↖ Topic Index

Navigation

Move Operation	Keystroke	Function	Note
Point Navigation ○ Shift-selection • Move by char • Move using avy • by line ○ to column • within line , by word ○ Move/search - space • Move by syntax, • by block ○ to symbol definition ○ by defun , by URL • by sentence, paragraph • by page ○ to line/buffer top/end • to other window • to compilation error • Recenter window	Emacs provides a large amount of commands for moving **point** (Emacs name for cursor) inside a buffer. • Several are built in Emacs. Others are provided by external packages or by PEL itself. This table list the main generic commands for navigation. PEL provides access and activation of the following external packages that provide extra navigation commands and modes: 📦 The **avy external package** activated when the **pel-use-avy** user option is set to **t**. 📦 The **ace-link external package** activated when the **pel-use-ace-link** user option is set to **t**. ➨ Also see the programming language specific sheets for more information on specialized navigation provided by these modes and the tools they support. • **Shift selection** is supported by some commands, not all. The following symbols are used to identify whether the command supports shifts selection: • ⇟ This command supports shift selection in GUI and terminal mode. • ⇓ This command supports shift selection only in GUI mode. • ⇓ This command supports shift selection in GUI mode and also in terminal mode under some conditions (described in the description cell for the command). • ǂ This command does **not** support shift selection. Sometimes for this you can first set the mark before moving. • Pressing the Shift key when using the key binding for commands that do not show any of these 3 arrows have no impact on the shift selection (and may be inappropriate for the command).		
Last updated on:	2025-05-22	See Σ **Marking** for more information on marking.	
Open this PDF file. See also: Σ **Help/Info**	• `<f11> ? p` • `<f11> p`	(**pel-help-pdf-select** &optional OPEN-WEB-PAGE)	Prompt for a PEL PDF and open it. Type: **Navigation** to open this page. • Supports tab completion.
Σ **Customize** Emacs navigation control	`<f11> <f2> P n 2`	(**pel-cfg-pkg-navigation** &optional OTHER-WINDOW)	Customize Emacs navigation tools support: **avy**. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Σ **Customize** PEL Σ **Completion/Input**	• `<f11> M-c <f2>` • `M-g <f4> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL input completion support: access the customization buffer that holds the PEL user options that activate the input completion packages and the **pel-goto-symbol** command. • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Σ **Customize** PEL imenu support See: Σ **Menus**	`<f11> <f10> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL imenu support. Provides access to: • **pel-imenu-follows-order-p** **pel-use-flimenu** **pel-use-imenu+** • **pel-use-imenu-anywhere** **pel-use-imenu-extra** **pel-use-popup-imenu** • **pel-use-popup-switcher** • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Shift-Selection	If you press and hold the **shift** key while typing a movement command, that sets the mark before moving point (Emacs name for cursor) so that the region extends from the original point to its new position. This **Shift-Selection** is called "*Shift-Marking*" in this document. It is available for only some commands. When running Emacs in Terminal mode, less commands support it. Ability to perform "Shift-Marking" is identified in the description of the commands below.		
Move Point	The following sub-sections describe how to navigate across various types of textual and syntactical entities.		
Move back to last marked location See also: Σ **Marking**	• `M-`` ` • `<f11> . `` • `<f6> <f6>` • `C-u C-SPC`	(**pel-jump-to-mark**)	Move point to current mark, set mark to top of buffer mark-ring, and then rotate the ring (by injecting old mark back at the bottom of mark ring).
			• When using this command in sequence this effectively move point to all previously marked locations. • Same as using the set-mark-command via **C-u C-SPC** (but easier to type and shows a more informative message). • pel-jump-to-mark is simply executing `(set-mark-command 1)` and display a more informative message. • The `M-`` key is often used by Linux desktops to switch applications: `<f6><f6>` is a quick alternative.
• **by character**	Some commands in following group support the bidirectional context: when editing right to left text these commands may move in the reverse direction.		
right/next char ➨ Supports bidirectional context. ⇓	`C-f`	(**forward-char** &optional N)	Move point N characters forward (backward if N is negative). N defaults to 1. N := **numeric arg.** • On reaching end or beginning of buffer, stop and signal error.
right/next char ⇟	`<right>`	(**right-char** &optional N)	Move point N characters to the right (to left if N is negative). N defaults to 1. N := **numeric arg.** • On reaching beginning or end of buffer, stop and signal error.
left/previous char ➨ Supports bidirectional context. ⇓	`C-b`	(**backward-char** &optional N)	Move point N characters backward (forward if N is negative). N defaults to 1. N := **numeric arg.** • On attempt to pass beginning or end of buffer, stop and signal error.
left/previous char ⇟	`<left>`	(**left-char** &optional N)	Move point N characters to the left (right if N is negative). N defaults to 1. N := **numeric arg.** • On reaching beginning or end of buffer, stop and signal error.
Go to a specific char position ǂ	`M-g c`	(**goto-char** POSITION)	Enter a character position, a decimal value identifying the index into the continuous set of characters in the buffer.
• **by character using avy**	When using these commands, type the character(s) where you want to move; avy highlights the target locations with another character: type that character to move to the location. The location can be inside any window. This provides **a very efficient way of moving** the point. • Move back to original location with `M-`` or `<f6><f6>` 📦 These commands require the **avy external package** activated when the **pel-use-avy** user option is set to t.		
Jump to visible char using avy ǂ	• `C-:` • `M-G` • `M-g M-c`	(**avy-goto-char** CHAR &optional ARG)	Jump to the currently visible CHAR. • The window scope is determined by 'avy-all-windows' (ARG negates it).
Jump to visible 2 chars using avy. ǂ	• `C-'` • `M-H` • `M-g M-j`	(**avy-goto-char-2** CHAR &optional ARG)	Jump to the currently visible CHAR1 followed by CHAR2. • The window scope is determined by 'avy-all-windows'. • When ARG is non-nil, do the opposite of 'avy-all-windows'. • BEG and END narrow the scope where candidates are searched.
• **by line**	• In terminal mode C-p and C-n cannot be used in conjunction with Shift for marking. The <up> and <down> cursor can be used with Shift for marking. • When moving up or down, if there is no character in the target line exactly over the current column, the cursor is positioned after the character in that line which spans this column, or at the end of the line if it is not long enough.		
Previous line ⇓ ⇟	`C-p` `<up>`	(**previous-line** &optional ARG TRY-VSCROLL)	Move cursor vertically up ARG lines. • `C-p` : ➨ Shift marking is available in graphics mode, not in terminal mode. • `<up>` : ➨ Shift marking works with this command.
Next line ⇓ ⇟	`C-n` `<down>`	(**next-line** &optional ARG TRY-VSCROLL)	Move cursor vertically down ARG lines. • `C-n` : ➨ Shift marking is available in graphics mode, not in terminal mode. • `<down>` : ➨ Shift marking works with this command.
Go to a specific line in current buffer ǂ	• `M-g g` • `M-g M-g` • ⌘-l	(**goto-line** LINE &optional BUFFER)	Go to LINE, counting from line 1 at beginning of buffer. LINE:= **numeric arg.** • Without a numeric prefix argument, read LINE from the minibuffer.
			• If optional argument BUFFER is non-nil, switch to that buffer and move to line LINE there. If called interactively with **C-u** as argument, BUFFER is the most recently selected other buffer. • Prior to moving point, this function sets the mark (without activating it), unless Transient Mark mode is enabled and the mark is already active. Move back to original location with `M-`` or `<f6><f6>`
Goto line using avy ǂ • **potentially in other window**	• `M-g f` • `M-g l`	(**avy-goto-line** &optional ARG)	Jump to line start in current (or all visible if 'avy-all-windows' is **t**) window. • Type highlighted key to move point.
	More control available with prefix argument: ARG=1: you can also type a number to cancel and use 'goto-line' for this typed number. ARG=4: negate the window scope determined by 'avy-all-windows'. ARG=any other number: use 'goto-line' to move point to this line number. • Move back to original location with `M-`` or `<f6><f6>` 📦 Requires the **avy external package** activated when **pel-use-avy** is set to t.		

Move Operation	Keystroke	Function	Note
• **To column**	The following command move point to a specified column. It does not provide Shift-marking.		
Go to a specific column	**M-g <tab>**	(**move-to-column** COLUMN &optional FORCE)	Prompts for a column number (or it can be entered as a command prefix). • Move point to column COLUMN in the current line.
	• The column of a character is calculated by adding together the widths as displayed of the previous characters in the line. • This function ignores line-continuation; there is no upper limit on the column number a character can have and horizontal scrolling has no effect. • If specified column is within a character, point goes after that character. • If it's past end of line, point goes to end of line. If a region is marked and point is at one end, modifies the region.		
Set Goal Column	The goal column identifies a target for point when moving to a line. The goal column is stored in the variable '***goal-column***'. This is a buffer-local setting.		
Set/reset Goal Column	**C-x C-n**	(**set-goal-column** ARG)	Set the current horizontal position as a goal for **C-n** and **C-p**. • Without argument: **activate** the goal column and set it to the current column. • With non nil argument (example: **C-u**): **disable** the goal column.
	The **C-n** and **C-p**commands will move to this position in the line moved to rather than trying to keep the same horizontal position. • When the goal column is active, it is shown as **G** on the ruler. Execute **ruler-mode** (**<f11> b -**) to activate the ruler & see if the goal column is active. ⚠ This command might be disabled at first, so in that case the first time you use it Emacs might prompt for activating it. See **enable-command.**		
• **within line**	The following commands move point within the current line. ☝ Typing **C-a** followed by **C-e** ensures point is at the line end (not before trailing whitespace). Similarly typing **C-e** followed by **C-a** ensures point is at column 0. Once point is at the extremity of a line, typing the last key again moves point to the first or last non-whiteface of the line.		
Beginning of line ★ PEL Enhanced Key ★	**C-a**	(**pel-beginning-of-line** &optional N)	Move point to beginning of current line. • Without argument N, uses 0: the current line. If point is already at the beginning of the line, move to the first non-whitespace character (using **back-to-indentation**). • With N>1 move to beginning of N lines forward, with N<1 move to beginning of abs(N) lines backward.
		(**org-beginning-of-line** &optional N)	**In Org-mode**: Go to the beginning of the current visible line. • If this is a headline, and 'org-special-ctrl-a/e' is set, ignore tags on the first attempt, and only move to after the tags when the cursor is already beyond the end of the headline.
End of line ★ PEL Enhanced Key ★	**C-e**	(**pel-end-of-line** &optional N)	Move point to end of current line. • Without argument N, uses 0: the current line. If point is already at the end of the line, move to the last non-whitespace character of the sin e(before trailing whitespace) if any. • With N>1 move to the end of N lines forward, with N<1 move to the end of abs(N) lines backward.
		(**org-end-of-line** &optional N)	**In Org-mode**: Go to the end of the line, but before ellipsis, if any. • If this is a headline, and 'org-special-ctrl-a/e' is set, ignore tags on the first attempt, and only move to after the tags when the cursor is already beyond the end of the headline.
First non-whitespace	**M-m**	(**back-to-indentation**)	Move point to the first non-whitespace character on this line. ➨ Shift marking works.
• **by word**	A "word" is a syntactic unit which is identified by a set of variables that can be modified and is controlled by Emacs **syntax table.** • See: **Help/Info** : **describe-syntax** (**C-h s**), **pel-syntax-at-point** (**<f11> ? e .**) show the syntax. • See **Text Modes**: the subword-mode and superword-mode can change their meaning.		
word forward	• **M-f** • **M-<right>** • **M-Z**	(**forward-word** &optional ARG)	Move point forward ARG words (backward if ARG is negative). • Supports superword-mode and subword-mode. • Shift marking: always work with **M-f** • works with **M-<right>** in graphics mode and in terminal mode when **pel-map-meta-left-right-to-Y-Z** user-option is nil. If it is **t** then in does not work in terminal mode. ⚠ ☝ Moves point right *after the end* of the word. To move to the first letter of next word use **M-n**.
Beginning of next word	**M-n**	(**pel-forward-word-start**)	Move point forward to beginning of next word. • Supports superword-mode but not the subword-mode. ⌨ On **Qwerty**, **Qwerztz** and **Azerty** keyboards the 'b' and 'n' letters are side by side. ⚠ This key is also remapped in other buffers and in several minor modes. For example: • In Info buffers, **M-n** is mapped to clone-buffer. • Inside shell buffers **M-n** is mapped to comint-next-input.
word backward	• **M-b** • **M-<left>** • **M-Y**	(**backward-word** &optional N)	Move backward ARG times until encountering the beginning of a word. • Supports superword-mode and subword-mode. • Shift marking always work with **M-b** • works with **M-<left>** in graphics mode and in terminal mode when **pel-map-meta-left-right-to-Y-Z** user-option is nil. If it is **t** then in does not work in terminal mode. ⚠
beginning of next token	**C-<right>**	(**pel-forward-token-start** &optional N)	Move to the beginning of next word/symbol. • It handles characters that may be part of symbol in the current major mode (like '_' in C), and jumps over them but stops at whitespace and operators. • Supports numerical argument for repetition. • Negative argument reverses the movement direction.
	☝ Useful when the superword-mode is not activated: allows jumping to next symbol while the word commands stop at each word separator character.		
beginning of previous token	**C-<left>**	(**pel-backward-token-start** &optional N)	Move to the beginning of previous word/symbol. • It handles characters that may be part of symbol in the current major mode (like '_' in C), and jumps over them but stops at whitespace and operators. • Supports numerical argument for repetition. • Negative argument reverses the movement direction.
	☝ Useful when the superword-mode is not activated: allows jumping to previous symbol while the word commands stop at each word separator character.		
Goto word using 1 letter with avy • **potentially in other window**	**M-g w**	(**avy-goto-word-1** CHAR &optional ARG BEG END SYMBOL)	Jump to the currently visible CHAR at a word start. Type first letter of target word, then highlighted key(s). • The window scope is determined by 'avy-all-windows'. • When ARG is non-nil, do the opposite of 'avy-all-windows'. 📦 Requires the **avy external package** activated when **pel-use-avy** user option is set to t.
Goto word with avy • **potentially in other window**	**M-g e**	(**avy-goto-word-0** ARG &optional BEG END)	• Jump to a word start. Highlights each word with letters to select to jump. • The window scope is determined by 'avy-all-windows'. • When ARG is non-nil, do the opposite of 'avy-all-windows' 📦 Requires the **avy external package** activated when **pel-use-avy** user option is set to t.

Move Operation	Keystroke	Function	Note
• **Specialized Search/ Move** See also: **Search/ Replace**	PEL provides a set of convenience/specialized search/navigation commands that move to pre-defined searched strings.		
Move point to next/ previous two consecutive spaces	• **<f11> s SPC** • **M-g M-SPC**	(**pel-search-two-spaces** BACKWARDS)	Move point forward to next location of 2 consecutive space characters. • With any argument: move backward to previous location of 2 consecutive spaces.
Move point to next/ previous empty line	• **<f11> s RET** • **M-g M-RET**	(**pel-search-empty-line** BACKWARDS)	Move point forward to the next empty line. • With any argument: move backward to previous empty line.
• **to next/previous space character**	The next commands move the point to the next whitespace character going forward or the previous whitespace character going backward.		
To next space char.	**<f11> M-SPC**	(**pel-to-forward-space**)	Move point to the next space character: ie after all non-whitespace characters. Repeatable.
To prev space char.	**<f11> C-SPC**	(**pel-to-backward-space**)	Move point to the previous space character: ie before all non-whitespace characters. Repeatable.
• **by syntactic elements**	Moving by syntactic elements, regardless of the word mode. These are marginally useful for normal operations. They are useful for: • investigating the syntax handling of various Emacs major modes • creating **keyboard macros** to fine tune the positioning of point. • See: **Help/Info** : **describe-syntax** (**C-h s**), **pel-syntax-at-point** (**<f11> ? e .**) show the syntax.		
Move point forward to next syntactic change	• **<f11> M-<right>** • **<f11> M-f**	(**pel-forward-syntaxchange-start**)	Move point forward: stop at beginning of character syntax change.
Move point backward to previous syntactic change	• **<f11> M-<left>** • **<f11> M-b**	(**pel-backward-syntaxchange-start**)	Move point backward: stop at beginning of character syntax change.
• **by blocks**	Blocks can be: pairs of brackets: (),[],{},<>,"", ''. Blocks using parentheses correspond to Lisp S-Expressions (sexp). This works in Lisp-like programming languages and programming languages that support block syntax.		
block backward	• **C-M-<left>** • **Esc C-<left>** ⚠ • **C-M-b** • **C-[C-b** • **Esc C-b**	(**backward-sexp** &optional ARG)	Move backward across one balanced expression (sexp). • With ARG, do it that many times. Negative arg -N means move forward across N balanced expressions. This command assumes point is not in a string or comment. • ⚠ With PEL: if you want to use **Esc C-<left>** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. ❖ **C-M-<left>** does not work on Windows, but **H-<left>** works. Several Linux distros map **C-M-<left>** to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems->settings->keyboard->shortcuts to prevent it from using that key sequence.
block forward	• **C-M-<right>** • **Esc C-<right>** ⚠ • **C-M-f** • **C-[C-f** • **Esc C-f**	(**forward-sexp** &optional ARG)	Move forward across one balanced expression (sexp). • With ARG, do it that many times. Negative arg -N means move backward across N balanced expressions. This command assumes point is not in a string or comment. • ⚠ With PEL: if you want to use **Esc C-<right>** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. ❖ **C-M-<right>** does not work on Windows, but **H-<right>** does. Several Linux distros map **C-M-<right>** to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems->settings->keyboard->shortcuts to prevent it from using that key sequence.
Up/inside sexp hierarchy	• **C-M-<up>** • **Esc C-<up>** • **C-M-u** • **C-[C-u** • **Esc C-u**	(**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. • ⚠ With PEL: if you want to use **Esc C-<up>** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. ❖ **C-M-<up>** does not work on Windows, but **H-<up>** does.
Down/inside sexp/block	• **C-M-<down>** • **Esc C-<down>** • **C-M-d** • **C-[C-d** • **Esc C-d**	(**down-list** &optional ARG)	Move forward down 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 backward but still go down a level. • This command assumes point is not in a string or comment. • ⚠ With PEL: To use **Esc C-<down>** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. ❖ **C-M-<down>** does not work on Windows, but **H-<down>** does.
Up/right sexp/block	**C-M-]**	(**up-list** &optional ARG ESCAPE-STRINGS NO-SYNTAX-CROSSING)	Move forward out of one level of parentheses. • This 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 backward 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.
Backward block/list	• **C-M-p** • **C-[C-p** • **Esc C-p**	(**backward-list** &optional ARG)	Move backward across one balanced group of parentheses. • This also work on other parentheses-like expressions defined by the current language mode. • With ARG, do it that many times. • Negative arg -N means move forward across N groups of parentheses. • This command assumes point is not in a string or comment.
Forward block/list	• **C-M-n** • **C-[C-n** • **Esc C-n**	(**forward-list** &optional ARG)	Move forward across one balanced group of parentheses. • This command will also work on other parentheses-like expressions defined by the current language mode. • With ARG, do it that many times. • Negative arg -N means move backward across N groups of parentheses. • This command assumes point is not in a string or comment.

Move Operation	Keystroke	Function	Note

• **to symbol definition**
  • **in *current* buffer**
  • **In all opened buffers**
 	
See also:
• Σ **Completion/Input**
• Σ **Menus**
• Σ **Speedbar**
	The following command can be used to move point to any quickly selected a symbol definition, in any major mode supported by **Emacs imenu**.
	• Most major modes for programming and markup languages support imenu. PEL adds extra support for some modes.
	• PEL provides 2 commands:
	  • **pel-goto-symbol** lists target symbols in the current buffer, allowing you to select one and jump to it.
	  • **pel-goto-symbol-any-buffer** does the same but for all buffers currently opened.
	• For each of these commands PEL provides a selectable user interface. The user interface used for each command when Emacs starts is selected by a customization user-option variable. During an editing session PEL provides a UI selection command. In both cases the available user interfaces depend on what you activate.
	  • Customize **pel-goto-symbol** user interface with **`M-g <f4> <f2>`** to access the customization buffer:
	  • 🎛 the **pel-initial-goto-symbol-UI** user option. Select one of:
	  • 0 = Use Emacs default: imenu
	  • 1 = Use Ido. 📦 Requires **idomenu** 🔄 **pel-use-ido** and **pel-use-idomenu** must both be turned on.
	  • 2 = Use Ivy. 📦 Requires **Ivy mode** and **Ivy mode completion with Counsel mode** 🔄**pel-use-ivy** and **pel-use-counsel** must both be on.
	  • 3 = Use helm. 📦 Requires **Helm mode** 🔄 **pel-use-helm** must be turned on.
	  • 4 = Use popup-imenu. 📦 Requires **popup-imenu** 🔄 **pel-use-popup-imenu** to be turned on (in **pel-pkg-for-imenu** group).
	  • 5 = Use popup-switcher. 📦 Requires **popup-switcher** 🔄 **pel-use-popup-switcher** to be turned on (in **pel-pkg-for-imenu** group).
	  • Modify the **pel-goto-symbol** UI for the current editing session with the **pel-select-goto-symbol-UI** command, bound to **`M-g <f4> h`**.
	  • Customize **pel-goto-symbol-any-buffer** user interface with with **`M-g <f4> <f2>`** to access the customization buffer:
	  • 🎛 📦 Requires **imenu-anywhere** 🔄 **pel-use-imenu-anywhere** user option must be set to one of the following values:
	  • Use emacs-default: basic Emacs completion. Use tab to see possible matches.
	  • Use Ido. 🔄 **pel-use-ido** must be turned on.
	  • Use Ivy. 📦 Requires **Ivy mode** 🔄**pel-use-ivy** must be on.
	  • Use helm. 📦 Requires **Helm mode** 🔄**pel-use-helm** must be turned on.
	  • Modify **pel-goto-symbol-any-buffer** UI for the current editing session with the **pel-select-goto-symbol-any-buffer-UI** command, bound to **`M-g <f4> y`**.
	• Use **pel-show-goto-symbol-settings ,** bound to **`M-g ?`** to show the current settings for both commands.
	When using Ido, for you have more options: you can select a different Ido prompt geometry and whether it uses 'flx' fuzzy matching.
	• Ido prompt geometries:
	  • The Emacs default: Ido linear selection,
	  • Grid initially collapsed or expanded. 📦 Requires **ido-grid-mode** 🔄 Activate it with **pel-use-ido-grid-mode** user-option turned on.
	  • Vertical list. 📦 Requires **ido-vertical-mode** 🔄 Activate it with **pel-use-ido-vertical-mode** user-option turned on.
	  • Select the initial geometry with the **pel-initial-ido-geometry**. Change it in the editing session with **pel-select-ido-geometry** (**`M-g <f4> M-g`**).
	• Ido 'flx' fuzzy matching 📦 requires **flx-ido**. 🔄 Activate it with **pel-use-flx** user-option turned on.
	• Also use **`<f11> <f10> <f2>`** to customize the PEL iMenu user-options which have an impact on the way the iMenu entries are displayed.
	☝ Note that it is also possible to use the **Speedbar** (which also uses the symbols detected by **imenu**). See Σ **Speedbar** .		

Find definitions using IMenu See also: • Σ **Completion/Input** • Σ **Menus**	• **`<f11> <f10> i`** • **`M-g i`** • **`M-g M-i`**	(**imenu** INDEX-ITEM)	Lists imenu-detected items from the current buffer (according to its major mode). • For example, in a elisp file, the entry points are the function definitions and may include the variables and other items depending what function does the parsing (it can be semantic which provides more information). Provides one of the following interfaces to let user select entry to jump to: • The default: input completion, using the minibuffer window and tab completion. • a pop-up window : available in Graphics mode selected by mouse or in both graphics and terminal (TTY) modes when the **imenu-use-popup-menu** user-option is turned on. • with PEL you can use **pel-imenu-toggle-popup** (bound to **`M-g <f4> p`**) to toggle the user interface used by **imenu**.
Move to imenu detected symbol definition in current buffer ★★	• **`M-g h`** • **`M-g M-h`**	(**pel-goto-symbol**)	Prompt using for imenu symbol of the current buffer and move point to it. • Refresh imenu and jump to a place in the buffer using the completion method selected. • Modify user interface currently used with **`M-g <f4> h`**. • The command sets a ref-marker before moving. Return to previous location by typing **`M-,`**
	⚠🚧 There is a bug in popup-switcher that prevents listing items in some files. 🚧		
Move to imenu detected symbol definition of all opened buffers ★★	• **`M-g y`** • **`M-g M-y`**	(**pel-goto-symbol-any-buffer**)	Prompt using for imenu symbol of all loaded menu supported buffers and move point to the selection. • Provide input completion using the currently selected method (emacs-default, ido, ivy or helm). • Select the default completion method by customization setting **pel-use-imenu-anywhere**. • Modify user interface currently used with **`M-g <f4> y`**. • The command sets a ref-marker before moving. Return to previous location by typing **`M-,`**

Display current setting of commands:
• **pel-goto-symbol**
• **pel-goto-symbol-any-buffer**
	`M-g ?`	(**pel-show-goto-symbol-settings**)	Display current settings used by the goto symbol commands in the echo area. For example:
			`goto-symbol UI is: popup-switcher`
			`goto-any-buffer UI is: Ido`
			`- iMenu lists are not flatten.`
			`- Ido uses:`
			`  - Ido prompt geometry: grid mode, starts collapsed: expand with tab`
			`  - Ido Ubiquitous mode: off`
			`  - flx-ido mode: off`

Select Input Completion used by pel-goto-symbol	**`M-g <f4> h`**	(**pel-select-goto-symbol-UI**)	Select the input completion method used by the **pel-goto-symbol** command for the duration of the current editing session.
			• When Emacs starts the method used is determined by the value of the **pel-initial-goto-symbol-UI** user-option. You can use this command to change what is used in the current editing session without affecting the customized default.
			• See also the commands to control input completion (see Σ **Completion/Input**)
			  • **pel-select-ido-geometry**: **`M-g <f4> M-g`**
			  • **pel-ido-ubiquitous** : **`M-g <f4> M-u`**
			  • **pel-flx-ido** : **`M-g <f4> M-f`**

Select Input Completion Method used by pel-imenu-anywhere	**`M-g <f4> y`**	(**pel-select-goto-symbol-any-buffer-UI**)	Select the input completion method used by the **pel-imenu-anywhere** command for the duration of the current editing session and used by the **pel-goto-symbol-any-buffer** command. • When Emacs starts the method used is determined by the value of the PEL **pel-use-imenu-anywhere** user-option. You can use this command to change what is used in the current editing session without affecting the customized default.

Toggle imenu between a hierarchical and a flat list.	• **`<f11> <f10> f`**
	• **`M-g <f4> f`**
		(**pel-imenu-toggle-flatten**)	Toggles between imenu using a hierarchical menu (the default) and a flat menu.
			• 🎛 The maximum number of entries in a imenu list is controlled by 2 imenu user-options:
			  • imenu-max-items: size limit of a pop-up imenu.
			  • imenu-max-item-length: size limit of a drop down imenu
			📦 Requires **flimenu** external package 🔄 activated by **pel-use-flimenu** user-option.
			• Note that when the number of items to display exceeds the maximum length of the imenu, there imenu will be split anyway in multiple sections and will end up being "hierarchical" but instead of being split by type of content, it will be split on type and by alphabetical names.

Toggle order of appliance in the imenu	• **`<f11> <f10> o`** • **`M-g <f4> o`**	(**pel-imenu-toggle-follows-order**)	Changes the order of entries in the imenu between the default and the order of appearance of the symbols in the buffer. 🎛 Set the default with the **pel-imenu-index-follows-order-p** user-option.
Toggle imenu I/F between completion buffer and pop-up menu	• **`<f11> <f10> p`** • **`M-g <f4> p`**	(**pel-imenu-toggle-popup** &optional IN-CURRENT-BUFFER)	Toggle the use of pop-up menu versus completion buffer for imenu. • By default this applies to imenu issued in all buffers, but with the IN-CURRENT-BUFFER argument set the change applies only to the current buffer.
Toggle automatic imenu rescan	• **`<f11> <f10> R`** • **`M-g <f4> R`**	(**pel-imenu-toggle-auto-rescan**)	Toggle imenu automatic rescan • Default is set by **imenu-auto-rescan** user-option.

Move Operation	Keystroke	Function	Note
• **by defun**	The commands move point by function definitions. In elisp code that's defun, defvar, etc.., but it also works in other modes, as the same keys are bounded to different commands. ☝ The **<f6>** cursor key mappings use **<up>** and **<down>** to move to the beginning of the defun, and **<left>** and **<right>** to the end of the defun. ☝ In this context the word *defun* corresponds to the concept of function, method, procedure, section, used for the current buffer. ⚠ These commands work well when editing Lisp-like programming languages. The first two commands will skip nested functions at a level nested relative to the current level (and that can be considered a nice feature) The extra commands provided by PEL are based on the first 2 commands and inherit these limitations: • The **pel-beginning-of-next-defun** works well in most cases but has problems handling some C++ template code. • The **pel-end-of-previous-defun** is even more affected by the limitations when used to move inside some nested code. Obviously need a better sequential navigation mechanism for nested functions definitions in source code.🚧		
Backward to beginning of defun	• **<f6> <up>** • **C-M-a** • **C-M-<home>** • **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. ⚠ 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 defun	• **<f6> <right>** • **C-M-e** • **C-M-<end>** • **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. ⚠ 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 defun	**<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>**. ☝ 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 Python and others.
Backward to end of previous define	**<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>**.
• **by URL**	With the following commands you can navigate quickly across the buffer to the next and previous URL. • You can also activate the **goto-address-mode** to act on the URL. PEL adds command to download a web page to a local temporary file and visit it. • See File-mngt for more on that.		
Move to end of next URL in buffer See also: **File-mngt**	**C-c C-n** **<f6> C-n**	(**pel-goto-next-url**)	Move point forward to the end of the next URL located in the current buffer. • The **C-c C-n** binding is only available when point is over the URL and the **goto-address-mode** minor mode is active. Use **<f11> f u** or **<f11> f U** to activate this mode. • The global **<f6> C-n** key binding activates the goto-address-mode if it is not already active.
Move to beginning of previous URL in buffer See also: **File-mngt**	**C-c C-p** **<f11> C-p**	(**pel-goto-previous-url**)	Move point backward to the beginning of the previous URL located in the current buffer. • The **C-c C-p** binding is only available when point is over the URL and the **goto-address-mode** minor mode is active. Use **<f11> f u** or **<f11> f U** to activate this mode. • The global **<f6> C-p** key binding activates the goto-address-mode if it is not already active.
• **by sentences**	The variable 'sentence-end' is a regular expression that matches ends of sentences. Also, every paragraph boundary terminates sentences as well. The definition of what is a sentence depends on the major mode. For example, in C++//l mode the end of sentence means end of C++ statement. More information on navigation is available on each mode.		
To beginning of sentence	**M-a**	(**backward-sentence** &optional ARG)	Move backward to start of sentence. With arg, do it arg times.
To end of sentence	**M-e**	(**forward-sentence** &optional ARG)	Move forward to next end of sentence. With argument, repeat. With negative argument, move backward repeatedly to start of sentence.
• **by paragraphs**	A paragraph start is the beginning of a line which is a 'paragraph-start' or which is ordinary text and follows a 'paragraph-separate'ing line; except: if the first real line of a paragraph is preceded by a blank line, the paragraph starts at that blank line. Note: It is not possible to Shift mark with these bindings. Use C-SPC to mark first then use the keys to move and extend the region.		
Backward paragraph	• **C-<up>** • **M-{**	(**backward-paragraph** &optional ARG)	Move backward to start of paragraph. • With argument ARG, do it ARG times; • a negative argument ARG = -N means move forward N paragraphs.
Forward paragraph	• **C-<down>** • **M-}**	(**forward-paragraph** &optional ARG)	Move forward to end of paragraph. • With argument ARG, do it ARG times; • a negative argument ARG = -N means move backward N paragraphs.
• **by pages**	A page boundary is any line whose beginning matches the regexp 'page-delimiter'. By default, that is a ^L (form feed) at the beginning of a line.		
Forward 1 page	**C-x]**	(**forward-page** &optional COUNT)	Move forward to page boundary. With arg, repeat, or go back if negative.
Backward 1 page	**C-x [**	(**backward-page** &optional COUNT)	Move backward to page boundary. With arg, repeat, or go forward if negative.

Move Operation	Keystroke	Function	Note
• **to buffer & *window top/end***	The following commands move point to top, bottom and centre of the current buffer or window. • PEL provides the **<home>** and **<end>** keys. They behave like the Brief/CRiSP equivalent keys with the additional handling of Emacs fields. • Emacs provide the **M-<** and **M->** to move to top and end of buffer, and **M-r** to move top top, centre and end of the visible portion of the current buffer.		
To beginning of: line, window, buffer **★ PEL Enhanced Key ★** See also: **Σ Scrolling**	**<home>**	(**pel-home**)	The behaviour of this command depends on the current point location: • ➜ beginning of field (if any) ➜ beginning of line ➜ beginning of window ➜ beginning of buffer So to go to beginning of buffer, type <home> 3 times if point is not at the beginning of line or window, 4 times if the line has a field (like prompt in interactive buffers like IELM) and point is not at the beginning of field. • Push mark at previous position, unless either a **C-u** prefix is supplied, or Transient Mark mode is enabled and the mark is active. • Scrolls other window when PEL window scroll mode is active. See **Σ Scrolling**. On macOS laptops, the **<home>** key is not available; use **Fn <left>** instead. ⚠ Avoid in Keyboard Macros: The behaviour of the key depends on the original position, so it's not a good fit for keyboard macros. Inside keyboard macros, use **C-a** instead inside keyboard macros when you want to move point to the beginning of a line.
To end of line, window, buffer **★ PEL Enhanced Key ★** **Under GNU Screen →** **Σ Keyboard Macros →** **Σ Outline →**	**<end>**	(**pel-end**)	The behaviour of this command depends on the current point location: • ➜ end of field (if any) ➜ end of line ➜ end of window ➜ end of buffer So to go to end of buffer, type <end> 3 times if point is not at the end last window line, or 4 times if there is a field in the line after the point's position. REPL like IELM use fields on prompt lines. • If the buffer is narrowed, this command uses the end of the accessible part of the buffer. • Push mark at previous position, unless either a **C-u** prefix is supplied, or Transient Mark mode is enabled and the mark is active. • Scrolls other window when PEL window scroll mode is active. See **Σ Scrolling**. On macOS laptops, the **<end>** key is not available; use **Fn <right>** instead. ☝ In some system, when running Emacs under **GNU Screen**, the **<end>** key registers as **<select>**. Set **pel-select-key-is-end** to circumvent this. ⚠ Avoid in Keyboard Macros: The the behaviour of the key depends on the original position, so it's not a good fit to use in a keyboard macro. Use **C-e** instead inside keyboard macros when you want to move point to the end of a line. ⚠ In collapsed outlines you may have to type the **<right>** cursor, or **C-a** then **C-e** to move to the real end of the last line of a window to be able to move to the last line of the buffer.
To beginning of buffer ‡	**M-<**	(**beginning-of-buffer** &optional ARG)	Move point to the beginning of the buffer. • With numeric arg N, put point N/10 of the way from the beginning. • If the buffer is narrowed, this command uses the beginning of the accessible part of the buffer. • Push mark at previous position, unless either a C-u prefix is supplied, or Transient Mark mode is enabled and the mark is active.
To end of buffer ‡	**M->**	(**end-of-buffer** &optional ARG)	Move point to the end of the buffer. • With numeric arg N, put point N/10 of the way from the end. • If the buffer is narrowed, this command uses the end of the accessible part of the buffer.

To left line center, top, bottom of window. ‡	**M-r**	(**move-to-window-line-top-bottom** &optional ARG)	Position point relative to window.
			• By default moves to beginning of line at: center, top, bottom of window in successive calls.
			  • The **recenter-positions** user-option can be modified to change that default.
			• Arguments:
			  • A negative argument reverses the order.
			  • A numeric argument identifies a line number.
			  • Number 0 identifies the first line in window: **M-0 M-r** : move to top of window
			  • Negative 0 identifies the last line in window: **M-- M-0 M-r** : move to end of window

• ***in buffer of* other *windows***	The following 2 commands do **not** move point in the current buffer, they move it in the buffer showing in the **other** window.		
To beginning of buffer in other window	• **Esc <home>** • **M-<home>**	(**beginning-of-buffer-other-window** ARG)	Move point position to the beginning of the buffer in the other window. Stay in current window. • Leave mark at previous position. • With arg N, put point N/10 of the way from the true beginning.
To end of buffer in other window	• **Esc <end>** • **M-<end>**	(**end-of-buffer-other-window** ARG)	Move point position to the end of the buffer in the other window. Stay in current window. • Leave mark at previous position. • With arg N, put point N/10 of the way from the true end.
• **Goto match/ Compilation Error**	A match is the result of a previous operation like: grep search result, compilation errors, etc… Use these commands inside a compilation-mode buffer that shows the list of warning/errors resulting in code/syntax checking.		
Jump to next match	• **C-x `** • **M-g n** • **M-g M-n**	(**next-error** &optional ARG RESET)	A prefix ARG specifies how many error messages to move; negative means move back to previous error messages. Just **C-u** as a prefix means reparse the error message buffer and start at the first error.
Jump to previous match	• **M-g p** • **M-g M-p**	(**previous-error** &optional N)	Prefix arg N says how many error messages to move backwards (or forwards, if negative).
recentering in current window	The following 2 command do **not** move point, but reposition the text in the current window. • These are quite useful as they can be used to refresh the view in the current window. See also: **Σ Windows**		

Position current line to window's Center / Bottom / Top .
Refresh screen.
	• **C-l**
	• **<f11> C-l**
		(**recenter-top-bottom** &optional ARG)	Without argument: moves the current line to window: center -> top -> bottom.
			• With arg: centre first:
			  • **C-u C-l C-l C-l C-l**
			  • ➜ center ➜ bottom ➜ center ➜ top
			• With negative arg: bottom first:
			  • **C-- C-l C-l C-l**
			  • ➜ bottom ➜ center ➜ top
			• With arg 0: top first:
			  • **M-0 C-l C-l C-l**
			  • ➜ top ➜ bottom ➜ center
			• With numeric positive: move current line to window top position N
			• With negative numeric: move current line to bottom window position: -1 := last line
			• PEL provides the **<f11> C-l** key binding because some modes use **C-l** as a prefix key.

Reposition comment/ definition in full view	• **C-M-l** • **C-[C-l** • **Esc C-l**	(**reposition-window** &optional ARG)	Attempts to make the current comment or current definition fully visible by scrolling the lines without changing the point. • Further invocations move it to the top of the window or toggle the visibility of comments that precede it (by scrolling the lines).