↖ Topic Index

🚦 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.
Supports built-in serial terminal emulator?				Yes, use: **M-x serial-term**		
Special installation/configuration Notes					term shell-side configuration	Read configuration/installation notes for the specific shell.
Advantage	Implemented in Emacs Lisp, available in all environments even on non-*nix like Windows.	Flexible, good compromise between speed and availability of a mix of features from the shell and from Emacs since Emacs key bindings are available.			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.
Limitations		The sub-process does not see the command until the RET key is pressed. Therefore do not use this shell for running interactive programs that wait on keyboard input.				
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-c C-j** in: **C-c C-k**	out: **C-c C-j** in: **C-c C-k**	out: **C-c C-t** in: **C-c C-t**	The shells differ in their way to allow key bindings. The eshell and shell buffers support all Emacs key bindings while the shell is in control. The ansi-term, term and vterm have two input modes and key sequences to switch between them.
Emacs key bindings available while shell input mode is active	Yes	Yes	Some of them, not all: in shell input mode, the **C-x** prefix is replaced by the **C-c** prefix. Type **C-c C-j** to switch to Emacs input mode, then use Emacs key sequences. Return to shell input mode by typing **C-c C-k**	Some of them, not all: in shell input mode, the **C-x** prefix is replaced by the **C-c** prefix. Type **C-c C-j** to switch to Emacs input mode, then use Emacs key sequences. Return to shell input mode by typing **C-c C-k**	Only some of them (the ones that start with **Esc**). Type **C-c C-t** to switch to Emacs input mode, then use Emacs key sequences. Return to shell input mode by typing **C-c C-t**	The term, ansi-term and vterm buffers operate with 2 different input modes: • shell input mode (char input) • Emacs input (line input) In **term** and **ansi-term** buffers you must put the buffer in Emacs input (line input) mode, by typing **C-c C-j**, to be able to access the PEL commands that use the **<f12>** key prefix. The **<f11>** key prefix is always available. In vterm you must put the buffer in Emacs input (line input) mode, by typing **C-c C-t**, to be able to access the PEL commands that use the **<f11>** or **<f12>** key prefix. Both are always available in the eshell and shell buffers.
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.)
Escape Sequences and colouring works	Implement its own, does not render everything applications support.	Partially. Escape sequences work partially but other type of 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 shell command works?	Almost: clears the screen but leaves cursor at the bottom of the window.	No However, the Emacs **comint-clear-buffer** does work. It's bound to **C-C M-o**. PEL adds a **<f12> c** key binding.	Yes	Yes	Yes	
Support bash aliases	No but supports its own.	Yes	Yes	Yes	Yes	
Shell tab completion	N/A	Yes, but completion is done by Emacs and it might get out of sync with the directory. Execute **shell-resync-dirs** to correct.			Yes	
History via cursor keys	Yes	• Not supported by cursors (which move point) • But supported by using CTRL key allowing with the cursor keys.	Yes	Yes	Yes	

Emacs Shell/Feature	eshell	shell	ansi-term	term	vterm	Comment
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 Lisp command code.
Interact with Emacs from the shell	Yes, using elisp code	No	No	No	Yes, with special escape sequences for message passing.	
Ability to write keyboard macros that interact wit h a shell	As the following columns show, the shell is the most flexible standard shell in term of ability to execute commands with any key bindings and can easily be used for keyboard macro that compose shell commands. The eshell is similar but you need to use Emacs Lisp syntax.					
Emacs Shell/Feature	**eshell**	**shell**	**ansi-term**	**term**	**vterm**	**Comment**
Can yank text in shell	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: No • macOS: No	• Linux: No • macOS: No	• Linux: Yes • macOS: Yes	
Can navigate out of buffer with commands with Esc key prefix	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: No • macOS: No	• Linux: No • macOS: No	• Linux: Yes • macOS: Yes	This is the same as being able to execute any commands that use an **Esc** key prefix.
Can navigate out of buffer with commands with <f1> key prefix	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: No • macOS: No	This is the same as being able to execute any commands that use any function key as key prefix.

Terminal Multiplexers and Emacs

Terminal multiplexer	Topic	Information & Links
GNU Screen	**References:**	• GNU Screen @ Wikipedia : start here if you do not know what this program is. • GNU Screen home page • GNU Screen Manuals • **GNU Screen Manual - all in 1 HTML Page** (useful to search)

	Using Emacs within an GNU Screen Session	• By default GNU Screen uses the **C-a** key as the Screen command key.
		  • To pass C-a to Emacs running inside a GNU Screen session: type **C-a** followed by **a**
		  • Screen command key can be changed with the **escape** setting in the ~/.screenrc file. See next lines for 2 examples:
		  • To change it to **C-^**, write: **escape ^^^**
		  • The first **^^** is the caret representation of Control-^. The last **^** is the single key to type after to pass **C-^** to the program running under Screen (like Emacs). Another character could be used, 6 for example.
		  • To change it to **C-z**, write: **escape ^zz**

	Logging with Screen	Screen supports dumping the current content of the screen to a file or **log the complete window session to a file**. • This second feature is quite useful when running long lasting commands like software builds preformed from a shell. • The session can be started inside a screen window, and hidden to speed it up while logging all the details inside the log file. • The log file will contain the entire output to stdout and stderr. It will also contain all the escape sequence codes printed on your shell to colonize it for example. • You can view this log file inside Emacs and use the **pel-screen-log-fix-rendering** command (bound to **<f11> t s**) to filter these escape codes out of the buffer and render the colours. See also: ⅀ **Text Modes**