↖ Topic Index

Manage and Launch Shells, REPLs & Applications

Description	Keystroke	Function	Note
Emacs Shells Last updated on: 2025-10-23	Emacs provides multiple ways of executing shell commands or running programming language specialized shells and programming language **REPL**. • It provides multiple terminal emulators and shells. There's also several external packages that provide more. This page describe the commands available to start these shells, terminal emulators and REPLs inside Emacs buffer windows. Note that inside the term, ansi-term and vterm buffers, all Emacs keys are not always available: these major modes operate in to input modes: • shell input (char) mode: where the shell gets the keys • Emacs input (line) mode: where Emacs key bindings are available. See 🚦 **Shells/Terminals Comparisons** for more information.		
Open this PDF file. See also: **Σ Help/Info**	`<f11> z <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open this **Σ 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.
Σ Customize PEL shell management control	`<f11> z <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL shell support • If OTHER-WINDOW is non-nil (like **C-u** or **M--**), open customize buffer in other window.
Σ Customize Emacs shell & term control	`<f11> z <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs shell group. • If OTHER-WINDOW is non-nil (like **C-u** or **M--**), open customize buffer in other window.
Σ Customize Emacs shell control	`<f11> SPC SPC s <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs **shell-mode** support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
	`<f12> <f3>`		In **shell-mode** the `<f12> <f3>` key opens the **shell** customization group.
Σ Customize Emacs shell control See also: **🚦 Shells/Terminals Comparisons**	`<f11> SPC SPC t <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs **term-mode** support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
	`<f12> <f3>`		In **term-mode** the `<f12> <f3>` key opens the **term** customization group. • ⚠️ The key sequence is only available in term and ansi-term buffers when the buffer is operating in Emacs (line) input mode. Toggle to line input mode by typing `C-c C-j`.
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 with **pel-use-counsel-osx-app** user option.
List Emacs Child Processes See also: **Σ Help/Info**	Emacs can run several synchronous and asynchronous processes as child processes.		
	• `<f11> z ?` • `<f11> ? e C-p`	(**list-processes** &optional QUERY-ONLY BUFFER)	Display a list of all processes that are Emacs sub-processes. • With optional argument (**C-u**) : 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. • Use these commands from buffer using any major modes. Also see: **Executing Shell Commands in Emacs** from **Mastering Emacs**.		
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. Both stdout and stderr are shown. • To send stderr to another buffer set **shell-command-default-error-buffer** user-option. • With **C-u** prefix, insert output in the current buffer.
Run a shell command as sudo	`<f11> z !`	(**pel-shell-as-sudo**)	Prompt for command, then sudo password and execute the command in a shell with sudo credentials. Print the results in the *Shell Command Output* buffer in Fundamental mode.
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 end of COMMAND to execute it asynchronously. • The output appears in the buffer '*Async Shell Command*'. That buffer is in shell mode.
Run a command on a marked region • **C-u** : replace region with cmd output ★★	`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. ☝ To **replace the region with the command output**: type **C-u M-|**
Open a shell or terminal buffer See **🚦 Shells/ Terminals Comparisons**	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. 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. 3. There are others such as term and eat. See below.		
Open an eshell • **Eshell manual** • **Mastering Eshell**	`<f11> z 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 shell in shell-mode **See: Σ shell-mode**	`<f11> z s`	(**pel-shell**)	Opens an inferior shell in the *current window* or moves point to the *shell* buffer already showing in one of the windows.
	Implementation • This is the PEL implementation which uses the built-in Emacs shell command and ensures it opens inside the current window, like **term**, **ansi-term**, **ielm** and **vterm**. ☝ On Emacs prior to 29.1, Emacs built-in **shell** commands creates a window in the *other* window. This is a surprising behaviour compared to the other inferior process commands and the PEL implementation fixes that. On Emacs 29.1 and later the **shell** command behaves properly (and so does **pel-shell**) • The Emacs shell command is the oldest one. It uses the **comint-mode**, which makes it quite versatile. Emacs keys are possible, however the sub-process does not see the keys until <RET> is pressed making it unfit for programs that directly read the input. **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, zsh 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. directly but PEL provides the **`<f12> c`** or **`C-c M-o`** . See below. • Not a good candidate for running UI applications such as top, htop, etc…		

Description	Keystroke	Function	Note
Open an ANSI term shell **See:** 𝕀 **term-mode**	`<f11> z a`	(**ansi-term** PROGRAM &optional NEW-BUFFER-NAME)	⚠️Normally **operates in character mode, i**n 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 **See:** 𝕀 **term-mode**	`<f11> z 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.		
Open a vterm shell **See** 𝕀 **vterm-mode**	`<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 **vterm-mode** external package and Emacs-libvterm (vterm) external package, the libvterm library. PEL activates it when the **pel-use-vterm** user option is set to **t** • On macOS that can be installed with Homebrew. • 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. ⚠️🎛 When using this shell please first read the **shell-side-configuration notes**.		
Open a eat terminal emulator **See** 𝕀 **eat-mode**	`<f11> z f`	(eat &optional *PROGRAM ARG*)	Start a new Eat terminal emulator in a buffer. • Start a new Eat session, or switch to an already active session. Return the buffer selected (or created). • With a non-numeric prefix ARG, create a new session. • With a numeric prefix ARG (like `C-u 42 <f11> z f`), switch to the session with that number, or create it if it doesn't already exist. • With double prefix argument ARG, ask for the program to run and run itin a newly created session. • PROGRAM can be a shell command.
	📦 Requires the **emacs-eat** external package. PEL activates it when the **pel-use-emacs-eat**user-option is set to **t**.		
Specialized REPL	You can run several read eval run loop programming shells in Emacs. • Several of those REPLs, like **ielm** and **run-python** are part of Emacs. • 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**). • It is also possible to use shells to run other REPL programs directly from an embedded terminal shell like vterm (see above). • The command for the Emacs Lisp REPL, ielm, is accessible via the pel:execute key prefix (`<f11> z`). • The REPL for the other programming languages are accessible via the pel:repl key prefix (`<f11> z r`). • All REPL commands are accessible via the `<f12> z` key binding of their respective major mode.		

Start Shell
See also: 𝔓𝔏 **- Arc**
 	
• From Arc buffer
	`<f11> z r C-a`
	
	`<f12> z`
		(**run-arc** CMD)	Run an inferior Arc process, input and output via buffer '*arc*'.
			• If there is a process already running in '*arc*', switch to that buffer.
			• With argument, allows you to edit the command line (default is value of 'arc-program-name').
			• Runs the hook 'inferior-arc-mode-hook' (after the 'comint-mode-hook' is run).
			• (Type h in the process buffer for a list of commands.)
			📦 Requires the **arc-mode** external package.
			PEL activates this when the **pel-use-arc** user-options is set to **t**.

Emacs Lisp shell See also: 𝔈𝔏 **- Emacs Lisp**	• `<f11> z l` • `<f12> z`	(**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. • `<f12> z` is only available in buffer in emacs-lisp-mode.

Open a Common Lisp REPL
 	
pel-use-common-lisp must be on.
 	
See also:
𝔓𝔏 **- Common Lisp**
 	
• From lisp-mode:
	• `<f11> z r L`
	
	• `<f12> z`
		(**pel-cl-repl** &optional N)	Open or switch to Common-Lisp REPL buffer window.
			• Use the Common Lisp REPL selected by the PEL user-options:
			  • SLY when `pel-used-sly' is on and `pel-clisp-ide' is set to sly,
			  • Slime when `pel-use-slime'is on and `pel-clisp-ide' is set to slime,
			  • the inferior lisp mode otherwise.
			• The behaviour of the command is affected by the optional argument N:
			  • with no buffers running REPL:
			  • N is nil or absent: open REPL in current window
			  • N is positive: open REPL in other window
			  • N is negative: create new REPL in current window
			  • with 1 or more REPL already running (if more than 1, prompt for one)
			  • if selected buffer is inside an opened window: switch to that window
			  • if selected buffer is not in an opened window:
			  • N is nil or absent: open REPL in current window
			  • N is positive: open REPL in other window
			  • N is negative: create new REPL in current window.

Description	Keystroke	Function	Note
Elixir Shell :IEx See also: **𝔓𝔢𝔩 - Elixir**	`<f11> z r 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 it when **pel-use-elixir** and **pel-use-alchemist** user-options are both set to **t**.
Start Erlang Shell See also: **𝔓𝔢𝔩 - Erlang**	• `<f11> z r e` • `C-c C-z` • `<f12> 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> z 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: **𝔓𝔢𝔩 - Forth** • From Forth buffer:	`<f11> z r f` `<f12> z`	(**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)
Start Haskell Shell See also: **𝔓𝔢𝔩 - Haskell** • From buffer	`<f11> z r h` `<f12> z`	(**run-haskell**)	Show the inferior-haskell buffer. Start the process if needed. 📦 Requires the **haskell-mode** and Haskel installed. PEL activates this when the **pel-use-haskell and the pel-use-haskell-mode** user-options are set to **t**.
Start Javascript REPL See also: **𝔓𝔢𝔩 - Javascript**	`<f11> z r i` `<f12> z`	(**js-comint-repl** &optional CMD)	Start a NodeJS REPL process. • Optional CMD will override 'js-comint-program-command' and 'js-comint-program-arguments', as well as any nvm setting. • When called interactively use a universal prefix to set CMD. 📦 Requires the **js-comint** Emacs package and node installed. PEL activates this when **pel-use-js** is activated and **pel-use-js-comint** user option is set to **t**.
Start Julia Shell See also: **𝔓𝔢𝔩 - Julia** • From Julia buffer:	`<f11> z r j` `<f12> z`	(**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)** • From LFE buffer:	`<f11> z r C-l` `<f12> z`	(**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**.
Lua Shell See also: **𝔓𝔢𝔩 -Lua** • From Lua buffer:	`<f11> z u` `<f12> z`	(**pel-lua-repl**) • (**lua-start-process** &optional NAME PROGRAM STARTFILE &rest SWITCHES) • (**lua-ts-inferior-lua**)	Run a Lua interpreter in an inferior process. The actual command used depends on whether **pel-use-tree-sitter** is on and the value of **pel-lua-repl-used** user-option. • The command provided by the lua-mode is used when pel-use-tree-sitter is nil or when **pel-lua-repl-used** value is always-use-lua-mode-repl: **lua-start-process** • This provide the most control: • Start a Lua process named NAME, running PROGRAM. • PROGRAM defaults to NAME, which defaults to '**lua-default-application**'. • The real command provided by lua-ts-mode is used otherwise.
Start OCaml Shell See also: • From OCaml buffer	`<f11> z r o` `<f12> z`	(**run-ocaml**)	Run an OCaml REPL process. I/O via buffer '*OCaml*'. 📦 Requires the **tuareg** external package. PEL activates this when the **pel-use-ocaml** and the **pel-use-tuareg** user-options are set to **t**.
Start Perl REPL See: **𝔓𝔢𝔩 - Perl**	`<f11> z r P` `<f12> z`	(**perl-repl**)	Run a Perl REPL in a *Perl-REPL* buffer. 📦 Requires the **perl-repl** external package activated by **perl-use-perl-repl** user-option. • The **perl-repl-file-path** user option specifies the name of the Perl REPL program, which may optionally specify the explicit file path. • PEL provides the **perl-repl** shell script which uses the Perl command line.
Start Python Shell See also: **𝔓𝔢𝔩 Python** • From Python buffer:	`<f11> z r p` `<f12> z`	(**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.
Start Chez Scheme Shell See also: • From Chez buffer	`<f11> z r C-z` `<f12> z`	(**pel-chez-repl** &optional N)	Run the Chez REPL in window specified by N. • By default use the other window. If a numeric argument is specified, its value correspond to the direction of a numeric keypad: 8 4 6 2 That is: • 8: up • 4: left • 6: right • 2: down • 0 and 5 identify the current window. Requires the Chez Scheme installed. PEL activates it when the **pel-use-chez** is set to **t**.
Start Chibi Scheme Shell See also: • From Chibi buffer	`<f11> z r C-i` `<f12> z`	(**pel-chibi-repl** &optional N)	Run the Chibi REPL in window specified by N. • See 'pel-chez-repl' for complete description. Requires the Chibi Scheme installed. PEL activates it when the **pel-use-chibi** is set to **t**.
Start Chicken Scheme Shell See also: • From Chicken buffer	`<f11> z r C-k` `<f12> z`	(**pel-chicken-repl** &optional N)	Run the Chicken REPL in window specified by N. • See 'pel-chez-repl' for complete description. Requires the Chicken Scheme installed. PEL activates it when the **pel-use-chicken** is set to **t**.
Start Gambit Scheme Shell See also: **𝔓𝔢𝔩 - Gambit Scheme** • From Gambit buffer	`<f11> z r C-b` `<f12> z`	(**pel-gambit-repl** &optional N)	Run the Gambit Scheme REPL in window specified by N. • See 'pel-chez-repl' for complete description. 📦 Requires the **gambit.el** file and Chicken Scheme installed. PEL activates it when the **pel-use-gambit** is set to **t**.

Description	Keystroke	Function	Note
Start Gerbil Scheme Shell See also: **PEL - Gerbil Scheme** • From Gerbil buffer	`<f11> z r C-e` `<f12> z`	(**pel-gerbil-repl** &optional N)	Run the Gerbil REPL in window specified by N. • See 'pel-chez-repl' for complete description. 📦 Requires the **gerbil-mode** external package and Gerbil Scheme installed. PEL activates it when the **pel-use-gerbil** is set to **t**.
Start Guile Shell • From Guile buffer	`<f11> z r C-g` `<f12> z`	(**pel-guile-repl** &optional N)	Run the Guile REPL in window specified by N. • See 'pel-chez-repl' for complete description. Requires Guile Scheme installed. PEL activates it when the **pel-use-guile** is set to **t**.
Start MIT/GNU Scheme Shell • From MIT/GNU Scheme buffer	`<f11> z r C-m` `<f12> z`	(**pel-mit-scheme-repl** &optional N)	Run the MIT/GNU Scheme REPL in window specified by N. • See 'pel-chez-repl' for complete description. Requires MIT/GNU Scheme Scheme installed. PEL activates it when the **pel-use-mit-scheme** is set to **t**.
Start Racket Shell See also: **PEL - Racket** • From Racket buffer	`<f11> z r C-r` `<f12> z`	(**pel-racket-repl** &optional N)	Run the Racket REPL in window specified by N. • See 'pel-chez-repl' for complete description. 📦 Requires the **racket-mode** external package and Racket installed. PEL activates it when the **pel-use-racket** is set to **t**.
Start Scsh Scheme Shell • From Scsh buffer	`<f11> z r` `<f12> z`	(**pel-scsh-repl** &optional N)	Run the Scsh REPL in window specified by N. • See 'pel-chez-repl' for complete description. Requires Scsh Scheme Scheme installed. PEL activates it when the **pel-use-scsh** is set to **t**.

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
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	
explainshell.com	Online application where you can type a shell command: the app explains each argument. Very useful.