent it from using that key sequence.</li> </ul>		
<b>Forward to end of block:</b> <ul style="list-style-type: none"> <li>⇒ fi</li> <li>⇒ done</li> <li>⇒ esac</li> </ul>	<ul style="list-style-type: none"> <li><b>C-M-f</b></li> <li><b>C-M-&lt;right&gt;</b></li> <li><b>C-[ C-f</b></li> <li><b>Esc C-f</b></li> <li><b>Esc C-&lt;right&gt;</b></li> </ul>	(forward-sexp &optional ARG)	Move forward across one balanced expression (sexp). <ul style="list-style-type: none"> <li>With ARG, do it that many times. Negative arg -N means move backward across N balanced expressions. This command assumes point is not in a string or comment.</li> <li><b>C-M-f</b> : ▸ Shift marking is available in graphics mode, <b>not in terminal mode</b>.</li> <li><b>C-M-&lt;right&gt;</b> : ▸ Shift marking works with this command.</li> </ul>
<b>(block forward)</b> See also: <a href="#">Navigation</a>	<ul style="list-style-type: none"> <li>⚠ With PEL: if you want to use <b>Esc C-&lt;right&gt;</b> binding you must ensure that <b>pel-windmove-on-esc-cursor</b> user option is set to nil.</li> <li>❖ <b>C-M-&lt;right&gt;</b> does not work on Windows, but <b>H-&lt;right&gt;</b> does.</li> <li>🔊 Several Linux distros map <b>C-M-&lt;right&gt;</b> to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems-&gt;settings-&gt;keyboard-&gt;shortcuts to prevent it from using that key sequence.</li> </ul>		
<b>Using Flymake</b>  <b>pel-use-shellcheck</b> := <ul style="list-style-type: none"> <li>flymake-manual</li> <li>flymake-automatic</li> </ul>	You can also use Emacs built-in flymake to control shell-check based syntax checking. <ul style="list-style-type: none"> <li>⚠ Note, however, than using flymake does not provide as many commands as when you use flycheck (as described above).</li> <li>Several key bindings are not available when flymake is used.</li> <li>🔊 Flymake has several customizable variables, which some listed here: <ul style="list-style-type: none"> <li>The following customization variables determine the exact circumstances whereupon Flymake decides to initiate a check of the buffer:</li> <li><b>flymake-start-on-flymake-mode</b> : t to start checking when flymake-mode is started. <b>nil</b> to prevent check.</li> <li><b>flymake-no-changes-timeout</b> : time to wait after last change to start checking. Default = 0.5 seconds.</li> <li><b>flymake-start-syntax-check-on-newline</b> : t to check after insertion or removal of newline char from buffer. <b>nil</b> to prevent check.</li> </ul> </li> <li>The following variable control navigation to next or previous error: <ul style="list-style-type: none"> <li><b>flymake-wrap-around</b> : If non-nil, moving to errors wraps around buffer boundaries.</li> <li><b>flymake-diagnostic-types-alist</b> : Alist ((KEY . PROPS)*) of properties of Flymake diagnostic types. See Emacs documentation for more info.</li> </ul> </li> </ul>		
Toggle Flymake mode on/off	<b>M-x flymake-mode</b>	(flymake-mode &optional ARG)	Toggle Flymake mode on or off. <ul style="list-style-type: none"> <li>With a prefix argument ARG, enable Flymake mode if ARG is positive, and disable it otherwise.</li> <li>Flymake is an Emacs minor mode for on-the-fly syntax checking.</li> <li>Flymake collects diagnostic information from multiple sources, called backends, and visually annotates the buffer with the results.</li> </ul>
Go to next flymake diagnostic	<b>M-n</b>	(flymake-goto-next-error &optional N FILTER INTERACTIVE)	Move point to the next Flymake diagnostic. <ul style="list-style-type: none"> <li>With a prefix arg, skip any diagnostics with a severity less than ‘:warning’.</li> <li>Display the error message in the echo line.</li> </ul>
Go to previous flymake diagnostic	<b>M-p</b>	(flymake-goto-prev-error &optional N FILTER INTERACTIVE)	Move point to the previous Flymake diagnostic. <ul style="list-style-type: none"> <li>With a prefix arg, skip any diagnostics with a severity less than ‘:warning’.</li> <li>Display the error message in the echo line.</li> </ul>

Description	Keystroke	Function	Note
Comments	Insert a comment, comment or un-comment a region with <b>M-;</b>		
Toggle display of comments in buffer or active region See also: <a href="#">ℹ Comments</a>	<b>&lt;f11&gt; ; ;</b>	(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 <a href="#">pel-use-hide-comnt</a> user option is <a href="#">t</a> .
Specialized Insertion			
Double quote word at point	<b>&lt;f12&gt; “</b>	(pel-sh-double-quote-word)	Surround word at point or selected area with double quotes.
Singe quote word at point	<b>&lt;f12&gt; ’</b>	(pel-sh-single-quote-word)	Surround word at point or selected area with single quotes.
Backtickquote word at point	<b>&lt;f12&gt; `</b>	(pel-sh-backtick-quote-word)	Surround word at point or selected area with back-tick characters.
Generic code skeletons • <a href="#">tempo skeletons</a> 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. <ul style="list-style-type: none"> <li>Emacs provides the built-in skeleton mechanism and the <a href="#">tempo skeletons</a>. <ul style="list-style-type: none"> <li>PEL supports both. They are used a little bit differently. <ul style="list-style-type: none"> <li>PEL provides key bindings to the tempo skeletons: the generic code templates, accessible via the <b>&lt;f6&gt;</b> prefix key, and the language-specific code templates, accessible via the <b>&lt;f12&gt;</b> key prefix.</li> </ul> </li> </ul> PEL provides <a href="#">generic</a> tempo skeletons the handle UNIX shell script files. </li></ul>		
<a href="#">ℹ Customize</a> PEL Text Insertions control	<b>&lt;f6&gt; &lt;f2&gt;</b>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL generic tempo skeleton customization groups that control the format of the various skeletons including the generic skeleton used by the <b>&lt;f6&gt; h</b> key (se below). <ul style="list-style-type: none"> <li>If OTHER-WINDOW is non-nil (use <b>C-u</b>), display in other window.</li> </ul>
Insert generic file module header block — Language agnostic  After inserting the template, navigate though areas that must be filled with: <ul style="list-style-type: none"> <li>tempo-forward-mark: <b>C-c .</b></li> <li>tempo-backward-mark: <b>C-c ,</b></li> </ul>	<b>&lt;f6&gt; h</b>	(pel-generic-file-header)	Insert a file header block at the top of the file. Works only for buffer visiting a file.  The command key binding <b>&lt;f6&gt; h</b> is available only 1 second after Emacs has started.
	 Specify the format of the header via the user-options in the <b>pel-pkg-generic-code-style</b> customization group accessible via <b>&lt;f6&gt; &lt;f2&gt;</b> <ul style="list-style-type: none"> <li>Inside a <b>sh-mode</b> buffer, <b>&lt;f12&gt; &lt;f2&gt;</b> provides access to the following customization groups: <ul style="list-style-type: none"> <li><b>pel-pkg-for-sh</b> for the control of the template format and <b>pel-sh-script-skeleton-control</b> for sh-mode specific user-options.</li> </ul> </li> <li>The files that have no extensions are often used in Unix-like OS shell scripts.</li> <li>These files are also supported as Emacs can recognize them if they are stored in a <b>bin</b> directory.</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	<b>&lt;f6&gt; SPC</b>	(pel-tempo-mode &optional ARG)	Toggle PEL tempo mode on/off. PEL tempo mode activates <b>C-c .</b> and <b>C-c ,</b> , as well as to <b>C-c C-.</b> and <b>C-c C-,</b> 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 when Emacs runs 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 <b>&lt;f6&gt; h</b> .
Jump to next tempo mark	<ul style="list-style-type: none"> <li><b>C-c M-f</b></li> <li><b>C-c .</b></li> <li><b>C-c C-.</b></li> </ul>	(tempo-forward-mark)	Jump to the next mark in ‘tempo-back-mark-list’: the location where code must be updated inside the inserted skeleton. <ul style="list-style-type: none"> <li>These key key bindings are only available when pel-tempo-mode is active.</li> </ul>
Jump to previous tempo mark	<ul style="list-style-type: none"> <li><b>C-c M-b</b></li> <li><b>C-c ,</b></li> <li><b>C-c C-,</b></li> </ul>	(tempo-backward-mark)	Jump to the previous mark in ‘tempo-back-mark-list’: the location where code must be updated inside the inserted skeleton. <ul style="list-style-type: none"> <li>These key binding are only available when pel-tempo-mode is active.</li> </ul>
Shell statement Insertion	The sh-mode provides the following commands to insert shell scripts code elements with templates defined with the <a href="#">Emacs skeleton language</a> . All of these statement insertion command share the same extra description: <ul style="list-style-type: none"> <li>This is a skeleton command (see ‘skeleton-insert’).</li> <li>Normally the skeleton text is inserted at point, with nothing "inside".</li> <li>If there is a highlighted region, the skeleton text is wrapped around the region text.</li> <li>A prefix argument ARG says to wrap the skeleton around the next ARG words.</li> <li>A prefix argument of -1 says to wrap around region, even if not highlighted.</li> <li>A prefix argument of zero says to wrap around zero words---that is, nothing.</li> <li>This is a way of overriding the use of a highlighted region.</li> </ul>		
Insert a case/switch	<b>C-c C-c</b>	(sh-case &optional STR ARG)	Insert a case/switch statement.
Insert a for loop	<b>C-c C-f</b>	(sh-for &optional STR ARG)	Insert a for loop.
Insert function definition	<b>C-c (</b>	(sh-function &optional STR ARG)	Insert a function definition.
Insert a if statement	<ul style="list-style-type: none"> <li><b>C-c &lt;tab&gt;</b></li> <li><b>C-c C-i</b></li> </ul>	(sh-if &optional STR ARG)	Insert a if statement.
Insert an indexed loop from 1 to n.	<b>C-c C-l</b>	(sh-indexed-loop &optional STR ARG)	Insert an indexed loop from 1 to n.
Insert a getopt loop	<b>C-c C-o</b>	(sh-while-getopts &optional STR ARG)	Insert a while getopt loop. <ul style="list-style-type: none"> <li>Prompts for an options string which consists of letters for each recognized option followed by a colon ‘:’ if the option accepts an argument.</li> </ul>
Insert a repeat loop definition	<b>C-c C-r</b>	(sh-repeat &optional STR ARG)	Insert a repeat loop definition.
Insert a select statement	<b>C-c C-s</b>	(sh-select &optional STR ARG)	Insert a select statement.
Insert an until loop	<b>C-c C-u</b>	(sh-until &optional STR ARG)	Insert an until loop.
Insert a while loop	<b>C-c C-w</b>	(sh-while &optional STR ARG)	Insert a while loop.
Show indentation	<b>C-c ?</b>	(sh-show-indent ARG)	Show how the current line would be indented. <ul style="list-style-type: none"> <li>This tells you which variable, if any, controls the indentation of this line.</li> <li>If optional arg ARG is non-null (called interactively with a prefix), a pop up window describes this variable.</li> <li>If variable ‘sh-blink’ is non-nil then momentarily go to the line we are indenting relative to, if applicable.</li> </ul>
Set indentation for current line	<b>C-c =</b>	(sh-set-indent)	Set the indentation for the current line. If the current line is controlled by an indentation variable, prompt for a new value for it.
Learn indentation from current line	<b>C-c &lt;</b>	(sh-learn-line-indent ARG)	Learn how to indent a line as it currently is indented. <ul style="list-style-type: none"> <li>If there is an indentation variable which controls this line’s indentation, then set it to a value which would indent the line the way it presently is.</li> <li>If the value can be represented by one of the symbols then do so unless optional argument ARG (the prefix when interactive) is non-nil.</li> </ul>

Description	Keystroke	Function	Note
Learn indentation from buffer	C-c >	(sh-learn-buffer-indent &optional ARG)	<p>Learn how to indent the buffer the way it currently is.</p> <ul style="list-style-type: none"> <li>• If 'sh-use-smie' is non-nil, call 'smie-config-guess'. Otherwise, run the sh-script specific indent learning command, as described below.</li> <li>• Output in buffer "**indent*" shows any lines which have conflicting values of a variable, and the final value of all variables learned.</li> <li>• When called interactively, pop to this buffer automatically if there are any discrepancies.</li> <li>• If no prefix ARG is given, then variables are set to numbers.</li> <li>• If a prefix arg is given, then variables are set to symbols when applicable -- e.g. to symbol '+' if the value is that of the basic indent.</li> <li>• If a positive numerical prefix is given, then 'sh-basic-offset' is set to the prefix's numerical value.</li> <li>• Otherwise, sh-basic-offset may or may not be changed, according to the value of variable 'sh-learn-basic-offset'.</li> <li>• Abnormal hook 'sh-learned-buffer-hook' if non-nil is called when the function completes. The function is abnormal because it is called with an alist of variables learned.</li> </ul> <p>⚠ This command can often take a long time to run.</p>