↖ Topic Index

Inside shell-mode

Description	Keystroke	Function	Note

shell-mode
Next page Topics: Scrolling, Shell Command History, Text Delete/kill, Job-control
	Emacs **shell-mode** is one of several terminal emulator modes provided by Emacs.
	• Also see:
	  • The 𝕀 **Shells** page for information on how to launch the various terminal shells.
	  • The 🚦 **Shells/Terminals Comparisons** which compares the features of these terminal shells.
	
	In **shell-mode**, Emacs gets all keys, so all Emacs key bindings are available; you do not need to switch the line mode as in the term-mode or the other shell modes. You can also navigate up and down inside the wheel window using the normal navigation keys. This does mean, however, that the terminal is not a normal terminal that does not behave as such. Some programs will not run properly in shell-mode.		

Open this PDF file. See also: 𝕀 **Help/Info**	`<f11> SPC z s <f1>` `<f12> <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the 𝕀 **Shells** 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.
𝕀 **Customize PEL shell management control**	`<f11> SPC z s <f2>` `<f12> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL shell support. • If the prefix argument is non-nil (like `C-u` or `M--`), display in other window.
𝕀 **Customize Emacs shell & term control**	`<f11> SPC z s <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs shell support: shell, comint. • If the prefix argument is non-nil (like `C-u` or `M--`), display in other window.
Show customization for shell-mode	`<f12> <f4> ?`	(**pel-shell-show-cfg** &optional APPEND)	Print shell-mode affecting configuration information in specialized buffer. • If the prefix argument is non-nil (like `C-u` or `M--`), append to previous report instead of clearing it. The buffer shows the list of relevant user option names and values. The names are links to their customization.
Describe active major/minor(s) modes and the key bindings	• `C-h m` • `<f1> m` • `<f11> ? k m`	(**describe-mode** &optional BUFFER)	Lists the active major mode, all active minor modes and the bound keystrokes. In various shell modes this is particularly useful. See 𝕀 **Help/Info** for more info.
Toggle shell command echo	`<f12> <f4> e`	(**pel-comint-toggle-shell-echoes**)	Toggle the **comint-process-echoes** user-option value for the current buffer. • Use this when you want to prevent shell from echoing the command in the current buffer and you do not want to modify the '**comint-process-echoes**' customized value. If you need to prevent the command echo in the shell permanently, set the **comint-process-echoes** user option to `t`. You can customize that user-option with `M-x customize-variable` (or the global PEL binding `<f11> <f2> o`).
Clear shell buffer	• `<f12> c` • `C-c M-o`	(**pel-comint-clear-buffer-and-get-prompt** @optional BUFFER-OR-NAME)	Clear the command interpreter buffer. Ensure the shell is ready to take input. • Useful in the *shell* buffer because the clear shell command does not work. ☝ The `C-c M-o` binding to the PEL function is local to the buffer. Bindings in buffers with the major modes remain attached to the Emacs built-in **comint-clear-buffer** command.
Send command	`RET`	(**comint-send-input** &optional NO-NEWLINE ARTIFICIAL)	Send input to process.
Tab completion	`TAB`	(**completion-at-point**)	Perform completion on the text around point. • The completion method is determined by '**completion-at-point-functions**'. • This uses the usual Emacs completion rules (see **Completion**), with the completion alternatives being file names, environment variable names, the shell command history, and history references (see **Shell History References**). For options controlling the completion, see **Shell Mode Options**.

Resync directories.
• Use to (re-)activate **tab completion** in shell.
 	
☛ Directory Tracking
	`<f12> r`	(**shell-resync-dirs**)	Resync the buffer's idea of the current directory stack.
			• Use this command to allow **tab completion** at the shell prompt if it does not work.
			  • This command queries the shell with the command bound to '**shell-dirstack-query**' (default "**dirs**"), reads the next line output and parses it to form the new directory stack.
			  • ⚠ Don't issue this command unless the buffer is idle, kat a shell prompt. If this hangs type `C-g` .
			  • Also, note that if some other subprocess decides to do output immediately after the query, its output will be taken as the new directory stack -- you lose. If this happens, just do the command again.

• Navigation	The **shell-mode** provides these mode specific navigation commands. Other global navigation commands are available and described in the 𝕀 **Navigation** page.		
Move point to previous prompt	`<f12> <up>`	(**pel-shell-previous-prompt N**)	Move point to the previous prompt line. Repeat N times. With negative N: reverse direction. • Use the **pel-shell-prompt-line-regexp** user-option to identify what to search. This places the point after the prompt at the beginning of the command.
Move point to next prompt	`<f12> <down>`	(**pel-shell-next-prompt N**)	Move point to the next prompt line. Repeat N times. With negative N: reverse direction. • Use the **pel-shell-prompt-line-regexp** user-option to identify what to search. This places the point after the prompt at the beginning of the command.
Move backward command	`C-c C-b`	(**shell-backward-command** &optional ARG)	Move backward across ARG shell command(s). Does not cross lines. • The variable **shell-command-regexp** specifies how to recognize the end of a command.
Move forward command	`C-c C-f`	(**shell-forward-command** &optional ARG)	Move forward across one shell command, but not beyond the current line. • The variable **shell-command-regexp** specifies how to recognize the end of a command.
Move in command (⚠ and anywhere else)	To navigate inside the command you can use any of the navigation keys. Some of them are very similar to what the various shells (sh, bash, zsh, etc..) provide since they normally support Emacs navigation keys. But the navigation is not restricted to the current command and point can move inside the prompt or above it.		

Move word forward	`C-<right>`	(**pel-forward-token-start** &optional N)	Move forward to the start of the next token.
			• ⚠ Point can move outside of command.
			A token being identified by:
			  • any word (with all characters allowed by syntax table)
			  • punctuation
			  • first character after whitespace.
			Move over whitespace but stop at comments, operators, punctuation.
			• Argument N is a numeric argument identifying the number of times the operation is done. If N is negative the move is reversed (and goes backward).

Move word backward	`C-<left>`	(**pel-backward-token-start** &optional N)	Move backward to the start of the previous token. • Argument N is a numeric argument identifying the number of times the operation is done. If N is negative the move is reversed (and goes forward).
Move point to beginning of line (after prompt)	`C-c C-a`	(**comint-bol-or-process-mark**)	Move point to beginning of line (after prompt) or to the process mark. • The first time you use this command, it moves to the beginning of the line (but after the prompt, if any). If you repeat it again immediately, it moves point to the process mark. • The process mark separates the process output, along with input already sent, from input that has not yet been sent. Ordinarily, the process mark is at the beginning of the current input line; but if you have used C-c SPC to send multiple lines at once, the process mark is at the beginning of the accumulated input.

Description	Keystroke	Function	Note
• **Scrolling**			
Scroll to beginning of last command output	• `C-c C-r` • `C-M-l`	(**comint-show-output**)	Scroll to display the beginning of the last command output at the top of the window; also move the cursor there. • Sets mark to the value of point when this command is run.
Scroll end of buffer to end of window	`C-c C-e`	(**comint-show-maximum-output**)	Put the end of the buffer at the bottom of the window.
• **Shell Command History**	To navigate shell history in a **shell-mode** buffer, you must use the following key bindings.		
Cycle shell command history backwards	• `C-<up>` • `M-p`	(**comint-previous-input** ARG)	Cycle backwards through input history, saving input.
Cycle shell command history forwards	• `C-<down>` • `M-n`	(**comint-next-input** ARG)	Cycle forwards through input history.
• **Text Delete/Kill**			
Delete	`C-d`	(**comint-delchar-or-maybe-eo**f ARG)	Delete ARG characters forward or send an EOF to subprocess. • Sends an EOF only if point is at the end of the buffer and there is no input.
Kill text up to point	`C-C C-u`	(**comint-kill-input**)	Kill all text from last stuff output by interpreter to point. • Essentially kill all text from the beginning of the command up to the point.
Kill previous word	• `C-c C-w` • `M-DEL`	(**backward-kill-word** ARG)	Kill characters backward until encountering the beginning of a word. With argument ARG, do this that many times.
Delete all previous output	`C-c C-o`	(**comint-delete-output** &optional KILL)	Delete all output from interpreter since last input. ☝ Useful if that output is not needed, or if you kill it to yank it somewhere else. • It does not delete the prompt: it deletes the output from the previous command as shown in the shell. • If KILL (interactively, the prefix), save the killed text in the kill ring.
• **Job control**	The following commands can be used to control the sub-job (sub-process) launched from the shell.		
Interrupt current subjob	`C-c C-c`	(**comint-interrupt-subjob**)	Interrupt the current subjob.
Stop current subjob	`C-c C-z`	(**comint-stop-subjob**)	Stop the current subjob. ⚠ ***WARNING***: if there is no current subjob, you can end up suspending the top-level process running in the buffer. If you accidentally do this, use `M-x comint-continue-subjob` to resume the process. (This is not a problem with most shells, since they ignore this signal.)
Quit subjob	`C-c C-\`	(**comint-quit-subjob**)	Send quit signal to the current subjob. • This command also kills any shell input pending in the shell buffer and not yet sent
• **Misc**			
Accumulate and send	`C-C SPC`	(**comint-accumulate**)	Accumulate a line to send as input along with more lines. • This inserts a newline so that you can enter more text to be sent along with this line. Use RET to send all the accumulated input, at once. • The entire accumulated text becomes one item in the input history when you send it.
Open file at point See: Σ **File-mngt**	Then result of some commands, like compilation, build, or other, may generate a message that identifies a file with line number and possibly column number. With. PEL, you can use the **pel-open-at-point** command to open the file at the specified location and the **pel-set-open-at-point-dir** to establish the root directory. These commands are described in the Σ File-mngt page. A partial copy of the information is placed here for convenience.		
Set base directory for pel-open-at-point relative file names See: Σ **File-mngt**	`<f11> f ;`	(**pel-set-open-at-point-dir**)	Set the behaviour of '**pel-open-at-point**' in **current buffer**. Which defaults to value selected by **pel-open-file-at-point-dir** user-option. • Select method used to determine the directory from which a relative file name is built from following methods: • Use visited file parent directory (the default). • Use buffer's current working directory. • Use a specified directory. Prompts for the directory name. Supports completion.
Open file or web-page whose name is at point ★★ See: Σ **File-mngt**	• `M-<f6>` • `<f11> f .` • `6y`	(**pel-open-at-point** &optional N)	Open the file, library or the URL, named at point, with potential line & column #s. • If necessary will search source code files in current project as specified by **pel-filename-at-point-finders** user-option. Type **<f12> <f4> ?** to **show used file search method** in supporting modes. 🎛 Supports glob characters, partial directory path. When multiple files are found it prompts using the method selected by **pel-prompt-read-method** user-option. 📦 The **6y** key-chord is available if **pel-use-key-chord** is non-nil. See Σ **Key-Chords**.