


















Description	Keystroke	Function	Note
Open a <u>shell</u>	<f11> z s	(shell &optional BUFFER)	👉 To open a shell instance inside another window: use the <b>C-u</b> prefix
	<b>Implementation</b> The oldest emacs shell. Uses the comint-mode. Emacs keys are possible, the sub-process does not see the keys until <RET> is pressed. <b>Supports</b> <ul style="list-style-type: none"> <li>Can run multiple shell, each inside its own buffer/name</li> <li>Cursor lateral cursor line beginning/end, kill, yank.</li> <li>Meta-cursor word-move keys.</li> <li>bash alias</li> <li>Command history (but with Control Up/Down)</li> <li>Can run Python scripts. Can run Python REPL, REPL is OK, echo is OK, no Python colouring, but each command is colored.</li> <li>Can run Common-Lisp (clisp) REPL</li> </ul> <b>Limitations:</b> <ul style="list-style-type: none"> <li>Clear screen does not work.</li> <li>ls colouring does not work, ls columns are misaligned.</li> <li>Can start and stop top, but the output is incorrect and cannot be read.</li> <li>The shell PS1 prompt is partially applied, remnants show up on the second line. <b>TO-INVESTIGATE?</b>.</li> </ul>		
Open an <u>ANSI term</u> shell	<f11> z a	(ansi-term PROGRAM &optional NEW-BUFFER-NAME)	⚠️ Normally <b>operates in character mode</b> , in which up/down navigation and kill/yank is not possible. Change to line mode to do that: <ul style="list-style-type: none"> <li>Use <b>C-x C-j</b> to change to line mode an allow movement, mark, saving.</li> <li>When done use <b>C-c C-k</b> to switch to character mode.</li> </ul>
	<b>Implementation</b> <ul style="list-style-type: none"> <li>Prompts for shell to use. Default is /bin/bash. Can use others. Opens in current window.</li> <li>A terminal emulator written in Emacs Lisp.</li> <li>Newer implementation than term.</li> <li>You can even run other editors within it (vi, emacs, others). But use character-mode.</li> </ul> <b>Specificities:</b> <ul style="list-style-type: none"> <li>C-x is mapped to term-escape-char</li> </ul> <b>Supports:</b> <ul style="list-style-type: none"> <li>Scroll up/down with M-&lt;up&gt;, M-&lt;down&gt;</li> <li>ls colouring, columns are aligned</li> <li>bash alias</li> <li>bash tab expansion</li> <li>command line redirection</li> <li>clear screen</li> <li>Command history</li> <li>Can run Python scripts.</li> <li>Running Python shell:               <ul style="list-style-type: none"> <li>REPL is OK, echo is OK</li> </ul> </li> </ul> <b>Limitations:</b> <ul style="list-style-type: none"> <li>Natively runs in character mode, which does not allow movement nor saving.</li> <li>&lt;up&gt;, &lt;down&gt; cursor, C-n/C-p do not work as navigating: used as shell command history. Change to line mode (see above) to enable these.</li> <li>Not yet found a way to control prompt (PS1 setup of .bash_profile does not seem to be used).💩</li> </ul>		
Open a <u>term</u> shell	<f11> z t	(term PROGRAM)	Prompts for shell to use. Default is /bin/bash. Can use others. Opens in current window.
	<b>Implementation:</b> Shell implemented in Emacs Lisp. The keys are sent directly to the sub-process, which means they are not interpreted by Emacs. Same access as normal shell: can use the bash alias, tab-autocomplete, clear screen, can use less and indirection, can execute python scripts. Can even run other terminal editors like vim, synaptic, etc... <b>Supports</b> <ul style="list-style-type: none"> <li>Cursor lateral cursor line beginning/end, kill, yank.</li> <li>Meta-cursor keys, but only in terminal Emacs, not in GUI Emacs.</li> <li>ls colouring, columns are aligned</li> <li>bash alias</li> <li>bash tab expansion</li> <li>command line redirection</li> <li>clear screen</li> <li>Command history</li> <li>Can run Python scripts.</li> <li>Running Python shell:               <ul style="list-style-type: none"> <li>REPL is OK, echo is OK</li> </ul> </li> </ul> <b>Limitations:</b> <ul style="list-style-type: none"> <li>In GUI Emacs: Meta-left/right cursor word move do not work. Use Esc-b and Esc-f here instead.</li> <li>Normal Emacs keystrokes does not always work, it depends on the programs that are executed from the shell. When it stops working, either use <b>C-c b</b> to switch to another buffer or exit the shell to gain control to Emacs keys in this buffer.</li> <li>Vertical cursor history works only with Control-Up and Control-Down</li> <li>Emacs keys with Meta do not work. The ones with Control do work.</li> <li>Can run top in the buffer, but then C-c does not stop it. To stop it split the buffer in 2, kill the buffer with C-x k, confirm, close the buffer.</li> </ul>		
Open a <u>vterm</u> shell	<f11> z v	(vterm &optional BUFFER-NAME)	Create a new vterm shell. A fast & full-featured *nix-compliant shell. 👉 Although vterm is relatively new this is the fastest shell. Highly recommended.
	📦 Requires the Emacs-libvterm (vterm) external package, the libvterm library. On macOS that can be installed with Homebrew. 🐧 PEL activates it when the pel-use-vterm user option is set to t. <ul style="list-style-type: none"> <li>Use C-c C-t to toggle the Vterm-Copy mode which allows navigation and text copy in the buffer.</li> <li>While the buffer is in Vterm mode you cannot use the PEL function keys as they are interpreted by the program running in the vterm shell. All other Emacs keys work. In Vterm-Copy the function keys are interpreted by Emacs so the PEL function key mappings do work.</li> <li>⚠️ vterm maximum scroll back size (the maximum number of lines the buffer can retain) is limited to 100000 lines. The value used is set by the <b>vterm-max-scrollback</b> user option which defaults to 1000. If you plan to use commands that print a long number of lines, you may want to change this value.</li> </ul> ⚠️🗿 When using this shell please first read the <a href="#">shell-side-configuration notes</a>		
Shell-mode Commands	The following commands are available in a shell-mode buffer.		
Resync directories. <ul style="list-style-type: none"> <li>Use to (re-)activate <b>tab completion</b> in shell.</li> </ul>	<f12> r	(shell-resync-dirs)	Resync the buffer's idea of the current directory stack.
	<ul style="list-style-type: none"> <li>Use this command to allow <b>tab completion</b> at the shell prompt if it does not work.               <ul style="list-style-type: none"> <li>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.</li> <li>DON'T issue this command unless the buffer is at a shell prompt.</li> <li>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.</li> </ul> </li> </ul>		

Description	Keystroke	Function	Note
<b>Specialized REPL</b> You can run several read eval run loop programming shells in Emacs. <ul style="list-style-type: none"> <li>Several of those REPLs, like <b>ielm</b> and <b>run-python</b> are part of Emacs.</li> <li>PEL makes the other available or adds some functionality to others when the corresponding pel-use- user option variable for the respective programming language is turned on (set to t).</li> <li>It is also possible to use shells to run other REPL programs directly from an embedded terminal shell like vterm (see above).</li> <li>The command for the Emacs Lisp REPL, ielm, is accessible via the pel:execute key prefix (<b>&lt;f11&gt; z</b>).</li> <li>The REPL for the other programming languages are accessible via the pel:repl key prefix (<b>&lt;f11&gt; z r</b>).</li> <li>All REPL commands are accessible via the <b>&lt;f12&gt; z</b> key binding of their respective major mode.</li> </ul>			
<b>Start Shell</b> See also: <b>⌘I - Arc</b>  • From Arc buffer	<b>&lt;f11&gt; z r C-a</b>	(run-arc CMD)	Run an inferior Arc process, input and output via buffer “arc”. <ul style="list-style-type: none"> <li>If there is a process already running in “arc”, switch to that buffer.</li> <li>With argument, allows you to edit the command line (default is value of ‘arc-program-name’).</li> <li>Runs the hook ‘inferior-arc-mode-hook’ (after the ‘comint-mode-hook’ is run).</li> <li>(Type h in the process buffer for a list of commands.)</li> </ul> 📦 Requires the <a href="#">arc-mode</a> external package. 🛠️ PEL activates this when the <b>pel-use-arc</b> user-options is set to t.
	<b>&lt;f12&gt; z</b>		
<b>Emacs Lisp shell</b> See also: <b>⌘I - Emacs Lisp</b>	• <b>&lt;f11&gt; z l</b>	(ielm)	Open the Interactive Emacs Lisp Mode buffer where you can interactively evaluate Emacs Lisp expressions, a REPL for Emacs Lisp. <ul style="list-style-type: none"> <li>Switches to the buffer “<b>ielm</b>”, or creates it if it does not exist.</li> <li><b>&lt;f12&gt; z</b> is only available in buffer in emacs-lisp-mode.</li> </ul>
	• <b>&lt;f12&gt; z</b>		
<b>Open a Common Lisp REPL</b>  🛠️ <b>pel-use-common-lisp</b> must be on.  See also: <b>⌘I - Common Lisp</b>  • From lisp-mode:	• <b>&lt;f11&gt; z r L</b>	(pel-cl-repl &optional N)	Open or switch to Common-Lisp REPL buffer window. <ul style="list-style-type: none"> <li>Use the Common Lisp REPL selected by the PEL user-options:               <ul style="list-style-type: none"> <li>SLY when ‘pel-used-sly’ is on and ‘pel-clisp-ide’ is set to sly,</li> <li>Slime when ‘pel-use-slime’ is on and ‘pel-clisp-ide’ is set to slime,</li> <li>the inferior lisp mode otherwise.</li> </ul> </li> <li>The behaviour of the command is affected by the optional argument N:               <ul style="list-style-type: none"> <li>with no buffers running REPL:                   <ul style="list-style-type: none"> <li>N is nil or absent: open REPL in current window</li> <li>N is positive: open REPL in other window</li> <li>N is negative: create new REPL in current window</li> </ul> </li> <li>with 1 or more REPL already running (if more than 1, prompt for one)</li> <li>if selected buffer is inside an opened window: switch to that window</li> <li>if selected buffer is not in an opened window:                   <ul style="list-style-type: none"> <li>N is nil or absent: open REPL in current window</li> <li>N is positive: open REPL in other window</li> <li>N is negative: create new REPL in current window.</li> </ul> </li> </ul> </li> </ul>
	• <b>&lt;f12&gt; z</b>		
<b>Elixir Shell !Ex</b>  See also: <b>⌘I - Elixir</b>	<b>&lt;f11&gt; z r x</b>	(alchemist-iex-run &optional ARG)	Start an <b>!Ex</b> process. <ul style="list-style-type: none"> <li>Show the !Ex buffer if an !Ex process is already run.</li> </ul> 📦 Requires the <a href="#">alchemist</a> package and the <a href="#">Elixir</a> programming language for your OS. 🛠️ PEL activates it when <b>pel-use-elixir</b> and <b>pel-use-alchemist</b> user-options are both set to t.
<b>Start Erlang Shell</b>  See also: <b>⌘I - Erlang</b>	• <b>&lt;f11&gt; z r e</b>	(erlang-shell)	Start a new Erlang shell. <ul style="list-style-type: none"> <li>The variable ‘erlang-shell-function’ decides which method to use, default is to start a new Erlang host. It is possible that, in the future, a new shell on an already running host will be started.</li> <li><b>C-c C-z</b> starts the Erlang Shell from the Erlang Mode.</li> <li><b>&lt;f11&gt; z r</b> starts it anytime, as long as it was installed.</li> </ul> 🛠️ Under PEL this command is available only when the <b>pel-use-erlang</b> user option is set to t.
	• <b>C-c C-z</b> • <b>&lt;f12&gt; z</b>		
<b>Open a Forth shell</b>  See also: <b>⌘I - Forth</b>  • From Forth buffer:	<b>&lt;f11&gt; z r f</b>	(run-forth)	Start an interactive forth session. <ul style="list-style-type: none"> <li>Prompt for a Forth executable.               <ul style="list-style-type: none"> <li>gforth is a good free implementation.                   <ul style="list-style-type: none"> <li>On macOS, you can install it with <b>brew install gforth</b> in a terminal shell.</li> </ul> </li> <li>⚠️ Notice that it is integrated with the Home-brew Emacs installation and it will upgrade your Homebre-based Emacs unless its pinned (in which case Homebrew won't install gforth).</li> </ul> </li> </ul> 📦 Requires the <a href="#">forth-mode</a> external package 🛠️ PEL installs and activates when the <b>pel-use-forth</b> user option is t. It also requires a Forth interpreter (which must be installed separately)
	<b>&lt;f12&gt; z</b>		
<b>Start Haskell Shell</b> See also: <b>⌘I - Haskell</b>  • From buffer	<b>&lt;f11&gt; z r h</b>	(run-haskell)	Show the inferior-haskell buffer. Start the process if needed. 📦 Requires the <a href="#">haskell-mode</a> and Haskell installed. 🛠️ PEL activates this when the <b>pel-use-haskell</b> and the <b>pel-use-haskell-mode</b> user-options are set to t.
	<b>&lt;f12&gt; z</b>		
<b>Start Julia Shell</b>  See also: <b>⌘I - Julia</b>  • From Julia buffer:	<b>&lt;f11&gt; z r j</b>	(julia-snail)	Start a Julia REPL and connect to it, or switch if one already exists. <ul style="list-style-type: none"> <li>The following buffer-local variables control it:               <ul style="list-style-type: none"> <li>‘julia-snail-repl-buffer’ (default: *julia*)</li> <li>‘julia-snail-port’ (default: 10011)</li> <li>To create multiple REPLs, give these variables distinct values (e.g.: *julia my-project-1* and 10012).</li> </ul> </li> </ul> 📦 Requires the <a href="#">julia-snail</a> Emacs package and the <a href="#">Julia</a> programming language installed. It also requires vterm (see above). 🛠️ PEL activates this when the <b>pel-use-julia</b> user option is set to t.
	<b>&lt;f12&gt; z</b>		
<b>LFE Shell (Lisp Flavoured Erlang)</b>  • From LFE buffer:	<b>&lt;f11&gt; z r C-l</b>	(run-lfe CMD)	Run an inferior LFE process, input and output via a buffer “inferior-lfe”. <ul style="list-style-type: none"> <li>If ‘CMD’ is given, use it to start the shell, otherwise:               <ul style="list-style-type: none"> <li>‘inferior-lfe-program’ ‘inferior-lfe-program-options’ -env TERM vt100.</li> </ul> </li> </ul> 📦 Requires the <a href="#">lfe-mode package</a> and LFE (Lisp Flavoured Erlang) installed. 🛠️ PEL activates this when the <b>pel-use-lfe</b> user option is set to t.
	<b>&lt;f12&gt; z</b>		
<b>Start OCaml Shell</b> See also: • From OCaml buffer	<b>&lt;f11&gt; z r o</b>	(run-ocaml)	Run an OCaml REPL process. I/O via buffer “OCaml”. 📦 Requires the <a href="#">tuareg</a> external package. 🛠️ PEL activates this when the <b>pel-use-ocaml</b> and the <b>pel-use-tuareg</b> user-options are set to t.
	<b>&lt;f12&gt; z</b>		
<b>Start Python Shell</b>  See also: <b>⌘I Python</b>  • From Python buffer:	<b>&lt;f11&gt; z r p</b>	(run-python &optional CMD DEDICATED SHOW)	Run an inferior Python process. <ul style="list-style-type: none"> <li>Argument CMD defaults to ‘python-shell-calculate-command’ return value. When called interactively with ‘prefix-arg’, it allows the user to edit such value and choose whether the interpreter should be DEDICATED for the current buffer. When numeric prefix arg is other than 0 or 4 do not SHOW.</li> <li>For a given buffer and same values of DEDICATED, if a process is already running for it, it will do nothing. This means that if the current buffer is using a global process, the user is still able to switch it to use a dedicated one.</li> </ul>
	<b>&lt;f12&gt; z</b>		

Description	Keystroke	Function	Note
<b>Start Chez Scheme Shell</b> See also: <ul style="list-style-type: none"> <li>From Chez buffer</li> </ul>	<f11> z r C-z	(pel-chez-repl &optional N)	Run the Chez REPL in window specified by N. <ul style="list-style-type: none"> <li>By default use the other window. If a numeric argument is specified, its value correspond to the direction of a numeric keypad:               <div> <div>8</div> <div>46</div> <div>2</div> </div>               That is:               <ul style="list-style-type: none"> <li>8: up</li> <li>4: left</li> <li>6: right</li> <li>2: down</li> <li>0 and 5 identify the current window.</li> </ul> </li> </ul> Requires the Chez Scheme installed.  PEL activates it when the pel-use-chez is set to t.
	<f12> z		
<b>Start Chibi Scheme Shell</b> See also: <ul style="list-style-type: none"> <li>From Chibi buffer</li> </ul>	<f11> z r C-i	(pel-chibi-repl &optional N)	Run the Chibi REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul> Requires the Chibi Scheme installed.  PEL activates it when the pel-use-chibi is set to t.
	<f12> z		
<b>Start Chicken Scheme Shell</b> See also: <ul style="list-style-type: none"> <li>From Chicken buffer</li> </ul>	<f11> z r C-k	(pel-chicken-repl &optional N)	Run the Chicken REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul> Requires the Chicken Scheme installed.  PEL activates it when the pel-use-chicken is set to t.
	<f12> z		
<b>Start Gambit Scheme Shell</b> See also:  - <b>Gambit Scheme</b> <ul style="list-style-type: none"> <li>From Gambit buffer</li> </ul>	<f11> z r C-b	(pel-gambit-repl &optional N)	Run the Gambit Scheme REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul>  Requires the gambit.el file and Chicken Scheme installed.  PEL activates it when the pel-use-gambit is set to t.
	<f12> z		
<b>Start Gerbil Scheme Shell</b> See also:  - <b>Gerbil Scheme</b> <ul style="list-style-type: none"> <li>From Gerbil buffer</li> </ul>	<f11> z r C-e	(pel-gerbil-repl &optional N)	Run the Gerbil REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul>  Requires the gerbil-mode external package and Gerbil Scheme installed.  PEL activates it when the pel-use-gerbil is set to t.
	<f12> z		
<b>Start Guile Shell</b> <ul style="list-style-type: none"> <li>From Guile buffer</li> </ul>	<f11> z r C-g	(pel-guile-repl &optional N)	Run the Guile REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul> Requires Guile Scheme installed.  PEL activates it when the pel-use-guile is set to t.
	<f12> z		
<b>Start MIT/GNU Scheme Shell</b> <ul style="list-style-type: none"> <li>From MIT/GNU Scheme buffer</li> </ul>	<f11> z r C-m	(pel-mit-scheme-repl &optional N)	Run the MIT/GNU Scheme REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul> Requires MIT/GNU Scheme Scheme installed.  PEL activates it when the pel-use-mit-scheme is set to t.
	<f12> z		
<b>Start Racket Shell</b> See also:  - <b>Racket</b> <ul style="list-style-type: none"> <li>From Racket buffer</li> </ul>	<f11> z r C-r	(pel-racket-repl &optional N)	Run the Racket REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul>  Requires the racket-mode external package and Racket installed.  PEL activates it when the pel-use-racket is set to t.
	<f12> z		
<b>Start Scsh Scheme Shell</b> <ul style="list-style-type: none"> <li>From Scsh buffer</li> </ul>	<f11> z r	(pel-scsh-repl &optional N)	Run the Scsh REPL in window specified by N. <ul style="list-style-type: none"> <li>See 'pel-chez-repl' for complete description.</li> </ul> Requires Scsh Scheme Scheme installed.  PEL activates it when the pel-use-scsh is set to t.
	<f12> z		

## Shells — References

Topic & Link	Extra Notes
<b>GNU Emacs - Running Shell Commands</b>	
<b>Eshell manual</b>	
<b>Difference between various emacs shells</b>	
<b>Difference between various emacs shells</b>	
<b>How to run multiple shells on Emacs</b>	
<b>EmacsWiki: Ansi Term</b>	Quick overview
<b>Emacswiki: Ansi Term Hints</b>	Several hints
<b>Copy/Paste in Ansi Term</b>	Quick overview of the capability for cut/paste.
<b>Launch GUI emacs from command line in OSX</b>	This describes a solution on how to start the GUI emacs in OSX, but not in the background
<b>How to launch GUI Emacs from command line in OSX?</b>	This one describes the solution for handling it in the background
<b>Run commands in background</b>	Describes the & and the disown
<b>Executing commands in background from bash scripts</b>	
<b>Pass command arguments to bash scripts</b>	