













## Emacs support for Unix Shell Scripting

Description	Keystroke	Function	Note
<b>UNIX-like Shell Script Editing</b>  See: <b><u>comparison of command shells</u></b>  • PEL sh support Activation 	Emacs provides the built-in <b>sh-mode</b> 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><b>bash</b>,</li> <li><b>cs</b><b>h</b>,</li> <li><b>ksh</b>,</li> <li><b>sh</b>, the Bourne shell</li> <li><b>zsh</b></li> </ul> </li> <li>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> <b>:</b></li> </ul>		
• Make script executable  • Distinguish script from sourced scripts  • Script extensions  • <b>⌘ Indentation</b> control  • Specialized templates 	PEL activates Unix shell-script support with the  <b>pel-use-sh</b> user-options. <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>  PEL provides the following customizable user-options. From a UNIX shell file, the <b>&lt;f12&gt; &lt;f2&gt;</b> key sequence opens the customization group for them. <ul style="list-style-type: none"> <li><b>pel-make-script-executable</b> : 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><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></li> <li><b>pel-shell-script-extensions</b>: identified file extensions that can be used to prevent PEL to recognize them as shell files to source.</li> </ul> </li> <li>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>.</li> </ul> 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.		
<b>Open this PDF file.</b> See also: <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 &amp;optional OPEN-WEB-PAGE)</b>	Open the <b>⌘ - 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 &amp;optional OTHER-WINDOW)</b>	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>
<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 &amp;optional OTHER-WINDOW)</b>	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>Set the buffer shell type</b>	<b>C-c :</b>	<b>(sh-set-shell SHELL &amp;optional NO-QUERY-FLAG INSERT-FLAG)</b>	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>	<b>(sh-execute-region START END &amp;optional FLAG)</b>	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>Generic code skeletons</b> • <b><u>tempo skeletons</u></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> </li> </ul> PEL provides <b>generic</b> tempo skeletons the handle UNIX shell script files.		
<b>⌘ Customize</b> PEL Text Insertions control	<b>&lt;f6&gt; &lt;f2&gt;</b>	<b>(pel-customize-pel &amp;optional OTHER-WINDOW)</b>	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>	<b>(pel-generic-file-header)</b>	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 sh-mode buffer the <code>&lt;f12&gt; &lt;f2&gt;</code> provides access to the <b>pel-pkg-for-sh</b> customization group which provides important user-option for the control of the template format and is the parent of the <b>pel-sh-script-skeleton-control</b> customization group.</li> <li>The files that have no extensions are often used in Unix-like OS shell scripts. These files are also supported as Emacs can recognize them if they are stored in a <b>bin</b> directory. PEL also has special support for them and is controlled by the <b>pel-sh-script-skeleton-control</b> customization group, which is accessible as a child of the main group.</li> </ul>  After inserting the template you can use the tempo-forward-mark and tempo-backward-mark to move point to the beginning of each section that must be filled.		
<b>Toggle pel-tempo-mode</b>	<b>&lt;f6&gt; SPC</b>	<b>(pel-tempo-mode &amp;optional ARG)</b>	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 ( <b>#</b> ) 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> .
<b>Jump to next tempo mark</b>	<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>	<b>(tempo-forward-mark)</b>	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>
<b>Jump to previous tempo mark</b>	<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>	<b>(tempo-backward-mark)</b>	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>

Description	Keystroke	Function	Note
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 getopts loop	C-c C-o	(sh-while-getopts &optional STR ARG)	Insert a while getopts 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)	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	C-c =	(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	C-c <	(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>
Learn indentation from buffer	C-c >	(sh-learn-buffer-indent &optional ARG)	Learn how to indent the buffer the way it currently is. <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> ⚠️ This command can often take a long time to run.
Go to beginning of command	M-a	(sh-beginning-of-command)	Move point to successive beginnings of commands.
Go to end of command	M-e	(sh-end-of-command)	Move point to successive ends of commands.
Comments			
Toggle display of comments in buffer or active region See also: <a href="#">☞ Comments</a>	<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="#">☞ Comments</a> ). 🔄 PEL activates it when the <a href="#">pel-use-hide-comnt</a> user option is t.