↖ Topic Index

Syntax Checking Tools

Description	Keystroke	Function	Note

Syntax Checking

Emacs syntax checking can be performed by the built-in **flymake** package or the **flycheck** external package.

- Historically **flymake** appeared first and **flycheck** used to be more versatile and powerful. However the latest version Emacs improved **flymake**. It is also used by the eglot Emacs built-in Language Server Protocol client. Note that eaglet only supports flymake but lsp-mode supports flymake and flycheck.
- ☝ **Emacs 30.1** fixed a shell injection vulnerability in man.el (CVE-2025-1244) that affects **flymake**, preventing it to execute code deemed unsafe by default.
  - Starting with Emacs 30.1, **flymake** will not execute elisp code unless the **trusted-content** customizable user-option identifies it as trustable.
- There is a large set of external package extensions for both **flymake** and **flycheck**. Language specific packages are described in the related pages. The packages listed below provide general feature sets. PEL installs and activates packages when the corresponding **pel-use-** user-option is turned on.

PEL supports both **flymake** and **flycheck**, and implements dispatching commands that use the engine currently used in the buffer.

- PEL uses the <f11> ! key prefix for syntax checking command bindings. The same key bindings are used for PEL dispatching commands common to both engines.

Aside from specific checker, described in the PDF page for their corresponding major mode, PEL can install and activate the following packages when the corresponding user-option is turned on:

📦 **flymake-collection**	**pel-use-flymake-collection**	A collection of checkers for **flymake** , replacing a lot of flymake specific extensions.
📦 **flycheck**	**pel-use-flycheck**	Syntax checking for Emacs, an alternative to the built-in flymake.
📦 **flycheck-eglot**	**pel-use-flycheck-eglot**	Flycheck support for eglot, which by default, only supports flymake.
📦 **flycheck-inline**	**pel-use-flycheck-inline**	Display flycheck detected diagnostics at the end of the affected line.
📦 **flycheck-projectile**	**pel-use-flycheck-projectile**	Project-wide flycheck diagnostics.

PEL provide the following 3 user-options you can use to identify whether you want syntax checking activated automatically for specific major modes, for specific files or directories and which syntax checking engine is used, **flymake** or **flycheck**, if any.

pel-fly-engine-for-mode	An list that maps the major mode name to the syntax checking engine to use for the major mode: **flymake** and **flycheck**. By default, it associates emacs-lisp to **flymake**. • If a major mode is associated with nil, then automatic activation of syntax checking for that mode is disabled, regardless of the values of **pel-modes-activating-syntax-check** and **pel-auto-activatefly-engines-in-files**.
pel-modes-activating-syntax-check	A list of major modes that automatically activate the selected syntax checker.
pel-files-activating-syntax-check	A list of file or directories where the syntax checking engine should be activated automatically when the file opens in the Emacs buffer. There are none by default.

☝ Syntax checking can also be activated by **eglot** or **lsp-mode**.

⚠ When using syntax checking without a Language Server, flymake and flycheck back-ends are often programs that check the content of the corresponding ***file***, instead of the buffer. This means that errors can only be detected once the buffer is saved into a file. This is not the case for all major modes.

More info in the pages of the following major modes:

- 𝔏𝔓𝔏 - Emacs Lisp
- 𝔓𝔏 - Erlang
- 𝔓𝔏 - Go
- 𝔓𝔏 - UNIX Shell.

Other modes support it but they are not yet well documented. 🚧 This includes:

- 𝔓𝔏 - Odin
- 𝔓𝔏 - Rust

Last updated on: 2025-12-12

Open this PDF file. See also: **Help/Info**	`<f11> ! <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the SyntaxChecklocal 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.
Customize PEL syntax checking control	`<f11> ! <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL support for: syntax checking. • If OTHER-WINDOW is non-nil (use `C-u`), display in other window.
Customize Emacs syntax checking control	`<f11> ! <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs spelling support. Opens the following customization groups: flymake, flymake-collection, flycheck, flycheck-eglot, flycheck-inline, flycheck-projectile.

PEL Common Syntax Checking

PEL provides the following dispatching commands that use the **flymake** and **flycheck** supporting command depending which one is being use. If you have a checker for both engine in the current major mode you can also use the command to switch engine.

Show syntax check setup information	`<f11> ! ?`	(**pel-fly-show-setup-info** &optional APPEND)	Print syntax check tools setup information in specialized buffer: *pel-fly-info*. • Lists all relevant user-options and their values. Each one is a button leading to more information and to its customization buffer. • If APPEND is non-nil, append information to the buffer, otherwise clear it and display information from the top.
Toggle between flymake and flycheck	`<f11> ! E`	(**pel-fly-toggle-engine**)	Toggle between using Flymake or Flycheck when one is used.
Toggle activation of the syntax checker engine in the current buffer	`<f11> ! !`	(**pel-fly-toggle-syntax-check** &optional *GLOBALLY*)	Toggle the current syntax check engine (**flymake** or **flycheck**) on/off. • The engine is first selected by the value of **pel-fly-engine-for-mode** user-option for the current major mode and whether **flycheck** is available (🗹 available when **pel-use-flycheck** is turned on). • It changes the buffer local state of the syntax check. • You can also toggle the global state of flycheck with the optional GLOBALLY parameter. That parameter is ignored for flymake.
Toggle inline display of diagnostics	`<f11> ! I`	(**pel-fly-toggle-diag-at-eol** &optional ONLY-ERROR)	Toggle diagnostics display at end of line. • If optional ONLY-ERROR is specified affect only the display of errors when activating otherwise activate display of all diagnostics at the end of line. This works fine with **flymake**. For **flycheck** it 📦 requires **flycheck-inline** 🗹 activated by **pel-use-flycheck-inline.** ⚠ When using **flycheck-inline** do NOT disable it while point is over an error message. It won't be removed.
List all diagnostics inside a specialized buffer	`<f11> ! L`	(**pel-fly-list-diagnostics** &optional ALL-FILES)	List all syntax diagnostics in a specialized buffer. • If ALL-FILES optional prefix used, list diagnostics of all project files. Show a list of diagnostics of current buffer inside a *Flymake diagnostic.* or *Flycheck diagnostic* buffer. • The following keys are available in those buffers:

<	beginning of buffer	**>**	end of buffer	**S**	Sort on current column
p	previous line	**n**	next line	**SPC**	Show diagnostic in affected buffer
S-TAB	previous line in description column	**TAB**	next line in description column	**RET**	Move to diagnostic in affected buffer
{	Narrow the width of current column	**}**	Increase thee width of current column	**g**	Refresh buffer
				q	Quit

Go to next flymake/ flycheck diagnostic	`M-n`	(**flymake-goto-next-error** &optional N FILTER INTERACTIVE)	Move point to the next Flymake diagnostic. • With a prefix arg, skip any diagnostics with a severity less than ':warning'. • Display the error message in the echo line.
		(**flycheck-next-error** &optional N RESET)	Visit the N-th error from the current point. • N is the number of errors to advance by, where a negative N advances backwards. With non-nil RESET, advance from the beginning of the buffer, otherwise advance from the current position.
Go to previous flymake/ flycheck diagnostic	`M-p`	(**flymake-goto-prev-error** &optional N FILTER INTERACTIVE)	Move point to the previous Flymake diagnostic. • With a prefix arg, skip any diagnostics with a severity less than ':warning'. • Display the error message in the echo line.
		(**flycheck-previous-error** &optional N)	Visit the N-th previous error. • If given, N specifies the number of errors to move backwards by. • If N is negative, move forwards instead.

Description	Keystroke	Function	Note
Flycheck	Flycheck is a minor mode for on-the-fly syntax checking. 📦 The **flycheck** external package is activated by PEL when **pel-use-flycheck** user-option is turned on or another activated PEL user-option requires it. 🎛 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. ☝ Type **<f10>** to open the **menu bar** and navigate to Tools/Syntax Checking for the **list of flycheck commands from the menu**.		
• **Info about Flycheck**	The following key bindings are available when flycheck-mode is active.		
Open Flycheck manual	`C-c ! i`	(**flycheck-manual**)	Open the Flycheck manual.
Display Flycheck version	`C-c ! V`	(**flycheck-version** &optional SHOW-VERSION)	Get the Flycheck version as string. • If called interactively or if SHOW-VERSION is non-nil, show the version in the echo area and the messages buffer. • The returned string includes both, the version from package.el and the library version, if both a present and different. • If the version number could not be determined, signal an error, if called interactively, or if SHOW-VERSION is non-nil, otherwise just return nil.
• **Flycheck setup**	The following key bindings are available when flycheck-mode is active.		
Display documentation about syntax checker	`C-c ! ?`	(**flycheck-describe-checker** CHECKER)	Display the documentation of CHECKER. • CHECKER is a checker symbol. • Pop up a help buffer with the documentation of CHECKER.
Select Flycheck Checker for current buffer	`C-c ! s`	(**flycheck-select-checker** CHECKER)	Select CHECKER for the current buffer. • CHECKER is a syntax checker symbol (see 'flycheck-checkers') or nil. In the former case, use CHECKER for the current buffer, otherwise deselect the current syntax checker (if any) and use automatic checker selection via 'flycheck-checkers'. • If called interactively prompt for CHECKER. With prefix arg deselect the current syntax checker and enable automatic selection again. • Set 'flycheck-checker' to CHECKER and automatically start a new syntax check if the syntax checker changed. • CHECKER will be used, even if it is not contained in 'flycheck-checkers', or if it is disabled via 'flycheck-disabled-checkers'.
Verify Flycheck setup ☝Identifies available checkers for the major-mode.	`C-c ! v`	(**flycheck-verify-setup**)	Check whether Flycheck can be used in this buffer. • Display a new buffer listing all syntax checkers that could be applicable in the current buffer. • For each syntax checkers, possible problems are shown. ☝Use this to identify the various checkers available for the current major mode and find how to install the missing ones.
Disable Flycheck checker	`C-c ! x`	(**flycheck-disable-checker** CHECKER &optional ENABLE)	Interactively disable CHECKER for the current buffer. • Prompt for a syntax checker to disable, and add the syntax checker to the buffer-local value of 'flycheck-disabled-checkers'. • With non-nil ENABLE or with prefix arg, prompt for a disabled syntax checker and re-enable it by removing it from the buffer-local value of 'flycheck-disabled-checkers'.
• **Flycheck buffer/file**	The following key bindings are available when flycheck-mode is active.		
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'.
• **Manage Errors**	The following key bindings are available when flycheck-mode is active.		
Show error list for current buffer	• `C-c ! l` • `<f11> ! l`	(**flycheck-list-errors**)	Show the error list for the current buffer.
Display all errors at point	`C-c ! h`	(**flycheck-display-error-at-point**)	Display all the error messages at point.
Explain error at point	`C-c ! e`	(**flycheck-explain-error-at-point**)	Display an explanation for the first explainable error at point. • The first explainable error at point is the first error at point with a non-nil ':error-explainer' function defined in its checker. The ':error-explainer' function is then called with this error to produce the explanation to display.
Copy errors	`C-c ! C-w`	(**flycheck-copy-errors-as-kill** POS &optional FORMATTER)	Copy each error at POS into kill ring, using FORMATTER. • FORMATTER is a function to turn an error into a string, defaulting to 'flycheck-error-message'. • Interactively, use 'flycheck-error-format-message-and-id' as FORMATTER with universal prefix arg, and 'flycheck-error-id' with normal prefix arg, i.e. copy the message and the ID with universal prefix arg, and only the id with normal prefix arg.
Clear all errors	`C-c ! C`	(**flycheck-clear** &optional SHALL-INTERRUPT)	Clear all errors in the current buffer. • With prefix arg or SHALL-INTERRUPT non-nil, also interrupt the current syntax check.
Move point to next error	• `C-c ! n` • `M-n`	(**flycheck-next-error** &optional N RESET)	Visit the N-th error from the current point. • N is the number of errors to advance by, where a negative N advances backwards. With non-nil RESET, advance from the beginning of the buffer, otherwise advance from the current position.
Move point to prior error	• `C-c ! p` • `M-p`	(**flycheck-previous-error** &optional N)	Visit the N-th previous error. • If given, N specifies the number of errors to move backwards by. • If N is negative, move forwards instead.
Using Flymake	Flymake performs syntax checking while the user is editing. PEL provides flymake support for some major modes. 🎛 Flymake has several customizable variables, which some listed here: The following customization variables determine the exact circumstances whereupon Flymake decides to initiate a check of the buffer: • **flymake-start-on-flymake-mode** : **t** to start checking when flymake-mode is started. **nil** to prevent check. • **flymake-no-changes-timeout** : time to wait after last change to start checking. Default = 0.5 seconds. • **flymake-start-syntax-check-on-newline** : **t** to check after insertion or removal of newline char from buffer. **nil** to prevent check. The following variable control navigation to next or previous error: • **flymake-wrap-around** : If non-nil, moving to errors wraps around buffer boundaries. • **flymake-diagnostic-types-alist** : Alist ((KEY . PROPS)*) of properties of Flymake diagnostic types. See Emacs documentation for more info.		
Toggle Flymake mode on/off explicitly	`M-x flymake-mode`	(**flymake-mode** &optional ARG)	Toggle Flymake mode on or off. • With a prefix argument ARG, enable Flymake mode if ARG is positive, and disable it otherwise. • Flymake is an Emacs minor mode for on-the-fly syntax checking. • Flymake collects diagnostic information from multiple sources, called backends, and visually annotates the buffer with the results.

Syntax Checking Tools— References

Topic & link	Description
Flymake	
GNU Flymake Manual	Flymake is part of Emacs. It has its own manual.
Flycheck	
Spotlight: Flycheck, a Flymake replacement	Flycheck description by Mickey Petersen
Flycheck home page	
Flycheck supported languages	List of programming and markup languages supported by Flycheck.
Modern Emacs setup for Erlang (with autocompletion and lint)	LambdaCat December 2015 blog, which describes how to use Flycheck for Erlang.