Navigation

Move Operation	Keystroke	Function	Note
Navigation Commands	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**.		
Navigation See also: **Customize**	`<f11> <f2> n`	(**pel-cfg-pkg-navigation** &optional OTHER-WINDOW)	Customize PEL navigation tools support. • If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window and open navigation tool related group: avy.
Shift-Selection	If you press and hold the **shift** key while typing a movement command, it 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 also called "*Shift-Marking*" in this document. It is available for several commands but not all. Also when running Emacs in Terminal mode, less commands are able to use it. The ability to perform "Shift-Marking" is identified in the description of the commands below.		
by character	Note: All commands in the following group support the bidirectional context, so when editing right to left text these commands may move the point in the reverse direction.		
right/next char	**`C-f`**	(**forward-char** &optional N)	Move point N characters forward (backward if N is negative). • On reaching end or beginning of buffer, stop and signal error. • Interactively, N is the numeric prefix argument. • If N is omitted or nil, move point 1 character forward. • Depending on the bidirectional context, the movement may be to the right or to the left on the screen. This is in contrast with <right>. ☛ Shift marking is available in graphics mode, not in terminal mode.
right/next char	**`<right>`**	(**right-char** &optional N)	Move point N characters to the right (to the left if N is negative). On reaching beginning or end of buffer, stop and signal error. ☛ Shift marking works with this command.
left/previous char	**`C-b`**	(**backward-char** &optional N)	Move point N characters backward (forward if N is negative). • On attempt to pass beginning or end of buffer, stop and signal error. • Interactively, N is the numeric prefix argument. • If N is omitted or nil, move point 1 character backward. • Depending on the bidirectional context, the movement may be to the right or to the left on the screen. This is in contrast with <left>. ☛ Shift marking is available in graphics mode, not in terminal mode.
left/previous char	**`<left>`**	(**left-char** &optional N)	Move point N characters to the left (to the right if N is negative). On reaching beginning or end of buffer, stop and signal error. ☛ Shift marking works with this command.
Go to a specific char position	**`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. ☛ Shift marking does not work with this command.
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. Shift selection is not supported but you can mark (see **Marking**) before moving to create a marked region. 📦 These commands require the **avy external package** activated when the **pel-use-avy** user option is set to t.		
Jump to visible char	• **`C-:`** • **`M-G`**	(**avy-goto-char** CHAR &optional ARG)	Jump to the currently visible CHAR. • The window scope is determined by 'avy-all-windows' (ARG negates it). ☛ Shift marking does not work with this command.
Jump to visible 2 chars	• **`C-'`** • **`M-H`**	(**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. ☛ Shift marking does not work with this command.
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	• **`M-g g`** • **`M-g M-g`** • `⌘-l`	(**goto-line** LINE &optional BUFFER)	Go to LINE, counting from line 1 at beginning of buffer. • If called interactively, a numeric prefix argument specifies LINE; 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. ☛ Shift marking does not work with this command.
Goto line using avy	**`M-g f`**	(**avy-goto-line** &optional ARG)	Jump to a line start in current buffer. Type the highlighted key to move point. • When ARG is 1, jump to lines currently visible, with the option to cancel to 'goto-line' by entering a number. • When ARG is 4, negate the window scope determined by 'avy-all-windows'. • Otherwise, forward to 'goto-line' with ARG. ☛ Shift marking does not work with this command. 📦 Requires the **avy external package** activated when the **pel-use-avy** user option is set to t.
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.
Set/reset 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.		

Move Operation	Keystroke	Function	Note
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`. Those commands will move to this position in the line moved to rather than trying to keep the same horizontal position. • Without argument: **activate** the goal column and set it to the current column. • With non nil argument (example: `C-u`): **disable** the goal column. ☛ When the goal column is active, it is shown as **G** on the ruler (when the ruler-mode is active.) Execute ruler-mode (`<f11> b -`) to activate the ruler to 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 this command. See **enable-command** in the Emacs Lisp table.
into line	The following commands move point within the current line.		
Beginning of line ★ PEL Enhanced Key ★	`C-a`	1. Text: (**pel-beginning-of-line** ARG) 2. Org-Mode: (**org-beginning-of-line** &optional N)	Move point to beginning of current line as displayed. If point is already at the beginning of the line, move to the fist non-whitespace character (using **back-to-indentation**). • (If there's an image in the line, this disregards newlines which are part of the text that the image rests on.) • With argument ARG not nil or 1, move forward ARG - 1 lines first. • If point reaches the beginning or end of buffer, it stops there. (But if the buffer doesn't end in a newline, it stops at the beginning of the last line.) ☛Shift marking is available in graphics mode, not in terminal mode. **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`	1. Text: (**pel-end-of-line** ARG) 2. Org-Mode: (**org-end-of-line** &optional N)	Move point to end of current line as displayed. If point is already at the end of the line, move point to the first trailing space character if there is any. • With argument ARG not nil or 1, move forward ARG - 1 lines first. • If point reaches the beginning or end of buffer, it stops there. ☛Shift marking is available in graphics mode, not in terminal mode. **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 with this command.
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 the subword-mode and superword-mode to change that.		
word forward	• `M-f` • `M-<right>`	(**forward-word** &optional ARG)	Move point forward ARG words (backward if ARG is negative). If ARG is omitted or nil, move point forward one word. • Supports superword-mode and subword-mode. ☛Shift marking works with this command (both keys). ☝ This moves point right *after the end* of the word. If you want 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. ☛Shift marking works with this command. ☝ On both Qwerty and Azerty keyboards the 'b' and 'n' letters are side by side. ⚠ This key binding differs in other buffers. • In Info buffers, `M-n` is mapped to clone-buffer. • Inside shell buffers `M-n` is mapped to comint-next-input. • For the moment PEL does not change this but might (via a PEL user option in the future) since it's such a useful key.
word backward	• `M-b` • `M-<left>`	(**backward-word** &optional N)	Move backward until encountering the beginning of a word. With argument ARG, do this that many times. • Supports superword-mode and subword-mode. ☛ Shift marking works with this command (both keys).
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. ☛Shift marking works with this command. ☝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. ☛Shift marking works with this command. ☝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	`M-g w`	(**avy-goto-word-1** CHAR &optional ARG BEG END SYMBOL)	Jump to the currently visible CHAR at a word start. • The window scope is determined by 'avy-all-windows'. • When ARG is non-nil, do the opposite of 'avy-all-windows'. • ☛ Shift marking does not work with this command. 📦 Requires the **avy external package** activated when the **pel-use-avy** user option is set to t.
Goto word with avy	`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. No need to type a key first. • The window scope is determined by 'avy-all-windows'. • When ARG is non-nil, do the opposite of 'avy-all-windows' ☛ Shift marking does not work with this command. 📦 Requires the **avy external package** activated when the **pel-use-avy** user option is set to t.
by syntactic elements	Moving by syntactic elements, regardless of the word mode. These are marginally useful except for investigating the syntax handling of various Emacs major modes.		
Move point forward to next syntactic change	`<f11> M-<right>`	(**pel-forward-syntaxchange-start**)	Move point forward: stop at beginning of character syntax change.
Move point backward to previous syntactic change	`<f11> M-<left>`	(**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).		
block backward	• `C-M-b` • `C-M-<left>` • `C-[ C-b` • `Esc C-b` • `Esc C-<left>`	(**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. • `C-M-b` : ☛ Shift marking is available in graphics mode, not in terminal mode. • `C-M-<left>` : ☛ Shift marking works with this command. ❖ `C-M-<left>` does not work on Windows, but `H-<left>` works.

Move Operation	Keystroke	Function	Note
block forward	• **C-M-f** • **C-M-<right>** • **C-[C-f** • **Esc C-f** • **Esc C-<right>**	(**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. • **C-M-f** : ☛ Shift marking is available in graphics mode, not in terminal mode. • **C-M-<right>** : ☛ Shift marking works with this command. ❖ **C-M-<right>** does not work on Windows, but **H-<right>** does.
Up/inside sexp hierarchy	• **C-M-u** • **C-M-<up>** • **C-[C-u** • **Esc C-u** • **Esc C-<up>**	(**backward-up-list** &optional ARG ESCAPE-STRINGS NO-SYNTAX-CROSSING)	Move backward out of one level of parentheses. • This command will also work on other parentheses-like expressions defined by the current language mode. With ARG, do this that many times. A negative argument means move forward but still to a less deep spot. • **C-M-u** : ☛ Shift marking is available in graphics mode, not in terminal mode. • **C-M-<up>** : ☛ Shift marking works with this command. ❖ **C-M-<up>** does not work on Windows, but **H-<up>** does.
Down/inside sexp/block	• **C-M-d** • **C-M-<down>** • **C-[C-d** • **Esc C-d** • **Esc C-<down>**	(**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. • **C-M-d** : ☛ Shift marking is available in graphics mode, not in terminal mode. • **C-M-<down>** : ☛ Shift marking works with this command. ❖ **C-M-<down>** does not work on Windows, but **H-<down>** does.
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 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 forward across N groups of parentheses. • This command assumes point is not in a string or comment. • **C-M-p** : ☛ Shift marking is available in graphics mode, not in terminal mode.
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. • **C-M-n** : ☛ Shift marking is available in graphics mode, not in terminal mode.
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 a better mechanism is required to navigate easily through the functions definitions in source code files in presence of various nesting scenarios.🚧		
Backward to beginning of defun	• **C-M-a** • **C-M-<home>** • **<f6> p** • **<f6> <up>** • **C-[C-a** • **Esc C-a**	(**beginning-of-defun** &optional ARG)	Move backward to the beginning of a defun. • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. ☛Shift marking is available in graphics mode, not in terminal mode (for **C-M-a** and **C-M-<home>**). However **<f6> p** handles Shift-marking fine in terminal mode. ⚠ 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	• **C-M-e** • **C-M-<end>** • **<f6> <right>** • **C-[C-e** • **Esc C-e**	(**end-of-defun** &optional ARG)	Move forward to next end of defun. With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. ☛Shift marking is available in graphics mode, not in terminal mode (both keys). ⚠ This command moves to the end of the next **top-level** function or class. It skips the nested functions and methods.
Forward to start of next defun	• **<f6> n** • **<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-`**. ☛Shift marking is available. ☝ 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-`**. ☛Shift marking is available. ☝ This command complements this set of 4 commands. • ⚠ It handles most nested functions or class methods in languages like Python and others but not always. In some cases it does not move the point. Better logic is needed. 🚧
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. ☛ Shift marking works with this command.
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. ☛ Shift marking works with this command.
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. • **C-<up>** : ☛ Shift marking works with this key. • **M-{** : ☛ Shift marking does not work with this key.

Move Operation	Keystroke	Function	Note
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. • **C-<down>** : ☛ Shift marking works with this key. • **M-}** : ☛ Shift marking does not work with this key.
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. ☛ Shift marking does not work with this key.
Backward 1 page	**C-x [**	(**backward-page** &optional COUNT)	Move backward to page boundary. With arg, repeat, or go fwd if negative. ☛ Shift marking does not work with this key.
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.		
Position current line to window's Center / Bottom / Top . **Refresh screen.**	**C-l**	(**recenter-top-bottom** &optional ARG)	Without argument: moves the current line to window: center -> top -> bottom. • With arg: center -> bottom -> center -> top • With negative arg: bottom -> bottom -> center -> top • With numeric positive: move current line to window top position N, 0 := top window line • With negative numeric: move current line to bottom window position: -0 := bottom line
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).
in window & 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. ☛ Shift marking does not work with this key.
To beginning of: line, window, buffer **★PEL Enhanced Key ★** See also: **∑ 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 **∑ Scrolling**. ☛ Shift marking is available in graphics mode, not in terminal mode. On macOS laptops, the **<home>** key is not available; use **Fn <left>** instead. ⚠ Because the behaviour of the key depends on the original position avoid using this key inside keyboard macros when you cannot guarantee the position when the keyboard macro is invoked. Use **C-a** instead inside keyboard macros when you want to move point to the beginning of a line.
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. ☛ Shift marking does not work with this key.
To end of line, window, buffer **★ PEL Enhanced Key ★**	**<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 **∑ Scrolling**. ☛ Shift marking is available in graphics mode, not in terminal mode. On macOS laptops, the **<end>** key is not available; use **Fn <right>** instead. ⚠ Because the behaviour of the key depends on the original position avoid using this key inside keyboard macros when you cannot guarantee the position when the keyboard macro is invoked. Use **C-e** instead inside keyboard macros when you want to move point to the end of a line.

To left line center, top, bottom	**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.
			Arguments:
			• A negative argument reverses the order.
			• A numeric argument identifies a line number.
			  • Number 0 identifies the first line in window.
			  • A Negative 0 identifies the last line in window.
			☛ Shift marking does not work with this key.

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 to the beginning of the buffer in the other 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 to the end of the buffer in the other 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…		
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).