















## Emacs support for Unix Shell Scripting

Description	Keystroke	Function	Note
<b>UNIX-like Shell Script Editing</b>  See: • <b>comparison of command shells</b> • <b>ShellCheck Wiki</b>  • PEL sh support Activation 	Emacs provides the built-in <b>sh-mode</b> to support UNIX-style shell script programming. • It supports several shell variants including: • <b>bash</b> - see <b>Bash Reference Manual</b> • <b>csh</b> - see <b>An Introduction to C shell</b> , <b>csh OpenBSD man page</b> , <b>csh NetBSD Man page</b> . • <b>ksh</b> . • <b>sh</b> , the Bourne shell • <b>zsh</b> - see <b>zsh Manual</b> Several other shel types are supported . Use the sh-set-shell command to force the use of a specific shell type, with <b>C-c</b> :		
• Make script executable  • Distinguish script from sourced scripts  • Script extensions  • <b>Indentation</b> control   • Specialized templates  • Superword mode on 	PEL activates Unix shell-script support with the  <b>pel-use-sh</b> user-options. • When it is turned on the <b>&lt;f11&gt; SPC H</b> prefix is made available. In a shell script buffer these commands are accessible via the <b>&lt;f12&gt;</b> key. • It also activates the ability to activate minor modes for the sh major mode through the PEL <b>pel-sh-activates-minor-modes</b> user-option.  PEL provides the following customizable user-options. From a UNIX shell file, the <b>&lt;f12&gt; &lt;f2&gt;</b> key sequence opens their customization group. • <b>pel-make-script-executable</b> : when turned on (set to t), Emacs makes the saved shell script file executable. • PEL provides the ability to automatically identify shell scripts that must be sourced and are therefore not executables: • <b>pel-shell-sourced-script-file-name-prefix</b> : 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> • <b>pel-shell-script-extensions</b> : identified file extensions that can be used to prevent PEL to recognize them as shell files to source. • Use of hard tab for indentation is set by <b>pel-sh-use-tabs</b> . The number of columns used for indentation is controlled by <b>pel-sh-tab-width</b> .  • 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. • PEL activates the <b>superword-mode</b> automatically in shell script buffers. See <b>Indentation</b> for more info.		
<b>Open this PDF file.</b> See also: <b>Indentation</b> <b>Help/Info</b>	<b>&lt;f11&gt; SPC H &lt;f1&gt;</b> <b>&lt;f12&gt; &lt;f1&gt;</b>	<b>(pel-help-pdf</b> &optional OPEN-WEB-PAGE)	Open the <b>git - UNIX Shell</b> 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 <b>pel-flip-help-pdf-arg</b> user-option is set it's the other way around.
<b>Customize</b> PEL UNIX Shell support	<b>&lt;f11&gt; SPC H &lt;f2&gt;</b> <b>&lt;f12&gt; &lt;f2&gt;</b>	<b>(pel-customize-pel</b> &optional OTHER-WINDOW)	Customize PEL UNIX Shell support. • If OTHER-WINDOW is non-nil (use <b>C-u</b> ), display in another window.
<b>Customize</b> Emacs UNIX Shell support	<b>&lt;f11&gt; SPC H &lt;f3&gt;</b> <b>&lt;f12&gt; &lt;f3&gt;</b>	<b>(pel-customize-library</b> &optional OTHER-WINDOW)	Customize Emacs UNIX Shell support: sh, sh-script, sh-indentation. • If OTHER-WINDOW is non-nil (use <b>C-u</b> ), display in another window.
<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>	<b>(sh-set-shell</b> SHELL &optional NO-QUERY-FLAG INSERT-FLAG)	Set this buffer's shell to SHELL (a string). Prompts, support tab-completion. • 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'. • Calls the value of 'sh-set-shell-hook' if set. • 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).
<b>Execute region in a sub-shell</b>	<b>C-M-x</b>	<b>(sh-execute-region</b> START END &optional FLAG)	Pass optional header and region to a subshell for noninteractive execution. • 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. • 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. • Print result on the echo area if it fits, otherwise into the "Shell Command Output" buffer.
<b>Syntax checking with shellcheck</b>	Emacs shell script buffer syntax checking is done by <b>shellcheck</b> . It can be provided by the built-in <b>flymake</b> or the <b>flycheck</b> external package.  With PEL, the <b>pel-use-shellcheck</b> user-option determines which one is supported, if any. Defaults to no support.		
<b>Using Flymake</b>  <b>pel-use-shellcheck</b> := • flymake-manual • flymake-automatic	Flymake performs these checks while the user is editing.  Flymake has several customizable variables, which some listed here: The following customization variables determine the exact circumstances whereupon Flymake decides to initiate a check of the buffer: • <b>flymake-start-on-flymake-mode</b> : t to start checking when flymake-mode is started. nil to prevent check. • <b>flymake-no-changes-timeout</b> : time to wait after last change to start checking. Default = 0.5 seconds. • <b>flymake-start-syntax-check-on-newline</b> : t to check after insertion or removal of newline char from buffer. nil to prevent check.  The following variable control navigation to next or previous error: • <b>flymake-wrap-around</b> : If non-nil, moving to errors wraps around buffer boundaries. • <b>flymake-diagnostic-types-alist</b> : Alist ((KEY . PROPS)*) of properties of Flymake diagnostic types. See Emacs documentation for more info.		
<b>Toggle Flymake mode on/off</b>	<b>&lt;f12&gt; !</b>	<b>(flymake-mode</b> &optional ARG)	Toggle Flymake mode on or off. • With prefix argument ARG, enable Flymake mode if ARG is positive, disable it otherwise. • Flymake is an Emacs minor mode for on-the-fly syntax checking. • Flymake collects diagnostic information from multiple sources, called backends, and visually annotates the buffer with the results.
<b>Go to next flymake diagnostic</b>	<b>M-n</b>	<b>(flymake-goto-next-error</b> &optional N FILTER INTERACTIVE)	Move point to the next Flymake diagnostic. • With a prefix arg, skip any diagnostics with a severity less than 'warning'. • Display the error message in the echo line.
<b>Go to previous flymake diagnostic</b>	<b>M-p</b>	<b>(flymake-goto-prev-error</b> &optional N FILTER INTERACTIVE)	Move point to the previous Flymake diagnostic. • With a prefix arg, skip any diagnostics with a severity less than 'warning'. • Display the error message in the echo line.
<b>Flycheck</b>  <b>pel-use-shellcheck</b> := • flycheck-manual • flycheck-automatic	Flycheck is a minor mode for on-the-fly syntax checking.  The <b>flycheck</b> external package  is activated by PEL when <b>pel-use-shellcheck</b> is set to either flycheck-manual or flycheck-automatic. • It is also activated when the <b>pel-use-flycheck</b> user-option is turned on when another major mode specific user-option requires it.  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 <b>flycheck-keymap-prefix</b> user-option. You can change it for a different key prefix.		
<b>Toggle flycheck mode for current buffer</b>	<b>&lt;f11&gt; ! !</b>	<b>(flycheck-mode</b> &optional ARG)	Toggle flycheck minor-mode for the current buffer.
<b>Toggle flycheck mode for all buffers</b>	<b>&lt;f11&gt; ! M-!</b>	<b>(global-flycheck-mode</b> &optional ARG)	Toggle Flycheck mode in all buffers. • Flycheck mode is enabled in all buffers where 'flycheck-mode-on-safe' would do it.
<b>• Info about Flycheck</b>			
<b>Open Flycheck manual</b>	<b>C-c ! i</b>	<b>(flycheck-manual)</b>	Open the Flycheck manual.
<b>Display Flycheck version</b>	<b>C-c ! v</b>	<b>(flycheck-version</b> &optional SHOW-VERSION)	Get the Flycheck version as string. • If called interactively or if SHOW-VERSION is non-nil, show the version in the echo area and the messages buffer. • The returned string includes both, the version from package.el and the library version, if both a present and different. • 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.

Description	Keystroke	Function	Note
• <b>Flycheck setup</b>			
Display documentation about syntax checker	<b>C-c ! ?</b>	(flycheck-describe-checker CHECKER)	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>
Select Flycheck Checker for current buffer	<b>C-c ! s</b>	(flycheck-select-checker CHECKER)	Select CHECKER 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>
Verify Flycheck setup	<b>C-c ! v</b>	(flycheck-verify-setup)	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>
Disable Flycheck checker	<b>C-c ! x</b>	(flycheck-disable-checker CHECKER &optional ENABLE)	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>
• <b>Flycheck buffer/file</b>			
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>			
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>Specialized Insertion</b>			
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.
<b>Generic code skeletons</b> • <b>tempo skeletons</b> See also: <ul style="list-style-type: none"> <li><b>Inserting Text</b></li> <li><b>T Templates</b></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 <b>tempo skeletons</b>. <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 <b>generic</b> tempo skeletons the handle UNIX shell script files. </li></ul>		
<b>Customize PEL Text Insertions control</b>	<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>
<b>Insert generic file module header block — Language agnostic</b>  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. <div>⚠ The command key binding <b>&lt;f6&gt; h</b> is available only 1 second after Emacs has started.</div>
	<div>👉 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> </div> <div>👉 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.</div>		
<b>Toggle pel-tempo-mode</b>	<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. <div>👉 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>.</div>

Description	Keystroke	Function	Note
Jump to next tempo mark	<ul style="list-style-type: none"> <li>C-c M-f</li> <li>C-c .</li> <li>C-c C-.</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>C-c M-b</li> <li>C-c ,</li> <li>C-c C-,</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	<p>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:</p> <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	C-c C-c	(sh-case &optional STR ARG)	Insert a case/switch statement.
Insert a for loop	C-c C-f	(sh-for &optional STR ARG)	Insert a for loop.
Insert function definition	C-c (	(sh-function &optional STR ARG)	Insert a function definition.
Insert a if statement	<ul style="list-style-type: none"> <li>C-c &lt;tab&gt;</li> <li>C-c C-i</li> </ul>	(sh-if &optional STR ARG)	Insert a if statement.
Insert an indexed loop from 1 to n.	C-c C-1	(sh-indexed-loop &optional STR ARG)	Insert an indexed loop from 1 to n.
Insert a getopt loop	C-c C-o	(sh-while-getopts &optional STR ARG)	Insert a while getopt's 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	C-c C-r	(sh-repeat &optional STR ARG)	Insert a repeat loop definition.
Insert a select statement	C-c C-s	(sh-select &optional STR ARG)	Insert a select statement.
Insert an until loop	C-c C-u	(sh-until &optional STR ARG)	Insert an until loop.
Insert a while loop	C-c C-w	(sh-while &optional STR ARG)	Insert a while loop.
Show indentation	C-c ?	(sh-show-indent ARG)	<p>Show how the current line would be indented.</p> <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-nil (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	C-c =	(sh-set-indent)	<p>Set the indentation for the current line.</p> <p>If the current line is controlled by an indentation variable, prompt for a new value for it.</p>
Learn indentation from current line	C-c <	(sh-learn-line-indent ARG)	<p>Learn how to indent a line as it currently is indented.</p> <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>
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>
Comments			
Toggle display of comments in buffer or active region See also: <a href="#">☞ Comments</a>	<f11> ; ;	(hide/show-comments-toggle &optional START END)	<p>Toggle hiding/showing of comments in the active region or whole buffer.</p> <ul style="list-style-type: none"> <li>If the region is active then toggle in the region. Otherwise, in the whole buffer.</li> </ul> <p>📦 This requires the <a href="#">hide-comnt.el</a> package (see <a href="#">☞ Comments</a>). <a href="#">🔗</a> PEL activates it when the <a href="#">pel-use-hide-comnt</a> user option is <a href="#">t</a>.</p>