🚧 Emacs support for Unix Shell Scripting 🚧

Description	Keystroke	Function	Note

UNIX-like Shell Script Editing
 	
See: **comparison of command shells**
	Emacs provides the built-in **sh-mode** to support UNIX-style shell script programming.
	• It supports several shell variants including:
	  • **bash**,
	  • **csh**,
	  • **ksh**,
	  • **sh**, the Bourne shell
	  • **zsh**
	  Several other shel types are supported . Use the sh-set-shell command to force the use of a specific shell type, with **C-c :**
	
	PEL activates Unix shell-script support with the **pel-use-sh** user-options.
	• When it is turned on the **<f11> SPC H** prefix is made available. In a shell script buffer these commands are accessible via the **<f12>** key.
	• It also activates the ability to activate minor modes for the sh major mode through the PEL **pel-sh-activates-minor-modes** user-option.
	🎛️ PEL provides the following customizable user-options. From a UNIX shell file, the **<f12> <f2>** key sequence opens the customization group for them.
	• **pel-make-script-executable** : 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:
	  • **pel-shell-sourced-script-file-name-prefix**: use a regexp to identify name of files that are meant to be sourced.
	  • **pel-shell-script-extensions**: identified file extensions that can be used to prevent PEL to recognize them as shell files to source.
	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.		

Open this PDF file. See also: **∑ Help/Info**	**<f11> SPC H <f1>** **<f12> <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **𝔓𝔈𝔏 - UNIX Shell** local PDF. If the prefix argument (like **C-u** or **M--**) is used, then it opens the remote GitHub hosted raw PDF instead. If the **pel-flip-help-pdf-arg** user-option is set it's the other way around.
∑ Customize PEL UNIX Shell support	**<f11> SPC H <f2>** **<f12> <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL UNIX Shell support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
∑ Customize Emacs UNIX Shell support	**<f11> SPC H <f3>** **<f12> <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs UNIX Shell support: sh, sh-script, sh-indentation. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Set the buffer shell type	**C-c :**	(**sh-set-shell** 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 #!-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).
Execute region in a sub-shell	**C-M-x**	(**sh-execute-region** 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.

Generic code skeletons
• **tempo skeletons**
See also:
• **∑ Inserting Text**
• **T Templates**
	Several mechanisms have been developed to allow easy insertion of predefined text in Emacs.
	• Emacs provides the built-in skeleton mechanism and the **tempo skeletons**.
	  • PEL supports both. They are used a little bit differently.
	  • PEL provides key bindings to the tempo skeletons: the generic code templates, accessible via the **<f6>** prefix key, and the language-specific code templates, accessible via the **<f12>** key prefix.
	
	PEL provides **generic** tempo skeletons the handle UNIX shell script files.		

∑ Customize PEL Text Insertions control	**<f6> <f2>**	(**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 **<f6> h** key (se below). • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Insert generic file module header block — Language agnostic After inserting the template, navigate though areas that must be filled with: • tempo-forward-mark: **C-c .** • tempo-backward-mark: **C-c ,**	**<f6> h**	(**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 **<f6> h** is available only 1 second after Emacs has started.
	☝️Specify the format of the header via the user-options in the **pel-pkg-generic-code-style** customization group accessible via **<f6> <f2>** • Inside a sh-mode buffer the <f12> <f2> provides access to the **pel-pkg-for-sh** customization group which provides important user-option for the control of the template format and is the parent of the **pel-sh-script-skeleton-control** customization group. • 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 **bin** directory. PEL also has special support for them and is controlled by the **pel-sh-script-skeleton-control** customization group, which is accessible as a child of the main group. ☝️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.		
Toggle pel-tempo-mode	**<f6> SPC**	(**pel-tempo-mode** &optional ARG)	Toggle PEL tempo mode on/off. 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 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 **<f6> h**.
Jump to next tempo mark	• **C-c M-f** • **C-c .** • **C-c C-.**	(**tempo-forward-mark**)	Jump to the next mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton. • These key key bindings are only available when pel-tempo-mode is active.
Jump to previous tempo mark	• **C-c M-b** • **C-c ,** • **C-c C-,**	(**tempo-backward-mark**)	Jump to the previous mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton. • These key binding are only available when pel-tempo-mode is active.
Shell statement Insertion	The sh-mode provides the following commands to insert shell scripts code elements with templates defined with the **Emacs skeleton language**. All of these statement insertion command share the same extra description: • This is a skeleton command (see 'skeleton-insert'). • Normally the skeleton text is inserted at point, with nothing "inside". • If there is a highlighted region, the skeleton text is wrapped around the region text. • A prefix argument ARG says to wrap the skeleton around the next ARG words. • A prefix argument of -1 says to wrap around region, even if not highlighted. • A prefix argument of zero says to wrap around zero words---that is, nothing. • This is a way of overriding the use of a highlighted region.		
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.

Description	Keystroke	Function	Note
Insert function definition	`C-c (`	(**sh-function** &optional STR ARG)	Insert a function definition.
Insert a if statement	• `C-c <tab>` • `C-c C-i`	(**sh-if** &optional STR ARG)	Insert a if statement.
Insert an indexed loop from 1 to n.	`C-c C-l`	(**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 getopts loop. • Prompts for an options string which consists of letters for each recognized option followed by a colon ':' if the option accepts an argument.
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. • This tells you which variable, if any, controls the indentation of this line. • If optional arg ARG is non-null (called interactively with a prefix), a pop up window describes this variable. • If variable 'sh-blink' is non-nil then momentarily go to the line we are indenting relative to, if applicable.
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. • 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. • 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.
Learn indentation from buffer	`C-c >`	(**sh-learn-buffer-indent** &optional ARG)	Learn how to indent the buffer the way it currently is. • If 'sh-use-smie' is non-nil, call 'smie-config-guess'. Otherwise, run the sh-script specific indent learning command, as described below. • Output in buffer "*indent*" shows any lines which have conflicting values of a variable, and the final value of all variables learned. • When called interactively, pop to this buffer automatically if there are any discrepancies. • If no prefix ARG is given, then variables are set to numbers. • 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. • If a positive numerical prefix is given, then 'sh-basic-offset' is set to the prefix's numerical value. • Otherwise, sh-basic-offset may or may not be changed, according to the value of variable 'sh-learn-basic-offset'. • 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. ⚠ 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: **∑ Comments**	`<f11> ; ;`	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see **∑ Comments**). PEL activates it when the **pel-use-hide-comnt** user option is **t**.