↖ Topic Index

🚦 Performance/Feature Comparisons of Emacs Shells/Terminals

Emacs Shell/Feature	eshell	shell (𝕀 shell-mode)	ansi-term (𝕀 term-mode)	term (𝕀 term-mode)	emacs eat (𝕀 eat-mode)	vterm (𝕀 vterm-mode)	Comment
Last updated on:	2025-05-05						
Relative speed comparison: Execute "ls -lFGO" inside /usr/local/bin/ on macOS. (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	Not measured.	• 0.065568 • 0.073241 • 0.053149 • 0.048021 • 0.060560 • 0.109644	
SEE ALSO: PEL shell/REPL invocation commands	• **EShell Manual** • **Mastering EShell**, from Mickey Petersen						Tested the execution time of listing a directory that has 861 entries (mostly symlinks), a /usr/local/bin on a 2014 macOS computer.
Supports built-in serial terminal emulator?				Yes, use: `M-x serial-term`			
Support running GNU Screen within an Emacs internal shell in local host? ☝ One would normally start screen at the **remote** host to establish a context and connect to it via ssh. If the ssh link breaks you can re-connect to the screen session where it left off. • Using screen inside a Emacs terminal buffer is probably not very useful unless you want to use **GNU screen logging** facility to record the stdout/stderr output of a long running job and want to interact with other Emacs buffer while doing so.	**No**, the screen command launches inside a term buffer. The eshell remains running independently.	**No**, the mode lacks screen clear capability.	**Yes: Linux, macOS** • Start term with **`M-x ansi-term RETURN`**. Inside the created shell, execute the ***screen*** command. • Start screen directly with **`M-x ansi-term screen RETURN`**.	**Yes: Linux, macOS** • Start term with **`M-x term RETURN`**. Inside the created shell, execute the ***screen*** command. • Start screen directly with **`M-x term screen RETURN`**.	Yes: macOS	Yes: macOS	Tested in Linux and macOS environments only. • Did not test vterm in Linux yet.
Support running GNU Screen within shell in remote host by issuing a ssh command within that shell and then executing screen.	**No**, the screen command launches inside a term buffer. The eshell remains running independently.	**No**, the mode lacks screen clear capability.	**Yes: Linux, macOS** • Within the term invoked shell, issue the **ssh** command first to connect to the remote host, then issue the **screen** command in the remote host.			Yes: macOS	Tested in Linux and macOS environments only. • Did not test vterm in Linux yet.
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.			Saw some problems briefly using eat 0.9.4 On macOS Sonoma, arm64 CPU, in both Terminal.app text mode and GUI mode Emacs 29.3 with zsh and bash configurations identified as prompt model 2 in my **USRHOME project**, the backspace key did not work in zsh and bash prompt failed. ✅ setting **TERM** to **xterm-256color** inside the eat terminal shell solved the above problems. ***Very flexible, fast, compared to term, but still young with some bugs left (eg. cloning buffers, dir tracking not working). Worth trying !***	Currently does not work on macOS Silicon. There's an open bug: **vterm-module compiles as x86_64 instead of arm64e on macOS M1 #593**	

Emacs Shell/Feature	eshell	shell (𝕀 shell-mode)	ansi-term (𝕀 term-mode)	term (𝕀 term-mode)	emacs eat (𝕀 eat-mode)	vterm (𝕀 vterm-mode)	Comment
Open multiple shell of this type by renaming the buffer of the existing one with rename-buffer. • **`M-x rename-buffer`** • **`<f11> b R`**	Yes	Yes	Yes	Yes	Yes	Yes	• This method is not specific to the terminals. It also applies to other types of buffers like all REPL buffers. • It's best to use a name that starts and ends with a '*' to identify these buffer as special.
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`**	It has several, depending of the input mode that is currently active (see below). • **`C-c C-j`** : ➡ semi-char mode • **`C-c C-e`** : ➡ emacs mode • **`C-c M-d`** : ➡ char mode • **`C-c C-l`** : ➡ line input mode • **`C-M-m`** or **`M-RET`**: ➡ semi-char	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`**
					eat has 4 input modes:
					• ***semi-char mode***: most keys are sending to terminal except C-\, C-c, C-x, C-g, C-h, C-M-c, C-u, C-q, M-x, M-:, M-!, M-& . Special bindings are:
					  • **`C-q`**: send next key to terminal
					  • **`C-y`** : like yank, but send text to terminal
					  • **`M-y`**: like yank-pop: but send text to terminal
					  • **`C-c C-k`** : kill process
					  • **`C-c C-e`** : ➡ emacs mode
					  • **`C-c M-d`** : ➡ char mode
					  • **`C-c C-l`** : ➡ line input mode
					• ***emacs mode:*** *use it to navigate and edit*: no special key binding except:
					  • **`C-c C-j`**: ➡ semi-char mode
					  • **`C-c M-d`** : ➡ char input mode
					  • **`C-c C-l`** : ➡ line input mode
					• ***char mode***: All keys are sent to terminal except:
					  • **`C-M-m`** or **`M-RET`**: ➡ semi-char
					• ***line mode***: similar to comint, shell-mode and term line mode: terminal inout is sent line-wise, allowing editing line with Emacs commands. Extra binding:
					  • **`C-c C-e`** : ➡ emacs mode
					  • **`C-c C-j`** : ➡ semi-char mode
					  • **`C-c M-d`** : ➡ char mode
						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	• ***semi-char mode: No*** • ***emacs mode: No*** *(but irrelevant)* • ***char mode:*** **Yes.** • Can use htop and exit it with **`<f10>`** • ***line mode***: No	Yes • Can use htop and exit it with **`<f10>`**	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.) Use an application like htop that use the function keys with eat in char mode or vterm.

Handle backspace/delete at program prompt properly	N/A	Yes	No, escape sequences are printed and passed to the program stdin	No, escape sequences are printed and passed to the program stdin	No, escape sequences are printed and passed to the program stdin. None of the modes handle that properly.	Yes	Tested with this simple/naive Perl script:
							`#!/usr/bin/perl`
							`use v5.10;`
							`print "Enter a number: ";`
							`chomp($number = <STDIN>);`
							`say "The number is $number.";`

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. ⚠ However on macOS, **TERM** must be set to **xterm-256color.** • See USRHOME shell config code	Yes	

Emacs Shell/Feature	eshell	shell (𝕀 shell-mode)	ansi-term (𝕀 term-mode)	term (𝕀 term-mode)	emacs eat (𝕀 eat-mode)	vterm (𝕀 vterm-mode)	Comment
Shell prompt definition support (PS1, PS2, PS3, PS4, …)	Irrelevant. eshell is not a POSIX shell and is controlled from within Emacs.	Yes, but tput expressions to boldface prompt does not work.	Yes	Yes	• ***semi-char mode:*** *Yes* • ***emacs mode:*** *Yes, but irrelevant.* • ***char mode:*** *Yes* • ***line mode***: Yes	Yes but requires code in shell configuration . This also provides extra functionalities like current directory tracking.	
Handle zsh RPROMPT	Irrelevant. eshell is not a POSIX shell and is controlled from within Emacs.	No	No	No	• ***semi-char mode:*** *Yes* • ***emacs mode:*** *Yes, but irrelevant.* • ***char mode:*** *Yes* • ***line mode***: Not really. The typed command appears at the right of the RPROMT. This is not what zsh intent is.	Yes	The zsh can print information at the right-hand side of the prompt line. So the command being typed is shown to its left. The zsh RPROMPT does not work well with any of the terminal emulators I have tested except for vterm. It almost works for eat except for its line-mode.
Interfere with Bash PROMPT_COMMAND	No	No	No	No	Yes see this **bug report**	No	
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	• ***semi-char mode:*** *Yes* • ***emacs mode:*** *No. For editing only.* • ***char mode:*** *Yes* • ***line mode***: Yes	Yes	
Support bash aliases	No but supports its own.	Yes	Yes	Yes	Yes	Yes	
Shell tab completion	Yes, but eshell is not a POSIX shell and is controlled within Emacs, providing a tight integration with Emacs.	Yes, but completion is done by Emacs and it might get out of sync with the directory. Execute **shell-resync-dirs** to correct. This, however, does not always work.	Yes	Yes	• ***semi-char mode:*** *Yes* • ***emacs mode:*** *No, but irrelevant.* • ***char mode:*** *Yes* • ***line mode***: Yes	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	• ***semi-char mode:*** *Yes* • ***emacs mode:*** *No, but irrelevant.* • ***char mode:*** *Yes* • ***line mode***: Yes	Yes	
Can run scripts (interpret shebang line)	No, since it's not a POSIX shell. • But can run script if the interpreter is specified explicitly. • It can, however, run any elisp code!	Yes	Yes	Yes		Yes	
Runs other REPLs inside the terminal	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.	• ***semi-char mode:*** *Yes, no colouring.* • ***emacs mode:*** *No. For editing only.* • ***char mode:*** *Yes, no colouring.* • ***line mode***: Yes, no colouring.	Yes, good speed, supports colouring. Use `C-c C-c` for Control-C, `C-c C-g` for Control-G	The best shell to run another REPL from the command line is vterm. However, it's also possible to run these REPLs directly from within Emacs. Using them from within another shell allows using one quickly or testing.
Can run Emacs Lisp commands via key bindings	Yes	No	No	No	Yes	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.	
Supports all shell prompt formatting	N/A	No, some escape sequences are not supported.	Yes, all formatting is supported.	Yes, all formatting is supported.	Yes, all formatting is supported.	Yes, all formatting is supported.	
Handle Window resizing nicely	Yes, all is fine, no bleeding, prompt repeating.	No, with some shell prompts, resizing the window may cause extra prompt printing	Yes, all is fine, no bleeding, prompt repeating.	Yes, all is fine, no bleeding, prompt repeating.	Yes, all is fine, no bleeding, prompt repeating, even in line-input mode (as used in shell-mode)	Yes, all is fine, no bleeding, prompt repeating.	

Emacs Shell/Feature	eshell	shell (𝕀 shell-mode)	ansi-term (𝕀 term-mode)	term (𝕀 term-mode)	emacs eat (𝕀 eat-mode)	vterm (𝕀 vterm-mode)	Comment

Handles ssh sessions nicely	N/A	• This sets TERM to dumb. It's a problems on hosts that have logic to help Tramp connect and prevent their zsh or bash configuration on dumb terminals.
		  • A work-around is to create a sub-shell by setting TERM to something else, as in:
		  TERM=xterm-256color bash
		• It has problems with prompts that use escape codes to color it. Under USERHOME use prompt mode 1:
		  usrhome-prompt-model-to 1
			All works fine.	All works fine.	All works fine.	All works fine.	

Keyboard Macros and Shells	One of the most compelling reasons for using a terminal shell within Emacs is the ability to interact between the rest of Emacs and the shell in a semi automated way with 𝕀 **Keyboard Macros**. Using a terminal shell inside a window and a file or another REPL inside another window, it becomes possible to create keyboard macros that insert text inside a file from the result of commands executed in a shell or a REPL (like a Python or Erlang REPL) . To enable this, you must be able to move across Emacs windows using key bindings as simply as possible and you must be able to copy text from one window and yank it inside another.	As the following columns show, this can be done most flexibly with two of the available terminal shell modes: • shell • eat Both of these modes are flexible enough to execute commands and use the Emacs key bindings to navigate and copy text across windows without changing mode input mode. The shell mode always works. The eat-mode runs faster and can also be used in line-mode for that. If you are willing to switch inputs mode, you can use extra keystrokes to use semi-char and char modes too.	The eshell is similar but you need to use Emacs Lisp syntax.

Emacs Shell/Feature	eshell	shell	ansi-term	term	emacs eat	vterm	Comment

Can yank text in shell	• Linux: Yes
	• macOS: Yes
		• Linux: Yes
		• macOS: Yes
			• Linux: No
			• macOS: No
				• Linux: No
				• macOS: No
					• ***semi-char mode:*** *Yes:*
					  • `C-y` : like yank, but send text to terminal
					  • `M-y`: like yank-pop: but send text to terminal
					• ***emacs mode: No***
					• ***char mode:*** *Yes, using the OS key sequence.*
					• ***line mode***: *Yes, using the OS key sequence.*
						• Linux: Yes
						• macOS: Yes	

Can navigate out of window with PEL Esc cursor key sequences	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: No • macOS: No	• Linux: No • macOS: No	• ***semi-char mode: No*** • ***emacs mode:*** *Yes* • ***char mode: No*** • ***line mode***: Yes	• 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 window with PEL <f1> cursor key sequences	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• Linux: Yes • macOS: Yes	• ***semi-char mode:*** *Yes* • ***emacs mode:*** *Yes* • ***char mode: No*** • ***line mode***: 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)
	GNU Screen source code	• GNU Screen Git Repository - Savannah

	Compile GNU Screen:	`git clone https://git.savannah.gnu.org/git/screen.git`
		`cd screen/src`
		`./autogen.sh`
		`./configure --prefix=/usr/local \`
		`  --enable-pam \`
		`  --enable-colors256 \`
		`  --enable-rxvt_osc \`
		`  --enable-use-locale \`
		`  --enable-telnet`
		`make && make install`

	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	GNU 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: **Buffers** , **Text Modes**
	Multi-user screen	Use GNU screen to allow simultaneous access to a shell for several users! See: • GNU Screen Manual - Multiuser Session • https://aperiodic.net/screen/multiuser • Unix & Linux: Sharing a terminal with multiple users (with screen or otherwise) • 2012 UTOSC - Screen vs. tmux faceoff - Jon Jensen - Youtube video
Terminal Emulators with capabilities to support more key bindings	Running Emacs in text mode restricts the key bindings that can be used with Emacs. According to Yuri Khan, on this help-gnu-emacs mailing list post it's possible to use specialized terminal emulators to extend the key binding possibilities: *Terminal emulators such as Kitty and Wezterm have an advanced key encoding[1] where every key is distinguishable from every other key. You'd have to write a terminal-specific library that enables that encoding and modifies input-decode-map to decode that encoding. Also you'd have to make sure the terminal properly advertises itself in the TERM environment variable so that Emacs knows when to apply your library. (TERM=xterm or TERM=xterm-256color won't cut it.) After you do this, you basically have the same key set as a GUI frame* There's an emacs package that uses the kitty protocol package: **kkp.el**	
	kitty	• Kitty @ Wikipedia • Kitty home • Kitty @ Github • Comprehensive keyboard handling in terminals. Read this to use it to provide extra binding capabilities.
	Wezterm	• WezTerm home page • WezTerm @ Github • WezTerm @ Gentoo

Terminal multiplexer	Topic	Information & Links
Terminal Emulators	**Information about terminal emulators**	
	Background	• **The TTY demystified,** a very good introduction.