Emacs support for Ruby 🚧

Description	Keystroke	Function	Note
Ruby Editing Last updated on:	Emacs provides the built-in **ruby-mode** and **ruby-ts-mode** to support Ruby programming. ☑ PEL supports it when **pel-use-ruby** user options is turned on. • On **Emacs >= 30**, PEL supports tree-sitter if **pel-use-tree-sitter** is set to t. • You can activate tree-sitter for Ruby by setting **pel-use-ruby** to 'with-tree-sitter (as long as **pel-use-tree-sitter** is t and Emacs >= 30). See Σ **Tree Sitter** and 🚦**Tree-sitter** • The Ruby files (.rb and several other extensions) are recognized as Ruby source files and use the ruby-mode or ruby-ts-mode according to **pel-use-ruby** value. • Speedbar support for .ruby files listing functions and types. See Σ **Speedbar** for more info about it. • When it is turned on the **`<f11> SPC U`** prefix is made available. In a ruby buffer these commands are accessible via the **`<f12>`** key. It also activates the ability to activate minor modes for the ruby major mode through the PEL **pel-ruby-activates-minor-modes** user-option. 🚧 PEL support for Ruby is not complete. More commands should be provided and documented. Ruby support is preliminary. 2025-10-15		

Open this PDF file.
See also: Σ **Help/Info**
	`<f11> SPC U <f1>`
	`<f12> <f1>`
		(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the 𝕻𝔩 **- Ruby** 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 Ruby support	**`<f11> SPC U <f2>`**
	`<f12> <f2>`
		(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL Ruby support.
			• If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.

Σ **Customize** Emacs Ruby support	**`<f11> SPC U <f3>`**
	`<f12> <f3>`
		(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Ruby support: ruby.
			• If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.

Select ruby-mode for extension-less file ☝️The **`<f12>`** key is available only until a PEL controlled major mode is activated. Then it becomes a buffer prefix key.	**`<f12>`**	(**pel-as** &optional FORCE)	Inside a fundamental-mode buffer, interactively select major mode for the buffer. Re-do it with arg. ☝️see **Create extension-less executable scripts with PEL**.
	This command is mostly used to set the major mode of a buffer in fundamental-mode', when the **`<f12>`** key binding is available for it. After being used once in a buffer the major mode is selected and the PEL key binding will not be available when PEL supports the major mode. For Ruby file, select **ruby.** It will insert a shebang line specified by ☑ **pel-ruby-shebang-line** user option. PEL defines the (as &optional FORCE) alias unless ☑ **pel-has-alias-as** user-option is set to nil. You can use **`M-x as`** to invoke it.		

Show PEL setup for Ruby	**`<f12> ?`**
	`<f11> SPC U ?`
		(**pel-ruby-setup-info** &optional APPEND)	Display Ruby setup information inside a *pel-ruby-info* buffer with buttons providing quick access to the customization buffer of each variable shown. The information shown includes the value and interpretation of:
			• pel-use-ruby (whether the classic or tree-sitter based major mode is used).
			• the user options controlling indentation and hard tab width rendering.
			To append information in the buffer instead of clearing the previous content type any prefix argument (such as **`C-u`**) before the command keystroke.

Comments			
Toggle display of comments in buffer or active region See also: Σ **Comments**	**`<f11> ; ;`**	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see Σ **Comments**). ☑ PEL activates it when the **pel-use-hide-comnt** user option is **t**.
Generic code skeletons • **tempo skeletons** See also: • Σ **Inserting Text** • **T Templates**	Several mechanisms have been developed to allow easy insertion of predefined text in Emacs. ⚠️ PEL does not yet define skeletons for Ruby. You can use the generic one. • Emacs provides the built-in skeleton mechanism and the **tempo skeletons**. • PEL supports both. They are used a little bit differently. PEL provides **generic** tempo skeletons you can use for Ruby until PEL adds Ruby-specific skeletons. • PEL provides key bindings to the tempo skeletons: the generic code templates, accessible via the **`<f6>`** prefix key, and the language-specific code templates, accessible via the **`<f12>`** key prefix.		

Σ **Customize** **PEL Text Insertions control for Ruby code skeletons.**	**`<f6> <f2>`**
	`<f12> <f12> <f2>`
		(**pel-customize-pel** &optional OTHER-WINDOW)
		(**pel-customize-generic-skels** &optional OTHER-WINDOW)
			Open the customization groups that control the format of the various skeletons including the generic skeleton used by the **`<f6> h`** key and the **`<f12><f12> h`** key (see below).
			• If OTHER-WINDOW is non-nil (use **`C-u`**), display in other window.

Insert generic file module header block — Language agnostic
 	
After inserting the template, navigate though areas that must be filled with:
• forward: **`C-c .`**
• backward: **`C-c ,`**
	`<f6> h`
	`<f12> <f12> h`
		(**pel-generic-file-header**)	Insert a file header block at the top of the file. Works only for buffer visiting a file.
			⚠️ The command key binding **`<f6> h`** is available only 1 second after Emacs has started.
			⚠️ As mentioned above PEL does not yet define Ruby-specific skeletons, this uses the generic one.

	☝️Specify the format of the header via the user-options in the **pel-pkg-generic-code-style** customization group accessible via **`<f6> <f2>`** • Inside a **Ruby** buffer, **`<f12> <f2>`** provides access to the following customization groups: ☝️After inserting a template, use **tempo-forward-mark** and **tempo-backward-mark** to move to the beginning of each section that must be filled.		

Toggle pel-tempo-mode	**`<f6> SPC`**
	`<f12> <f12> SPC`
		(**pel-tempo-mode** &optional ARG)	Toggle PEL tempo mode on/off.

	PEL tempo mode activates **`C-c .`** and **`C-c ,`** as well as to **`C-c C-.`** and **`C-c C-,`** key bindings to navigate across tempo mark hot-spots. When pel-tempo-mode is active the pel-tempo-mode lighter (‡) is shown on the status bar. The second set of keys are only available in graphics mode. ☝️The pel-generic-file-header command inserts the text using a tempo skeleton: the PEL tempo mode is automatically activated by typing **`<f6> h`**.		

Expand any tag in template
 	
Note: PEL default skeleton does not use tags.
	`<f6> <f12>`
	`<f12> <f12> <f12>`
		(**tempo-complete-tag** &optional SILENT)	Look for a tag and expand it. All the tags in the tag lists in '**tempo-local-tags**' (this includes 'tempo-tags') are searched for a match for the text before the point. The way the string to match for is determined can be altered with the variable 'tempo-match-finder'. If 'tempo-match-finder' returns nil, then the results are the same as no match at all.
			• If a single match is found, the corresponding template is expanded in place of the matching string.
			• If a partial completion or no match at all is found, and SILENT is non-nil, the function will give a signal.
			• If a partial completion is found and 'tempo-show-completion-buffer' is non-nil, a buffer containing possible completions is displayed.

Ruby-mode control			
Toggle string literal quoting	**`C-c '`**	(**ruby-toggle-string-quotes**)	Toggle string literal quoting between single and double.
Toggle block type	**`C-c {`**	(**ruby-toggle-block**)	Toggle block type from do-end to braces or back. • The block must begin on the current line or above it and end after the point. • If the result is do-end block, it will always be multiline.

Description	Keystroke	Function	Note
Navigation	The following navigation commands are specialized for Ruby and complement what is described in the 𝛴 **Navigation** section.		
• **by block**	The following commands move point through Ruby code blocks		
Move forward to end of current block	`C-M-n`	(**ruby-end-of-block** &optional ARG)	Move forward to the end of the current block. • With ARG, move out of multiple blocks.
Move backward to beginning of current block	`C-M-p`	(**ruby-beginning-of-block** &optional ARG)	Move backward to the beginning of the current block. • With ARG, move up multiple blocks.
Move forward down one nested level	`C-M-d`	(**smie-down-list** &optional ARG)	Move forward down one level paren-like blocks. Like ‘down-list’. • With argument ARG, do this that many times. • A negative argument means move backward but still go down a level. • This command assumes point is not in a string or comment.
Go up in the block hierarchy	• `C-M-u` • `C-M-<up>` • `C-[ C-u` • `Esc C-u` • `Esc C-<up>`	(**backward-up-list** &optional ARG ESCAPE-STRINGS NO-SYNTAX-CROSSING)	Move backward out of one level of parentheses. • This command will also work on other parentheses-like expressions defined by the current language mode. With ARG, do this that many times. A negative argument means move forward but still to a less deep spot. If ESCAPE-STRINGS is non-nil (as it is interactively), move out of enclosing strings as well. If NO-SYNTAX-CROSSING is non-nil (as it is interactively), prefer to break out of any enclosing string instead of moving to the start of a list broken across multiple strings. On error, location of point is unspecified.
• **by class/function definition**	The commands move point by function and class definitions. ☝ The `<f6>` cursor key mappings use `<up>` and `<down>` to move to the beginning of the function/class definition, and `<left>` and `<right>` to the end of the function/class definition. ⚠️ These work with function definitions and allow moving forward to the end of a class definition, but not backward to the beginning or end of a class definition. 🚧		
Backward to beginning of function definition	• `C-M-a` • `C-M-<home>` • `<f6> <up>` • `C-[ C-a` • `Esc C-a`	(**beginning-of-defun** &optional ARG)	Move backward to the beginning of a defun. • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. ☛ Shift marking is available in graphics mode, not in terminal mode (for **`C-M-a`** and **`C-M-<home>`**). It’s always available for `<f6> <up>` : hold Shift after typing `<f6>`. ⚠️ This command moves to the beginning go the next function or of the same nesting level of the current location. It skips the functions and methods that are more deeply nested.
Forward to end of function and class definition	• `C-M-e` • `C-M-<end>` • `<f6> <right>` • `C-[ C-e` • `Esc C-e`	(**end-of-defun** &optional ARG)	Move forward to next end of defun. With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. ☛ Shift marking is available in graphics mode, not in terminal mode (both keys). ⚠️ This command moves to the end of the next **top-level** function or class. It skips the nested functions and methods.
Forward to start of next function definition	`<f6> <down>`	(**pel-beginning-of-next-defun** &optional SILENT DONT-PUSH_MARK)	Move forward to the beginning of the next function definition. • Beeps if does not find beginning of next function unless SILENT is non-nil. • If the beginning of next function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with ``M-` `` or `<f6><f6>`. ☛ Shift marking is available: hold Shift after typing `<f6>`. ☝ This command complements what end-of-defun does. • It moves forward but not to the end of the function definition (like end-of-defun) but to the beginning of the function definition, which is often what users of other editors expect. • It handles nested functions or class methods in languages like Ruby and others.
Backward to end of previous function definition	`<f6> <left>`	(**pel-end-of-previous-defun** &optional SILENT DONT-PUSH_MARK)	Move backwards to the end of the previous function definition. • Beeps if does not find end of previous function unless SILENT is non-nil. • If the end of previous function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with ``M-` `` or `<f6><f6>`. ☛ Shift marking is available. ☝ This command complements this set of 4 commands. • ⚠️ It handles most nested functions or class methods in Ruby but not always. In some cases it does not move the point. Better logic is needed. 🚧
Search Support	In Python mode, the superword mode can be useful since snake_case is often used. Using superword-mode helps searching. PEL activates the superword mode by default in Python mode. To change this use the `<f11> t <f2>` to access the customize buffer.		
Toggle superword-mode See also: • 𝛴 **Text Modes** • 𝛴 **Search/Replace**	• `<f11> t m p` • `<f12> M-p`	(**superword-mode** &optional ARG)	Toggle superword-mode: a minor mode that treats snake_case as one word. In Ruby ‘_’ are treated as part of words. • With a prefix argument ARG, enable superword mode if ARG is positive, and disable it otherwise. • PEL provides the `<f12> M-p` key for the programming language modes where snake_case is popular (Emacs Lisp, C, C++, Erlang, Python, Ruby, etc…)
Highlight blocks	The following commands can be used to activate or toggle useful modes to highlight blocks of (), {}, and []. • show-paren-mode, which highlights the parens that matches the one before or after point. • rainbow delimiters mode, where matching nested parens are highlighted with the same colour.		
Toggle show-paren mode on/off See also: 𝛴 **Highlight**	• `<f12> M-9` • `M-<f12> M-9` • `<f11> h (`	(**show-paren-mode** &optional ARG)	Toggle visualization of matching parens (Show Paren mode). • With a prefix argument ARG, enable Show Paren mode if ARG is positive, and disable it otherwise. • Show Paren mode is a global minor mode. When enabled, any matching parenthesis is highlighted in ‘show-paren-style’ after ‘show-paren-delay’ seconds of Emacs idle time.
Enable/Disable coloured highlight of nested blocks (),{},[] See also: 𝛴 **Highlight**	• `<f12> M-r` • `M-<f12> M-r` • `<f11> h R`	(**rainbow-delimiters-mode** &optional ARG)	Highlight nested parentheses, brackets, and braces with different colours according to their depth. • Customize the depth and colours with **M-x customize-group rainbow-delimiters** 📦 **Requires:** rainbow-delimiters.el ☑️ PEL activates this when the **pel-use-rainbow-delimiters** user option is set to **t**.

Description	Keystroke	Function	Note
Indentation	Indent/un-indent lines with following Ruby-specific commands. These complement what is available in the ⅀ **Indentation** section.		
Indent expression after point	`C-M-q`	(**prog-indent-sexp** &optional DEFUN)	Indent the expression after point. • When interactively called with prefix, indent the enclosing defun instead. • Does nothing if indentation is currently correct.
Open the indent-tools hydra See also: ⅀ **Indentation**	• `<f11> <tab> <f7>` • `<f7> <tab>` • `C-c >`	(**indent-tools-hydra/body**)	Activate the body in the "indent-tools-hydra" hydra. 📦 Requires **indent-tools** external package ☑ PEL activates it when the **pel-use-indent-tools** user-option is turned on (set to **t**). 🎛 With PEL, this key binding is only available when: • globally, when **pel-indent-tools-key-bound** is set to **globally**, • in python-mode only when **pel-indent-tools-key-bound** is set to **python**. • The actual key is selected by indent-tools **indent-tools-keymap-prefix** user-option, the default is `C-c >`

See also: ⅀ **Hide/Show**	The heads for the associated hydra are:
	`>:` 'indent-tools-indent',
	`<:` 'indent-tools-demote',
	`E:` 'indent-tools-indent-end-of-defun',
	`c:` 'indent-tools-comment',
	`U:` 'indent-tools-uncomment',
	`P:` 'indent-tools-indent-paragraph',
	`l:` 'indent-tools-indent-end-of-level',
	`K:` 'indent-tools-kill-tree',
	`C:` 'indent-tools-copy-hydra/body',
	`s:` 'indent-tools-select',
	`e:` 'indent-tools-goto-end-of-tree',
	`u:` 'indent-tools-goto-parent',
	`d:` 'indent-tools-goto-child',
	`S:` 'indent-tools-select-end-of-tree',
	`n:` 'indent-tools-goto-next-sibling',
	`p:` 'indent-tools-goto-previous-sibling',
	`i:` 'helm-imenu',
	`j:` 'forward-line',
	`k:` 'previous-line',
	`SPC:` 'indent-tools-indent-space',
	`_:` 'undo-tree-undo',
	`L:` 'recenter-top-bottom',
	`f:` 'yafolding-toggle-element',
	`q:` exit
			`Indent           | Navigation           | Actions`
			`------------------+----------------------+-----------`
			`> indent         | j v                  | K kill`
			`< de-indent      | k ʌ                  | i imenu`
			`l end of level   | n next sibling       | C Copy…`
			`E end of fn      | p previous sibling| c comment`
			`P paragraph      | u up parent          | U uncomment (paragraph)`
			`SPC space        | d down child         | f fold`
			`_ undo           | e end of tree        | q quit`
			
			☝️The **f** key toggles the element folding. Press once to hide the sub-tree, press-again to display it back.

Emacs & Ruby — References

Document	Notes
Ruby Programming Language	• Ruby @ Wikipedia • Ruby Homepage
Notes on Emacs Ruby Support	• Ruby Mode @ Emacs Wiki • Ruby On Rails @ Emacs Wiki
LSP Support	• LSP Mode for Ruby • solargraph @ GitHub
Blogs on adding support for Ruby	• Getting Started with Emacs for Ruby , by Horace William, 07 June 2016. • Ruby and Emacs Tip: Advanced Pry Integration, from Thiago Araújo Silva, Aug 27, 2018 and updated May 11, 2019
Tools for Ruby	
• **Pry - an alternative for the Ruby IRB shell**	• Pry @ GitHub • Pry Home Page