↖ Topic Index

Key-Chords & Key-Seq

Action	Keystroke	Function	Note

Two Characters
• **key-chord**, or
• **key-seq**
 	
See also: ∑ **Customize**
	PEL supports the use of key chords: two keys typed at the same time to invoke a specific command.
	📦 This requires the **key-chord** external package ↗ PEL activates it with the **pel-use-key-chord user-option** set to **t** or to **use-from-start**,
	• PEL activates the global and mode-specific key chord bindings identified in the **pel-key-chords** user option.
	• If **pel-use-key-chord** is set to **use-from-start** it activates the key-chords when Emacs starts, otherwise you must first activate the key-chord mode with the key-chord-mode command which PEL maps to the keystroke **<f11> <f5> k k**.
	📦 The **key-seq** external package ↗ activated by **pel-use-key-seq** user option set to **t**, allows creation of key-chords as key-seq sequences.
	  • The key-seq sequences impose a key order for detection which might help fast typists: if you define "4r" as you key-seq sequence it will only trigger the action if you type '4' then 'r' quickly. Typing the 'r' then the '4' quickly will not trigger the action.
	  • ⚠️ Note, however that key sequences defined with key-seq must only use ASCII characters in the decimal range of [32,126]. This means you cannot use control characters in key-seq sequences.
	• For key-chord sequences you can use ASCII control characters; to include them in the 2 character sequence when editing you key-chord in the pel-key-chords, type C-q followed by the control key. For example type C-q C-i to insert a tab.
	🎛 PEL provides a set of pre-defined key-chords in the **pel-key-chords** user option and maps the **<f11> <f5> k <f2>** to quickly access the PEL key-code customize buffer. and edit these values. You can add, delete or edit any of the provided key-chords, which provide examples of the ways to define your own key-chords. The list of key-chords PEL pre-defines and provides as default are show in the rows below.
	• A key chord is a group of 2 normal, non-modifier keys that must be typed simultaneously to activate the action identified in the key chord definition.
	  • Here, we are not talking of something like the normal Emacs key bindings like C-s, where the Control key and the s key are type together to do a CONTROL-S or where M-b represents using the Meta key and the b key together. The key-chords discussed here allow you to define actions when you type, for example, the key 'j' and the key 'k' together, or when you type the '.' key twice quickly. *When the key-chord-mode is active* these special key-chord events are triggering the action you key-chord definition identifies. If the key-chord-mode is off, you get the normal Emacs behaviour of inserting the two keys inside the current buffer at point location.
	🎛 PEL also provides the following control user options for key-chords and key-seq:
	• **pel-key-chord-two-keys-delay**: Max time delay between two key press to be considered a key chord.
	• **pel-key-chord-one-key-delay**: Max time delay between 2 press of the same key to be considered a key chord. This should normally be a little longer than `key-chord-two-keys-delay'.
	• **pel-key-chord-in-macros:** If nil, don't expand key chords when executing keyboard macros. If non-nil, expand chord sequences in macros, but only if a similar chord was entered during the last interactive macro recording. (This carries a bit of guesswork. We can't know for sure when executing whether two keys were typed quickly or slowly when recorded.)
	⚠️ Switching input-method (as described in ∑ **Input Method**) prevents key-chord from working properly.
	  • To fix the problem: toggle the **key-chord-mode** off and back on.		

Open this PDF file. See also: ∑ **Help/Info**	**<f11> <f5> k <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the ∑ **Key-Chords** 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.
∑ **Customize PEL key-chord control**	**<f11> <f5> k <f2>**	(**pel-cfg-pkg-key-chord** &optional OTHER-WINDOW)	Customize PEL Key Chord support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
∑ **Customize Emacs key-chord control**	**<f11> <f5> k <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs support for: key-chord • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
PEL Key-chords	The following rows describe the key-chords PEL defines by default in the **pel-key-chords** user option. • You can use them when the key-chord-mode is active. • You can also decide to change them if they do not suit you, delete or add new ones by customizing the pel-key-chords user option. PEL provides a key binding to quickly access the customize buffer for key-chord control: **<f11> <f2> P M-K** • The **pel-key-chords** user option has complete docstring that describes how to add news values.		
Toggle key-chord mode	**<f11> <f5> k k**	(**key-chord-mode** ARG)	Toggle key chord mode. • With positive ARG enable the mode. With zero or negative arg disable the mode. • A key chord is two keys that are pressed simultaneously, or one key quickly pressed twice. 📦 Requires **key-chord** ↗ PEL activates it with **pel-use-key-chord**.
Show state of key-chord mode	• **<f11> <f5> k ?** • **<f11> ? k M-K**	(**pel-key-chord-describe**)	Show state of key-chord-mode. When key-chord mode is on, list key chord bindings in a help buffer. See also: ∑ **Help/Info**

PEL Pre-defined key-chords	• PEL default for **pel-key-chords** are identified in the tables of this document with the characters underlined.
	  • In some cases the key-chord is a simple binding to execute a command or an Emacs Lisp lambda form. In that case the 2 key-chord keys are shown in the keystroke column alone, simply underlined.
	  • In other cases, the key-chord inserts characters and execute commands. In such as case, the 2 key-chord keys are also shown in the keystroke column alone, but instead of describing the function in the function column, the cell shows the key-chord string which represent both the character inserted and the key code for the command.
	  • For example, the key-chord that consist of typing the **<** key and the **>** key together is represented as the **<>** key-chord and the expansion is show as **"<>\C-b"** . The effect is to insert both angle brackets and put point in between, since **C-b** is bound to to command backward-char.
	• The **color of the key-chord** corresponds to the availability of the commands used, if any. A key-chord that depends only on Emacs standard commands or simple characters is therefore shown in black.
	☛ PEL pre-defined key-chords are key-chords, not key-seq. Note that key-seq cannot use tab the way it is used for pel-indent-rigidly below.
	💡 With key-chord or key-seq defined as lambdas, you can pass arguments to the called command, just as any other key binding.
	  • You can control whether a key-chord is allowed in a read-only buffer for example, and/or pass numeric arguments.
	  • PEL uses this ability in the definitions of the key-chords using pel commands but it could be applied to anything defined with a lambda.		

Insert <> and place point between them	**<>**	**<>\C-b**	Global: available in all modes.
Insert [ ] and place point between them	**[]**	**[]\C-b**	Global: available in all modes.
Insert {} and place cursor between	**{}**	**{\n\n}\C-p\C-p**	Available in c-mode and c++-mode
Move to window above	**yu**	(**windmove-up** &optional ARG)	Select the window above the current one. • With no prefix argument, or with prefix argument equal to zero, "up" is relative to the position of point in the window; otherwise it is relative to the left edge (for positive ARG) or the right edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled. **Global**: available in all modes.
Move to window below	**bn**	(**windmove-down** &optional ARG)	Select the window below the current one. • With no prefix argument, or with prefix argument equal to zero, "down" is relative to the position of point in the window; otherwise it is relative to the left edge (for positive ARG) or the right edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled. **Global**: available in all modes.
Move to window at left	**gf**	(**windmove-left** &optional ARG)	Select the window to the left of the current one. • With no prefix argument, or with prefix argument equal to zero, "left" is relative to the position of point in the window; otherwise it is relative to the top edge (for positive ARG) or the bottom edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signalled. **Global**: available in all modes.
	⚠️ This key chord might be problematic for programming or writing English text. Using key-seq will help, as you will have to type the letter 'g' before the letter 'f'. To remove its customize value: use **<f11> <f5> k <f2>** .		

Action	Keystroke	Function	Note
Move to window at right	`jk`	(**windmove-right** &optional ARG)	Select the window to the right of the current one. • With no prefix argument, or with prefix argument equal to zero, "right" is relative to the position of point in the window; otherwise it is relative to the top edge (for positive ARG) or the bottom edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled. **Global**: available in all modes.

Indent rigidly
 	
See also:
• **𝔓𝔢𝔩 - C**
• **𝔓𝔢𝔩 - C++**
• **𝔓𝔢𝔩 - D**
	`<tab>q`	(**pel-indent-rigidly &optional N**)	Indent rigidly the marked region or current line N times.
			• **If a region is marked**, it uses 'indent-rigidly' and provides the same prompts to control indentation changes.
			• **If no region is marked**, it operates on current line(s) identified by the numeric argument N (or if not specified N=1):
			  • N = [-1, 0, 1] : operate on current line
			  • N > 1 : operate on the current line and N-1 lines below.
			  • N < -1 : operate on the current line and (abs N) -1 lines above.
			☝ Command numeric prefix **is available** with the key-chord binding.
			Indent all lines starting in the region.
			• If called interactively with no prefix argument, activate a transient mode in which the indentation can be adjusted interactively by typing **`<left>`**, **`<right>`**, **`<S-left>`**, or **`<S-right>`**.
			These commands activate a transient mode where Emacs prompts for extra keys to control how to indent. Indenting and un-indenting is possible. The capabilities are controlled by the variable *indent-rigidly-map* with by default provides:
			  • **`S-<right>`** indent-rigidly-right-to-tab-stop
			  • **`S-<left>`** indent-rigidly-left-to-tab-stop
			  • **`<right>`** indent-rigidly-right
			  • **`<left>`** indent-rigidly-left
			Typing any other key deactivates the transient mode.
			☝ The **`S-<right>`** and **`S-<left>`** keys indent/de-indent to the next tab-stop position, which is controlled by the **tab-width** user option.

Correct mode at point See also: **∑ Highlight**	`4r`	(**flyspell-correct-word-before-point** &optional EVENT OPOINT)	Pop up a menu of possible corrections for misspelled word before point. • Available when current buffer has flyspell-mode or flyspell-prog-mode enabled. • ⚠ **fci-mode** interferes with pop-up menu displays in terminal-mode, at least with the one used by **flyspell-correct-word-before-point:** the menu lines become all jagged, they do not line up vertically. The problem does not affect Emacs running in graphics mode.

Open file or web-page whose name is at point
 	
★★
 	
See also:
• **∑ File mngt**
• **M reStructuredText**
• **𝔓𝔢𝔩 - C**
• **𝔓𝔢𝔩 - C++**
• **𝔓𝔢𝔩 - Erlang**
	`6y`	(**pel-open-at-point** &optional N)	Open the file, library or the URL, named at point, with potential line & column #s.
			• **Global:** available in all modes with specialization in some modes (C, C++, Erlang, rst)
			☝ This command works generically but is also specialized for some major modes, like C, C++, Erlang, reStructuredText.
			  • See their respective pages for the mode specific features.
			  • When executed from with a buffer in sh-mode, the '=' and ':' characters are used as additional delimiters. Shell variables (such as $HOME) are expanded
			  • In general the command extracts the file (or directory) name (and possibly line and column numbers) from text at point and tries to open the file or directory. If the file is not found then the command searches the file inside a directory tree holding the current file. The root of that directory tree is identified by the presence of a project marker file, one of the file identified in the **pel-project-root-identifiers** user option. Something like .git, .hg, .project or .pel-project by default.
			
			This command extracts the file name to search from text at point. The file name is either surrounded by white space characters or the delimiters listed below.
			• If embedded space(s) are allowed in the filename, then point must be located at the first of the 2 delimiter characters.
			  • These delimiter character can be any of the following: **`"`'|()[]{}<>`** ‘’“”「」〔〕〈〉《》【】〖〗«»‹›❮❯〘〙。
			  • Tab and newline are also delimiter characters.
			  • If embedded space in the file name is not allowed, then the file name must also be enclosed in the above delimiters, the space acts as an extra delimiter, and point can be positioned anywhere between the delimiters.
			• If the string identifies a **URL**, the function opens the page in the systems' default browser.
			• The file name extracted from the file may include glob characters (even though this is not used in a `#include ""` or `#include <>` statement but it can be used elsewhere).
			• Otherwise the command attempts to open the file name with the specified name. If that file does not exists it then proceed to search for it.
			• If the file name is followed by **line and column numbers** the point is moved to that position.

Select prompt method ☛			🎛 When several file names are found, the command lists them and prompts using the method selected by **pel-prompt-read-method** user-option.
			  • The default is a very primitive function implemented by PEL. You can select a more powerful **ivy** prompting instead.
			  • With **ivy** selected PEL will automatically set **pel-use-ivy** to **t** and **Ivy mode** will be installed automatically when you restart Emacs.
			  • Note that the command shows all files found by the specified search method, it does not only use the first one found.
			  • This allows you to detect potential duplication in header file names in large include paths.
			• It prompts for incomplete file names, allowing editing the find file (with completion), search for libraries files (type **1**) according to current file type.

Select target window ☛
 	
See function docstring for more info.
			• **Select target window:**
			  • Without argument:
			  • If file is already opened in a window, move point to that window and to the line column coordinates if specified following the file name at point.
			  • If no window holds that file, select the target window based on the number of editable windows in frame: if 1, split that window and use the new window, if 2: use the other window, if 3 or more, use the current window.
			  • With numeric argument N:
			  • N < 0 : create a new window and use that.
			  • N = 0: use the '*other*' (the next) window.
			  • N = 1, 3, 7or above (excluding 8, 9 and 10): select the target window based on the number of editable windows in frame:
			  • if 1 window: split that window and use the new window,
			  • if 2 windows: use the other window,
			  • if 3 or more windows: use the current window.
			  • N is: 8: up, 2: down, 4:left, 5:current, 6:right.
			  • N is 9: **open the file in the system's browser**,
			  open a directory name at point with directory browsing (eg. macOS Finder, Windows Explorer).
			  • N is 10: open the URL at point in the system's browser.
			  • Selecting Minibuffer, inexistent or dedicated window is not allowed.

Open filename at point in a browser See also: **∑ File mngt**	`6u`	(**pel-browse-filename-at-point**)	Open the file name at point inside the system's browser. • If point is at a directory name, open the systems application that browses directories (eg. macOS Finder, Windows Explorer). ☝ This is the same as using pel-open-at-point with the argument N set to 9. It is easier to type and PEL assigns its own key-chord for it.
Open URL at point in a browser See also: **∑ File mngt**	`7u`	(**pel-browse-at-point**)	Open the URL at point inside the system's browser. ☝ This is the same as using **pel-open-at-point** with the argument N set to 10. It is easier to type and PEL assigns its own key-chord for it.

Action	Keystroke	Function	Note
Search word at point from top of current buffer	`. ;`	(**pel-search-word-from-top** &optional N)	Search word at point from top/bottom of buffer in window identified by N. Global: available in all modes.

- **Search direction**:
  - If N is nil, 0 or larger, perform a search-forward from the top of the buffer in window identified by N.
  - If N is negative: perform a isearch-backward from the bottom of the buffer in the window selected by the absolute value of N.
- **Window selection**:
  - If N is not specified, nil, 1, 3, 7 or 9 and larger: search in current window.
  - If N is 0: : search in other window
  - If N in [2,8] range, search in window identified by the direction corresponding to the cursor in a numeric keypad:

    8 := 'up
    4 := 'left 5 := 'current 6 := 'right
    2 := 'down
- **Temporary word mode toggle**: detecting a '*word*' is affected by the subword-mode and superword-mode. When searching in current buffer, the following values of N temporary toggle the mode when grabbing the word:
  - If N is 7: temporary toggle **subword-mode** to grab the word.
  - If N is 9: temporary toggle **superword-mode** to grab the word.

- Explicitly selecting the minibuffer window, or a non-existing window is not allowed, and search is done in current window.
- Searched word is remembered and can be used again to repeat an interactive search with **`C-s`** or **`C-r`**.
- Position before searched word is pushed on the mark ring.

☝ Using **superword-mode** allows you to search for function names in buffer for programming languages. If you do not want to change the mode but want to search for the word as interpreted by the other state of the mode type the command with N equal to 9: **`M-9 <f11> s .`**

☝ Command numeric prefix **is available** with the key-chord binding.

Key-Chords — References

Topic & Link	Description
Emacs normal key sequences	Emacs supports binding commands to key sequences of your choice. In the sequence you can have keys that are typed along with one or several key modifiers (Control, Meta, Super, Hyper, Alt) and you can use several keys: the first set(s) being used as key prefixes.
Key Sequences @ EmacsWiki	Describes what a normal/standard Emacs key sequence is.
Key-Chord extension package	The key-chord package provides the ability to unbind commands to an event consisting of typing 2 keys simultaneously, those keys not using key modifiers. For example you could bing a command to pressing the '4' and the 'r' key simultaneously.
key-chord @ MELPA	This page shows the doc coming from the key-chord.el. It's where PEL gets the file from.
Key Chords @ Emacs Wiki	Some interesting discussion about key-chord.
key-chord.el @ Emacs Wiki	
Key-Set extension package	The key-seq package builds on key-chord package. It changes the way simultaneous keys are detected and imposes an order to these keys. So for a key-seq of "jk" it only accepts the keys if 'j' is type before and 'k' is typed quickly following it. Depending of your typing skills this may help reduce the unwanted triggers of key chords. PEL supports both key-chord and key-seq to the level where you can define a mix of bindings: they can be key-chord or key-seq bindings.
key-seq @ MELPA	
key-seq @ GitHub	