↖ Topic Index

YAML Data Serialization Support

Operation	Keystroke	Function	Note
Editing YAML files ○ Help & Customization • Syntax check • Highlight Column • Indentation / Text Folding • indent-tools ○ Smartparens • smart-shift ○ YAML Reference	Emacs supports YAML via the built-in **yaml-ts-mode** a Σ **Tree Sitter** mode that PEL supports on **Emacs >= 30.1** only. See also: **Ɱ CWL** You can also use **yaml-mode** external package with provides a classic major-mode that does not require Tree-Sitter support. • PEL installs and activates it when the PEL user-option below is set.		
	📦 **yaml-mode**	**pel-use-yaml**	A classic major mode for YAML files. Set user-option to **t** to install and activate.
	Long YAML files are notoriously difficult to edit properly. The following packages provide useful minor-modes for working with YAML files:		
	📦 **flycheck**	**pel-use-flycheck**	Supports checking the syntax of the YAML file. See Σ **SyntaxCheck**
	📦 **indent-tools**	**pel-use-indent-tools**	Tools to indent, unindent, navigate across indentation levels See Σ **Indentation**
	📦 **smartparens**	**pel-use-smart-parens**	Provide commands to manipulate code in matching pairs. See Σ𝑥 **Smartparens**
	📦 **smart-shift**	**pel-use-smart-shift**	Provide commands to shift (indent/un-indent) block of lines. See Σ **Indentation**
	📦 **vline.el**	**pel-use-vline**	A command that display a vertical line: helps with alignment. See Σ **Highlight**
	pel-yaml-activates-minor-modes Identify the minor modes to activate automatically inside YAML buffers using classic or Tree-Sitter major mode.		
Last updated on:	2026-01-12	• PEL associates the following file extensions with **yaml-mode**: .yml, .yaml, .eyaml, .raml.	
Open this PDF file. See also: Σ **Help/Info**	`<f11> SPC M-y <f1>` `<f12> <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the Ⓓ **YAML** 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 YAML control**	`<f11> SPC M-y <f2>` `<f12> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL YAML support. • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Σ **Customize Emacs YAML control**	`<f11> SPC M-y <f3>` `<f12> <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs YAML support groups: yaml, fly check, indent-tools and smartparens • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Show PEL setup information for the major mode.	`<f11> ? /`	(**pel-mode-setup-info** &optional APPEND)	Display yaml-mode setup information inside a *pel-mode-info* buffer with buttons providing quick access to the customization buffer of each variable shown. • Use any prefix key (such as **C-u**) too append information in the buffer instead of clearing it.
Toggle between classic and Σ Tree Sitter major mode	`<f11> C-t C-t`	(**pel-treesit-toggle-mode**)	Toggle the major mode between the classic mode and the Tree-Sitter based mode. • If the other major mode is not available the command signals a user error.
Flycheck See also: Σ **SyntaxCheck**	📦 The **flycheck** activated by PEL **pel-use-flycheck** is a minor mode for on-the-fly syntax checking.. 🎛 Aside from the following 2 key bindings that PEL provides to toggle the flycheck mode, flycheck key prefix is **C-c !** as set by its **flycheck-keymap-prefix** user-option. You can change it for a different key prefix.		
Toggle flycheck mode for current buffer	`<f11> ! !`	(**flycheck-mode** &optional ARG)	Toggle flycheck minor-mode for the current buffer.
Toggle flycheck mode for all buffers	`<f11> ! M-!`	(**global-flycheck-mode** &optional ARG)	Toggle Flycheck mode in all buffers. • Flycheck mode is enabled in all buffers where 'flycheck-mode-on-safe' would do it.
• **Flycheck buffer/file**			
Syntax Check current buffer	`C-c ! c`	(**flycheck-buffer**)	Start checking syntax in the current buffer. • Get a syntax checker for the current buffer with 'flycheck-get-checker-for-buffer', and start it.
Check syntax of current file	`C-c ! C-c`	(**flycheck-compile** CHECKER)	Run CHECKER via 'compile'. • CHECKER must be a valid syntax checker. Interactively, prompt for a syntax checker to run. • Instead of highlighting errors in the buffer, this command pops up a separate buffer with the entire output of the syntax checker tool, just like 'compile'.
Highlight current column	The following command provide a vertical line across the entire window at the cursor location. • Useful when creating tables or checking indentation manually. • vline also provides the vline-global-mode to activate the vertical line in all buffers; PEL has no binding for it because it slows Emacs too much.		
Toggle Vline Mode See also: Σ **Highlight**	• `<f11> h |` • `<f11> 9`	(**vline-mode** &optional ARG)	Toggle the display of a vertical line spanning the entire window at the cursor column. 📦 Requires: **vline.el** PEL activates it when **pel-use-vline** user option is **t**.
Indented Text Folding	The following command folds (hide or show) all lines that are indented more than the current line. • You can also use the **f** key inside the indent-tools Hydra, shown below, to fold indented sections.		
Toggle hiding lines more indented than current line See also: Σ **Hide/Show**	`<f11> H I`	(**pel-toggle-hide-indent**)	Toggle hiding lines more indented than current line. • Affects the entire buffer. Not syntax sensitive. Can be used anywhere. ⚠ Do not modify the buffer while lines are hidden, it's allowed but its using selective display and you don't see what you change.
Indent-tools PEL activates it when the **pel-use-indent-tools**	The **indent-tools** external package provides a minor-mode with several commands to indent, un-indent, navigate across indented text levels with a key **hydra** 🎛 PEL provide a global key binding to its key **hydra** and provides the ability to activate the proposed key binding globally and for python mode: • **pel-indent-tools-key-bound** : activates the **C-c >** key binding either globally or for python-mode only.		
Open the indent-tools hydra See also: Σ **Indentation**	• `<f11> <tab> <f7>` • `<f7> <tab>` • `C-c >`	(**indent-tools-hydra/body**)	Activate the body in the "indent-tools-hydra" hydra. 🎛 With PEL, these key bindings are 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 >**
The indent-tools hydra provide keys you can use to navigate across the indented YAML elements. See also: Σ **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** ^ | **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. Press once to hide the sub-tree, press-again to display it back.

Operation	Keystroke	Function	Note

Smartparens Mode
• **Smartparens manual**
 	
See also: 𝕀𝑥 **Smartparens**
	Simplify insertion of matching pairs with the **smartparens** minor mode. PEL binds a set of keys, described below, to toggle activation of that mode.
	📦 This uses the **smartparens** external package. PEL activates it when **pel-use-smartparens** is set to **t**.
	• Mode line lighter:
	  • smartparens-mode: SP
	  • smartparens-strict-mode: SP/s		

Help on smartparens	**<f11> (?**	(**sp-cheat-sheet** &optional ARG)	Generate a cheat sheet of all the smartparens interactive functions. Shows inside Emacs buffer. • Without a prefix argument, print only the short documentation and examples. • With non-nil prefix argument ARG, show the full documentation for each function. • You can follow the links to the function or variable help page. • To get back to the full list, use M-x help-go-back. • You can use 'beginning-of-defun' and 'end-of-defun' to jump to the previous/next entry. • Examples are fontified using the 'font-lock-string-face' for better orientation.
Describe user system	**<f11> (M-?**	(**sp-describe-system** STARTERKIT)	Describe user's system. Prompt for starter kit: Evil, Spacemac, Vanilla. • The output of this function can be used in bug reports.
Toggle smartparens mode	**<f11> ((**	(**smartparens-mode** &optional ARG)	Toggle smartparens mode.
Toggle smartparens-strict mode	**<f11> ( )**	(**smartparens-strict-mode** &optional ARG)	Toggle the strict smartparens mode. • When strict mode is active, 'delete-char', 'kill-word' and their backward variants will skip over the pair delimiters in order to keep the structure always valid (the same way as 'paredit-mode' does). This is accomplished by remapping them to 'sp-delete-char' and 'sp-kill-word'. There is also function 'sp-kill-symbol' that deletes symbols instead of words, otherwise working exactly the same (it is not bound to any key by default). • When strict mode is active, this is indicated with "/s" after the smartparens indicator in the mode list
Toggle smartparens mode	**<f11> (M-(**	(**smartparens-global-mode** &optional ARG)	Toggle Smartparens mode in all buffers. • With prefix ARG, enable Smartparens-Global mode if ARG is positive; otherwise, disable it. • Smartparens mode is enabled in all buffers where 'turn-on-smartparens-mode' would do it.
Toggle smartparens-strict mode	**<f11> (M-)**	(**smartparens-global-strict-mode** &optional ARG)	Toggle Smartparens-Strict mode in all buffers. • With prefix ARG, enable Smartparens-Global-Strict mode if ARG is positive; otherwise, disable it. • Smartparens-Strict mode is enabled in all buffers where 'turn-on-smartparens-strict-mode' would do it.

Smart-shift
 	
See also: 𝕀 **Indentation**
	The **smart-shift** external package simplifies shifting a complete line or region of lines right or left but also up or down.
	• It is implemented as a minor or global minor mode that must be enabled first. You can identify the smart-shift-mode inside one of the pel-<mode>-activates-minor-modes user-options to activate it automatically. You can also use the commands manually or through the key bindings provided by PEL to activate the smart-shift-mode in the current buffer or globally for all buffers.
	• PEL controls it through customization user-options:
	  📦 The **smart-shift** external package PEL activates it when the pel-use-smart-shift user-option is turned on (set to t).
	  🎛 PEL also provides the **pel-smart-shift-keybinding** user-option that allows you to select additional alternative key bindings for the smart-shift commands that shift line(s). By default the key bindings are using **C-c** as a key prefix. With PEL you can also use a control key for the cursor or change the prefix key to use the **<f9>** key. The 3 possible key bindings are shown below but only one of them will be available at any given time. The one available is the one selected by the user-option value.		

Toggle smart-shift mode in current buffer	**<f11> <tab> s**	(**smart-shift-mode** &optional ARG)	Activate/de-activate the smart-shift mode in the current buffer. • Activate the line-shift key bindings listed below, in the current buffer. • With PEL, the actual key binding selected for the line shift commands depend on the value of the **pel-smart-shift-keybinding** user-option.
Toggle smart-shift mode globally	**<f11> <tab> S**	(**global-smart-shift-mode** &optional ARG)	• Toggle Smart-Shift mode in all buffers. • With prefix ARG, enable Global Smart-Shift mode if ARG is positive; otherwise, disable it. • Smart-Shift mode is enabled in all buffers where 'smart-shift-mode-on' would do it.
Shift line or region right	• **C-c <right>** • **C-c C-<right>** • **<f9> <right>**	(**smart-shift-right** &optional ARG)	Shift the line or region to the ARG times to the right. ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.
Shift line or region left	• **C-c <left>** • **C-c C-<left>** • **<f9> <left>**	(**smart-shift-left** &optional ARG)	Shift the line or region to the ARG times to the left. ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.
Shift line or region up	• **C-c <up>** • **C-c C-<up>** • **<f9> <up>**	(**smart-shift-up** &optional ARG)	Shift the line or region to the ARG times to the upwards. ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.
Shift line or region down	• **C-c <down>** • **C-c C-<down>** • **<f9> <down>**	(**smart-shift-down** &optional ARG)	Shift the line or region to the ARG times to the downwards ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.

YAML & Emacs — References

Description & URL	Notes
YAML	
YAML @ Wikipedia	Overview, syntax, criticisms
YAML official home page	Links to YAML specification, links to various resources and projects. • YAML 1.2 Specs • YAML 1.1 Specs • YAML 1.0 Specs
YAML Resource sites	• Learn YAML in Y Minutes • Online YAML validator (runs **yamllint.py**) ⚠ No link as the site is not using https. Instead install **yamllint.py** locally and use it on the command line or via Emacs.
StrictYAML	A stricter, type-safe YAML
StrictYAML @ Github	
StrictYAML @ hitchdev (Python libraries)	
RAML	RESTful API Modeling Language : RAML files have the .raml file extension.
	• **RAML @ Wikipedia** • **RAML.org** • RAML Spec @ GitHub

Common Workflow Language
 	
See also: **M CWL**
	Common Workflow Language (CWL) uses a subset of YAML and provides YAML supporting tools.
	• **CWL home page**
	  • CWL User Guide
	  • CWL YAML Guide

Emacs support for YAML	
yaml-mode (major mode for YAML)	• **yaml-mode @ GitHub** • Yaml Mode @ Emacs Wiki
indent-tools	• **indent-tools @ GitLab** • indent-tools @ Melpa
smartparens	The smartparens mode can help deal with data that is within matching pair of characters. • **smartparens @ GitHub** • smartparens documentation
Emacs/YAML Support Articles	
Blogs about YAML editing on Emacs	• The best ways to work with yaml files in Emacs, from Chmouel Boudjnah's blog, 2016-09-07 • Editing ansible files in Emacs, from Enis Özgen, 2017-12-29
General blogs about YAML	• 10 YAML tips for people who hate YAML • BTW, the last tip is: use something else… well… S-expressions are very flexible and powerful.
Using YAML on Github	YAML is used to control the actions of several Continuous Integration/Development systems. Github is one of them.
	• Workflow syntax for GitHub Actions • Github Actions Runner Images : select the os environment • Setup Emacs for Github Actions : select the Emacs version