Shells, Terminal Emulators & Applications

Description	Keystroke	Function	Note
Emacs Shells	Emacs provides multiple ways of executing shell commands or running programming language specialized shells.		
Open this PDF file. See also: ∑ **Help/Info**	`<f11> x <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the local copy of the ∑ **Shells** PDF file unless a command prefix (like **C-u**) was used. In that case it opens the Github-hosted file instead.
∑ **Customize PEL shell management control**	`<f11> x <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL shell support: term. • If OTHER-WINDOW is non-nil (use **C-u)** , display in other window.
∑ **Customize Emacs shell control**	`<f11> x <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs shell support: term, terminal, term.
Launch OS Application from Emacs	With the following command you can launch an operating system application that will run independently of Emacs.		
	`<f11> A`	(**counsel-linux-app** &optional ARG)	Launch a Linux desktop application, similar to Alt-<F2>. When ARG is non-nil, ignore NoDisplay property in *.desktop files. 📦 On Linux, requires the **counsel** external package. PEL activates it when the **pel-use-counsel** user option is set to **t**.
		(**counsel-osx-app**)	Launch a macOS application via ivy interface. 📦 On macOS, requires the **counselx-osx-app** external package. PEL activates it when the **pel-use-counsel-osx-app** user option is set to **t**.
List Emacs Child Processes	Emacs can run several synchronous and asynchronous processes as child processes. They can be listed, showing the actual command line used to launch them with the following command.		
List processes	`<f11> x ?`	(**list-processes** &optional QUERY-ONLY BUFFER)	Display a list of all processes that are Emacs sub-processes. If optional argument QUERY-ONLY is non-nil, only processes with the query-on-exit flag set are listed. Any process listed as exited or signalled is actually eliminated after the listing is made.
Run Commands in system shell	The following commands can be used to quickly execute an external command inside a system shell process and display the result inside an Emacs buffer.		
Run a shell command	• `M-!` • `⌘-L`	(**shell-command** COMMAND &optional OUTPUT-BUFFER ERROR-BUFFER)	Prompts for the command in the minibuffer, show the command output in the next window in the *Shell Command Output* buffer in Fundamental mode.
Run a command on a marked region	`M-|`	(**shell-command-on-region** START END COMMAND &optional OUTPUT-BUFFER REPLACE ERROR-BUFFER DISPLAY-ERROR-BUFFER)	Execute string COMMAND in inferior shell with region as input. • Normally display output (if any) in temp buffer '*Shell Command Output*'; • Prefix arg means replace the region with it. Return the exit code of COMMAND. • Mark the region first. Then type **M-|**. Emacs prompts for the command to run. Use an argument to replace the region with the command output (ie. type **C-u M-|**)
Run a shell command asynchronously	`M-&`	(**async-shell-command** COMMAND &optional OUTPUT-BUFFER ERROR-BUFFER)	Execute string COMMAND asynchronously in background. • Like 'shell-command', but adds '&' at the end of COMMAND to execute it asynchronously. • The output appears in the buffer '*Async Shell Command*'. • That buffer is in shell mode.
Available Terminal-like Shells in Emacs Windows	Several terminal-like shells are available. They can be grouped in 3 categories: 1. **eshell**. Pure Emacs shell with all commands implemented in Emacs Lisp. Supports Unix style commands in any Operating System. Also support evaluation of Lisp expressions. If you know Emacs Lisp this can be extremely useful. 2. **vterm**. A relatively new shell for Emacs that is very fast. It's an external package that gets installed by PEL when the **pel-use-vterm** user option is set to **t**. Fo running Unix commands this is probably what you will want to use. See its installation information on its row below. 3. The other classical terminal and shells: **shell**, **ansi-term** and **term**. These all have pros and cons. They run slower than vterm but they are built-in. Of those, the ansi-term has more capabilities. Each have pros and cons. See the comparison table below in this document.		
Open an eshell	`<f11> x e`	(**eshell** &optional ARG)	Open an eshell buffer. ☝ To open another eshell instance: use the **C-u** prefix • To open a numbered eshell: use the **C-u number** prefix
	Implementation: • eshell is implemented in Emacs Lisp and implements several Unix commands, making them available to OS that do not natively have them (like Windows). If a command is not implemented it runs the one found in PATH. **Extra Features** • Can redirect output into a buffer. The grep command output goes to a grep result buffer which can be used to open the various files. • Support lisp commands. **Supports** • Cursor lateral cursor line beginning/end, kill, yank. • Meta-cursor word-move keys, but going left it does not stop at the prompt. • command tab expansion, command line re-direction • ls colouring (done by the eshell implementation), columns are aligned. • Command history (and shows history item # in mini-buffer) • Can run top, man, less (which start inside separate buffer) • Can run Python scripts. **Limitations:** • Meta-cursor word-move keys going left does not stop at the prompt. • Clear screen does not work • No bash alias, however eshell can remember its own aliases and will prompt for commands often ran & unfound.		
Open a vterm shell	`<f11> x 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**. • Use C-c C-t to toggle the Vterm-Copy mode which allows navigation and text copy in the buffer. • 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. • ⚠ 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 **vterm-max-scrollback** 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.		
Open a shell	`<f11> x s`	(**shell** &optional BUFFER)	☝ To open a shell instance inside another window: use the **C-u** prefix
	Implementation The oldest emacs shell. Uses the comint-mode. Emacs keys are possible, the sub-process does not see the keys until <RET> is pressed. **Supports** • Can run multiple shell, each inside its own buffer/name • Cursor lateral cursor line beginning/end, kill, yank. • Meta-cursor word-move keys. • bash alias • Command history (but with Control Up/Down) • Can run Python scripts. Can run Python REPL, REPL is OK, echo is OK, no Python colouring, but each command is colored. • Can run Common-Lisp (clisp) REPL ***Limitations***: • Clear screen does not work. • ls colouring does not work, ls columns are misaligned. • Can start and stop top, but the output is incorrect and cannot be read. • The shell PS1 prompt is partially applied, remnants show up on the second line. TO-INVESTIGATE?.		

Description	Keystroke	Function	Note
Open an ANSI term shell	`<f11> x a`	(**ansi-term** PROGRAM &optional NEW-BUFFER-NAME)	⚠ Normally **operates in character mode,** in which up/down navigation and kill/yank is not possible. Change to line mode to do that: • Use **C-x C-j** to change to line mode an allow movement, mark, saving. • When done use **C-c C-k** to switch to character mode.
	Implementation • Prompts for shell to use. Default is /bin/bash. Can use others. Opens in current window. • A terminal emulator written in Emacs Lisp. • Newer implementation than term. • You can even run other editors within it (vi, emacs, others). But use character-mode. **Specificities:** • C-x is mapped to term-escape-char **Supports:** • Scroll up/down with M-<up>, M-<down> • ls colouring, columns are aligned • bash alias • bash tab expansion • command line redirection • clear screen • Command history • Can run Python scripts. • Running Python shell: • REPL is OK, echo is OK **Limitations**: • Natively runs in character mode, which does not allow movement nor saving. • <up>, <down> cursor, C-n/C-p do not work as navigating: used as shell command history. Change to line mode (see above) to enable these. • Not yet found a way to control prompt (PS1 setup of .bash_profile does not seem to be used). 🚧		
Open a term shell	`<f11> x t`	(**term** PROGRAM)	Prompts for shell to use. Default is /bin/bash. Can use others. Opens in current window.
	Implementation: 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… **Supports** • Cursor lateral cursor line beginning/end, kill, yank. • Meta-cursor keys, but only in terminal Emacs, not in GUI Emacs. • ls colouring, columns are aligned • bash alias • bash tab expansion • command line redirection • clear screen • Command history • Can run Python scripts. • Running Python shell: • REPL is OK, echo is OK **Limitations**: • In GUI Emacs: Meta-left/right cursor word move do not work. Use Esc-b and Esc-f here instead. • Normal Emacs keystrokes does not always work, it depends on the programs that are executed from the shell. When it stops working, either use **C-c b** to switch to another buffer or exit the shell to gain control to Emacs keys in this buffer. • Vertical cursor history works only with Control-Up and Control-Down • Emacs keys with Meta do not work. The ones with Control do work. • 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.		
Specialized REPL	You can run several read eval run loop programming shells in Emacs. The **ielm** and **run-python** are part of Emacs. PEL makes the other available when the corresponding pel-use- user option variable is set to t. It is also possible to use shells to run other REPL programs directly from an embedded terminal shell like vterm (see above).		
Elixir Shell :IEx See also: **𝔓𝔏 - Elixir**	`<f11> x x`	(**alchemist-iex-run** &optional ARG)	Start an IEx process. • Show the IEx buffer if an IEx process is already run. 📦 Requires the alchemist package and the Elixir programming language for your OS. PEL activates this when the **pel-use-elixir** and **pel-use-alchemist** user options are both set to **t**.
Start Erlang Shell See also: **𝔓𝔏 - Erlang**	• `<f11> x r` • `C-c C-z`	(**erlang-shell**)	Start a new Erlang shell. • 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. • `C-c C-z` starts the Erlang Shell from the Erlang Mode. • `<f11> x r` starts it anytime, as long as it was installed. Under PEL this command is available only when the **pel-use-erlang** user option is set to **t**.

Open a Forth shell
 	
See also: **𝔓𝔏 - Forth**
	`<f11> x f`	(**run-forth**)	Start an interactive forth session.
			• Prompt for a Forth executable.
			  • gforth is a good free implementation.
			  • On macOS, you can install it with `brew install gforth` in a terminal shell.
			  • ⚠ 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).
			📦 Requires the **forth-mode** external package PEL installs and activates when the **pel-use-forth** user option is **t**. It also requires a Forth interpreter (which must be installed separately)

Emacs Lisp shell See also: **𝔓𝔏 - Emacs Lisp**	`<f11> x i`	(**ielm**)	Open the Interactive Emacs Lisp Mode buffer where you can interactively evaluate Emacs Lisp expressions, a REPL for Emacs Lisp. • Switches to the buffer '***ielm***', or creates it if it does not exist.

Start Julia REPL
 	
See also: **𝔓𝔏 - Julia**
	`<f11> x j`	(**julia-snail**)	Start a Julia REPL and connect to it, or switch if one already exists.
			• The following buffer-local variables control it:
			  • 'julia-snail-repl-buffer' (default: *julia*)
			  • 'julia-snail-port' (default: 10011)
			  • To create multiple REPLs, give these variables distinct values (e.g.: *julia my-project-1* and 10012).
			📦 Requires the julia-snail Emacs package and the Julia programming language installed. It also requires vterm (see above).
			PEL activates this when the **pel-use-julia** user option is set to **t**.

LFE Shell **(Lisp Flavoured Erlang)**	`<f11> x l`	(**run-lfe** CMD)	Run an inferior LFE process, input and output via a buffer '*inferior-lfe*'. • If 'CMD' is given, use it to start the shell, otherwise: 'inferior-lfe-program' 'inferior-lfe-program-options' -env TERM vt100. 📦 Requires the lfe-mode package and LFE (Lisp Flavoured Erlang) installed. PEL activates this when the **pel-use-lfe** user option is set to **t**.
Start Python Shell See also: **𝔓𝔏 Python**	`<f11> x p`	(**run-python** &optional CMD DEDICATED SHOW)	Run an inferior Python process. • 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. • 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.

Performance/Feature Comparisons of Emacs Shells/Terminals

Emacs Shell/Feature	eshell	shell	ansi-term	term	vterm	Comment
Relative speed comparison: Execute "ls -lFGO" inside /usr/local/bin/ . (Execution times in seconds for several attempts at the same command).	• 2.448571 • 4.247726 • 2.550193 • 2.631693 • 2.510235 • 4.220897	• 2.514221 • 2.472229 • 2.514438 • 2.468948 • 2.765349	• 6.169079 • 5.431559 • 5.493072 • 5.398879 • 5.435839	• 5.586079 • 5.531138 • 5.519672 • 5.227298 • 5.526750	• 0.065568 • 0.073241 • 0.053149 • 0.048021 • 0.060560 • 0.109644	Tested the execution time of listing a directory that has 861 entries (mostly symlinks), a /usr/local/bin on a macOS computer.
Toggle terminal mode to allow editing navigation	Standard Emacs keys always available for navigation but cursor keys used by the terminal for history.	Not available: always in Emacs editing mode.	out: **`C-x C-j`** in: **`C-c C-k`**	out: **`C-x C-j`** in: **`C-c C-k`**	out: **`C-c C-t`** in: **`C-c C-t`**	It is best to have 2 modes to use applications inside the terminal and have keys to switch between the "*pure terminal mode*" and Emacs navigation mode.
Escape Sequences and colouring works	Implement its own, does not render everything applications support.	Partially. Escape sequences work partially but colouring does not.	Yes	Yes	Yes	
Shell prompt definition support (PS1)		Yes, but tput expressions to boldface prompt does not work.	Yes	Yes	Yes but requires code in shell configuration	Although vterm requires extra configuration that also provides extra functionalities.
clear works	Almost: clears the screen but leaves cursor at the bottom of the window.	No (problem with escape sequences)	Yes	Yes	Yes	
Support bash aliases	No but supports its own.	Yes	Yes	Yes	Yes	
F1-F12 keys available to terminal. • Yes: available to terminal. • No: used by Emacs only.	No	No	No	No	Yes	When the F1-F12 keys are used by terminal they can be used by applications that use them. They are, however not available to Emacs until you toggle the terminal mode off (using the keys identified in the second row above (eg. **`C-c C-t`** for vterm.)
History via cursor	Yes	No	Yes	Yes	Yes	
Advantage	Implemented in Emacs Lisp, available in all environments even on non-*nix like Windows.				Best speed I have on my system, and pure terminal control.	For fast operations on something that is close to a real terminal, **vterm** is the best available on *nix platforms as far as I can tell at the moment (April 2020). The **eshell** is useful to perform operations on platforms where Unix-like utilities are not available and where you want to use Emacs lisp code. It integrates with Emacs functionality, standing on its own.
Can run scripts (interpret shebang line)	No. But can run script if the interpreter is specified explicitly.	Yes	Yes	Yes	Yes	
Runs other REPLs	Yes, as long that the shell is an executable on the PATH. It does not support bash alias that are sometimes used to launch shells.	Was able to use python, clisp, iex, but not LFE: it launched Erlang REPL instead. iex was coloured properly.	Yes, with colouring.	Yes, with colouring.	Yes, good speed, supports colouring. Use **`C-c C-c`** for Control-C, **`C-c C-g`** for Control-G	Again here, the best shell to run another real from the command line is vterm. However, it's also possible to run these REPLs from within Emacs. Using them from within another shell allows using one quickly or testing.
Can run Emacs Lisp commands	Yes	No	No	No	Yes	Some shells allow mapping keys to Emacs Lips command code.
Interact with Emacs from the shell	Yes, using elisp code	No	No	No	Yes, with special escape sequences for message passing.	

Shells — References

Topic & Link	Extra Notes
GNU Emacs - Running Shell Commands	
Eshell manual	
Difference between various emacs shells	
Difference between various emacs shells	
How to run multiple shells on Emacs	
EmacsWiki: Ansi Term	Quick overview
Emacswiki: Ansi Term Hints	Several hints
Copy/Paste in Ansi Term	Quick overview of the capability for cut/paste.
Launch GUI emacs from command line in OSX	This describes a solution on how to start the GUI emacs in OSX, but not in the background

Topic & Link	Extra Notes
How to launch GUI Emacs from command line in OSX?	This one describes the solution for handling it in the background
Run commands in background	Describes the & and the disown
Executing commands in background from bash scripts	
Pass command arguments to bash scripts	