↖ Topic Index

Emacs support for AWK

Description	Keystroke	Function	Note

Editing AWK
○ Help & customize
• cc-mode learn/mod
○ cc-mode set/help
• electric keys
• electric-pair mode
○ insert new line(s)
• comments
• hide/show comment
○ delete whitespace
• indentation
○ indent rigidly
• unindent
• open file at point
○ insert ( )
• mark function
• show function name
• search support
• highlighting blocks
• navigate in AWK code
  • by xref
  • by statement
  ○ by block
○ Programming help
 	
Last updated on:
	PEL extends Emacs support for the AWK programming language as it does for C, when **pel-use-awk** user-option is set to **t**.
	• Type `C-h o pel-use-awk RET` to open the customization buffer to activate it.
	Emacs support for AWK is, like C, an extension of the **CC Mode**. Therefore most functionality provided by the cc-mode is available for AWK editing.
	
	nformation about AWK:
	• **AWK @ Wikipedia**
	• **The GNU AWK Manual**
	• **AWK in 20 Minutes**
	• **AWK Linux Manual page**
	
	2025-05-16		

Open this PDF file. See also: **Help/Info**	`<f11> SPC W <f1>` `<f12> <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **AWK - Awk** 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 Awk support	`<f11> SPC W <f2>` `<f12> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL Awk support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Customize Emacs Awk support	`<f11> SPC W <f3>` `<f12> <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Awk support: c, electric. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Comments			
Toggle display of comments in buffer or active region See also: • **Comments**	`<f11> ; ;`	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see **Comments**). PEL activates it when the **pel-use-hide-comnt** user option is **t**.
CC Mode Style Management • **Learn/Modify style used in current buffer**	Automatic indentation, brace format style and several other AWK stylistic elements are controlled by the CC Mode and the CC mode variables. • You can impose an indentation style by customization. • You can also adjust the style to what is used in the current buffer: Emacs provides the following commands to parse the source code and identify the style it uses. It *learns* the style and sets the style controlling variables from what it detects in the buffer. • ☝ Use this to **adapt** to source code written by others and want to continue using the same style, or to **modify** the style. • ☝ For the following commands all commands that use a key binding that ends with an upper case letter install the style.		
Show/Modify syntactic context ☝ Set style indentation	`C-c C-o`	(**c-set-offset** SYMBOL OFFSET &optional IGNORED)	Change the value of a syntactic element symbol in 'c-offsets-alist'. • SYMBOL is the syntactic element symbol to change and OFFSET is the new offset for that syntactic element. ☝ Use this to **modify** a specific style, like how something is indented.
Show syntactic information for current line	`C-c C-s`	(**c-show-syntactic-information** ARG)	Show syntactic information for each syntactic element present *on the current line*. • Display the syntactic information list: style and position highlight the reference position(s) listed as argument to the syntactic list. • Each list starts with a **syntactic symbol** with zero or several reference positions. • With universal argument, inserts the analysis as a comment on that line.
Guess the style used in the current buffer, do not install it	`<f12> <f4> g g`	(**c-guess-buffer-no-install** &optional ACCUMULATE)	Guess the style on the whole current buffer; don't install it. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess the style of the code in the buffer and install it.	`<f12> <f4> g B`	(**c-guess-buffer** &optional ACCUMULATE)	Guess the style on the whole current buffer, and install it. • The style is given a name based on the file's absolute file name. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess style in the region and install it.	`<f12> <f4> g G`	(**c-guess** &optional ACCUMULATE)	Guess the style using the first '**c-guess-region-max**' bytes of the file, and install it. • The **c-guess-region-max** user-option defaults to 50,000 bytes, nil means all buffer. • The style is given a name based on the file's absolute file name. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess the style of a region and install it.	`<f12> <f4> g R`	(**c-guess-region** START END &optional ACCUMULATE)	Guess the style on the region and install it. • The style is given a name based on the file's absolute file name. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Set buffer style to guessed style and install it.	`<f12> <f4> g I`	(**c-guess-install** &optional STYLE-NAME)	Install the latest guessed style into the current buffer. • This guessed style is a combination of 'c-guess-guessed-basic-offset', 'c-guess-guessed-offsets-alist' and 'c-offsets-alist'. • The style is entered into CC Mode's style system by 'c-add-style'. Its name is either STYLE-NAME, or a name based on the absolute file name of the file if STYLE-NAME is nil.
View Guessed style as a set of Emacs Lisp statements	`<f12> <f4> g ?`	(**c-guess-view** &optional WITH-NAME)	Emit emacs lisp code which defines the last guessed style, so you can put the code into .emacs if you prefer the guessed code. • "STYLE NAME HERE" is used as the name for the style in the emitted code. If WITH-NAME is given, it is used instead. WITH-NAME is expected as a string but if this function called interactively with prefix argument, the value for WITH-NAME is asked to the user.

Description	Keystroke	Function	Note
CC Mode support Behaviour control Use **<f12> <f4> ?** to display the current state.	Use following commands to dynamically change the behaviour of important keys such as the return key, delete key, semi-colon, etc.. The CC Mode controls the indentation and bracket style and what happens when electric characters are typed (when electric mode is activated). • **CC Mode state displayed in the mode line**: ℒC{…} where: • ℒ is the CC mode programming language name: AWK, C, C++, ObjC, etc… • C is the C comment style: '*' for block command (/* */) and '/' for line comments (//) • {…} are the other electric flags: '1' for electric mode , 'a' for auto-newline mode, 'h' for hungry mode, 'w' for subword mode		
Toggle Electric state †	• **C-c C-l** • **<f12> <f4> e**	(**c-toggle-electric-state** &optional ARG)	Toggle the electric indentation feature done with the electric character keys. • Optional numeric ARG, if supplied, turns on electric indentation when positive, turns it off when negative, and just toggles it when zero or left out.
Set indentation style †	• **C-c .** • **<f12> <f4> s**	(**c-set-style** STYLENAME &optional DONT-OVERRIDE)	Set the bracket/indentation style for the current buffer. • Prompts for the name. • Supports tab completion (so use tab to see the list). Can be one of the values supported by Emacs. You can add your customized mode with Emacs Lisp code.
Change indentation width for current buffer †	**<f12> <f4> TAB**	(**pel-cc-set-indent-width** &optional NEW-WIDTH)	Interactively change the Indentation with for current buffer to NEW-WIDTH. • Prompt for new value. Use 0 to restore value specified by configuration (**pel-c-indent-width**). ☝ This can be used to change indentation several times in a file.
Toggle syntactic indentation	**<f12> <f4> i**	(**c-toggle-syntactic-indentation** &optional ARG)	Toggle syntactic indentation. Toggle if no ARG or if ARG is 0. • With positive ARG turn on syntactic indentation, turns it off when negative.
	• When syntactic indentation turned on (the default), the indentation functions and electric keys indent according to syntactic context keys, when applicable. • When it's turned off, the electric keys don't reindent, the indentation functions indents every new line to the same level as the previous nonempty line, and **M-x c-indent-command** adjusts the indentation in steps specified by 'c-basic-offset'. The indentation style has no effect in this mode, nor any of the indentation associated variables, e.g. '**c-special-indent-hook**'.		
Toggle Hungry Delete mode †	**<f12> <f4> DEL**	(**c-toggle-hungry-state** &optional ARG)	Toggle hungry-delete-key feature. Affects **<DEL>** and **C-d** keys. • Optional numeric ARG, if supplied, turns on hungry-delete when positive, turns it off when negative, and just toggles it when zero or left out. • When the hungry-delete-key feature is enabled (indicated by "/h" on the mode line after the mode name) the delete key gobbles all preceding whitespace in one step.
Toggle text alignment on pel-newline-and-indent-below See also: • Σ **Align** • Σ **Indentation** †	**<f11> M-RET**	(**pel-toggle-newline-indent-align**)	Toggle variable ***pel-newline-does-align*** for the local buffer: toggles how '**pel-newline-and-indent-below**' operates: If ***pel-newline-does-align*** is t, it aligns several syntactic element in the current block: the comments, the assignments.
	• 🎛 Identify modes where ***pel-newline-does-align*** is automatically activated (set to t) by adding the major mode to the list in the **pel-modes-activating-align-on-return** user option. • This affects the behaviour of the following commands: pel-cc-newline (assigned to **RET** in CC modes. pel-newline-and-indent-below (assigned the **M-RET**)		
Toggle auto-newline insertion mode †	• **C-c C-a** • **<f12> <f4> M-RET**	(**c-toggle-auto-newline** &optional ARG)	Toggle **auto-newline** feature. • Optional numeric ARG, if supplied, turns on auto-newline when positive, turns it off when negative, and just toggles it when zero or left out. • Turning on auto-newline automatically enables ***electric indentation***. • When the auto-newline feature is enabled (indicated by "/la" on the mode line after the mode name) newlines are automatically inserted after special characters such as brace, comma, semi-colon, and colon.
Change RET key behaviour: select return mode. †	**<f12> <f4> RET**	(**pel-cc-change-newline-mode**)	Change the way the RET key behaves in the CC modes and display the new mode in the echo area. Changes from one mode to the next and then rotate to the first one. The modes are: 1. context-newline : the default : uses (**c-context-line-break**) with the extra ability to repeat its execution with an argument. 2. newline-and-indent: uses (**newline** ARG t) to insert newline and indent. 3. just-newline-no-indent: uses (**electric-indent-just-newline** ARG)
	☛ Emacs default is to use newline. PEL sets the default to c-context-line-break which provides more functionality for CC modes. A mode change is local to the current buffer and does not affect RET key behaviour in the other buffers using the same mode. 🎛 PEL user option **pel-initial-c-newline-mode** can be set to change the default for c-mode.		
Electric Keys	The following **electric C characters** have special meaning when the electrical state is active in a buffer using awk-mode. • Toggle electric behaviour in the current buffer with: with c-toggle-electric-state (**C-c C-l** or **<f12> <f4> e**).		
( )	• **( )**	(**c-electric-paren** ARG)	Insert a parenthesis.
	• If 'c-syntactic-indentation' and 'c-electric-flag' are both non-nil, the line is reindented unless a numeric ARG is supplied, or the parenthesis is inserted inside a literal. • Whitespace between a function name and the parenthesis may get added or removed; see the variable 'c-cleanup-list'. • Also, if 'c-electric-flag' and 'c-auto-newline' are both non-nil, some newline cleanups are done if appropriate; see the variable 'c-cleanup-list'.		
{ }	• **{ }**	(**c-electric-brace** ARG)	Insert a brace.
	• If 'c-electric-flag' is non-nil, the brace is not inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: a) If the auto-newline feature is turned on (indicated by "/la" on the mode line) newlines are inserted before and after the brace as directed by the settings in 'c-hanging-braces-alist'. b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. c) If auto-newline is turned on, various newline cleanups based on the settings of 'c-cleanup-list' are done.		
:	**:**	(**c-electric-colon** ARG)	Insert a colon.
	• If 'c-electric-flag' is non-nil, the colon is not inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: a) If the auto-newline feature is turned on (indicated by "/la" on the mode line) newlines are inserted before and after the colon based on the settings in 'c-hanging-colons-alist'. b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. c) If auto-newline is turned on, whitespace between two colons will be "cleaned up" leaving a scope operator, if this action is set in 'c-cleanup-list'.		
; ,	• **; ,**	(**c-electric-semi&comma** ARG)	Insert a comma or semicolon.
	• If 'c-electric-flag' is non-nil, point isn't inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: a) When the auto-newline feature is turned on (indicated by "/la" on the mode line) a newline might be inserted. See the variable 'c-hanging-semi&comma-criteria' for how newline insertion is determined. b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. c) If auto-newline is turned on, a comma following a brace list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-cleanup-list'.		
Electric pairs	It is also possible to control the insertion of character pairs by activating the **electric-pair-mode** in the buffer. • Type the first of a pair to insert this one and its matching character for (), [], {}, "" and ''. • When the electric-pair-mode is active in a buffer the mode-line lighter set by the pel-electric-pair-lighter is shown. This defaults to ε(ǀ)		
Toggle electric-pair-mode in current buffer † **Lighter:=** ε(ǀ)	**<f11> M-e**	(**electric-pair-local-mode** &optional ARG)	Toggle automatic parens pairing (Electric Pair mode) and org-mode special pair electric keys only in this buffer. With this typing (inserts the matching). Same for other pairs. • With a prefix argument ARG, enable Electric Pair mode if ARG is positive, and disable it otherwise. • Electric Pair mode is a global minor mode. When enabled, typing an open parenthesis automatically inserts the corresponding closing parenthesis, and vice versa. (Likewise for brackets, etc.). If the region is active, the parentheses (brackets, etc.) are inserted around the region instead.

Description	Keystroke	Function	Note
Insert New Line(s)	The behaviour of the RET key depends on whether the CC Mode electric mode is active or not. When it is not active it simply inserts a new line. When it is active the point also moves to the proper indentation according to the syntactic context. The following commands can also be used. • With PEL the default behaviour can be selected by customization and modified dynamically for the current buffer with the **pel-cc-change-newline-mode** command (bound to **<F12> M-RET**) see the CC-Mode behaviour control section above. • The pel-cc-newline command also aligns comments and assignment in the code block if the **pel-modes-activating-align-on-return** user option list includes the current major mode. The state for the current buffer can also be modified by the **pel-cc-change-newline-mode** command (**`<f11> M-RET`**).		

Insert a new line and operate according to the currently active selected return mode.
 	
With PEL, modify behaviour with <F12> M-RET.
 	
See also:
• Σ **Filling/ Justification**
	`RET`	(**pel-cc-newline** &optional N)	Insert a newline and perhaps align. With argument N repeat N times.
			• For newline insertion, operate according to the value of the variable 'pel-cc-newline-mode' which selects one of 3 commands (see the full description in the 3 row below):
			  • c-context-line-break (PEL default for RET)
			  • newline (Emacs default for RET)
			  • electric-indent-just-newline
			• If 'pel-newline-does-align' is t, then do the text alignment done by the function 'align'.

	Use : (**c-context-line-break**) : Do a line break suitable to the context. • When point is outside a comment or macro, insert a newline and indent according to the syntactic context, unless 'c-syntactic-indentation' is nil, in which case the new line is indented as the previous non-empty line instead. • When point is inside the content of a preprocessor directive, a line continuation backslash is inserted before the line break and aligned appropriately. The end of the cpp directive doesn't count as inside it. • When point is inside a comment, continue it with the appropriate comment prefix (see the 'c-comment-prefix-regexp' and 'c-block-comment-prefix' variables for details). The end of a C++-style line comment doesn't count as inside it. • When point is inside a string, only insert a backslash when it is also inside a preprocessor directive.		
	Use: (**newline** &optional ARG INTERACTIVE): Insert a newline, and move to left margin of the new line if it's blank. • With ARG, insert that many newlines. • If option 'use-hard-newlines' is non-nil, the newline is marked with the text-property 'hard'. • If 'electric-indent-mode' is enabled, this indents the final new line that it adds, and reindents the preceding line. • To just insert a newline, use M-x electric-indent-just-newline. Calls 'auto-fill-function' if the current column number is greater than the value of 'fill-column' and ARG is nil.		
	Use: (**electric-indent-just-newline** ARG): Insert just a newline, without any auto-indentation. With ARG, insert that many newlines.		

Insert an indented line below unbroken current line
See also:
Σ **Indentation**
	• **`M-RET`**
	• **`<f11> <tab> RET`**
		(**pel-newline-and-indent-below**)	Insert an indented line just below current line regardless of the position of point and move point to the beginning of the next line. Does not break current line.
			For example if point is at the beginning, middle or end of the line it just insert a new line below the current one at the proper indentation.
			• If ***pel-newline-does-align*** is t, it aligns several syntactic element in the current block: the comments, the assignments.
			  • You can toggle this on/off with **`<f11> M-RET`**.
			  • 🎛 Identify modes where ***pel-newline-does-align*** is automatically activated (set to t) by adding the c-mode to the list in the **pel-modes-activating-align-on-return** user option.

Insert a newline	**`C-j`**	(**electric-newline-and-maybe-indent**)	Insert a newline. • If 'electric-indent-mode' is enabled, that's that, but if it is *disabled* then:
	when disable: additionally indent according to major mode. Indentation is done using the value of 'indent-line-function': • In programming language modes, this is the same as TAB. • In some text modes, where TAB inserts a tab, this command indents to the column specified by the function 'current-left-margin'.		
Open New Line in Context See also: • Σ **Whitespace**	**`C-o`**	(**c-context-open-line**)	Insert a line break suitable to the context and leave point before it.
	• This is the '**c-context-line-break**' equivalent to '**open-line**', which is normally bound to **`C-o`**. See 'c-context-line-break' for the details. ☝ Normally C-o is bound to open-line. PEL rebinds it to c-context-open-line for the CC modes. If you want to open the line without indenting the next use open-line via **`<f12> C-o`**		
Open new line	• **`<f12> C-o`** • **`M-<f12> C-o`**	(**open-line** N)	Insert a newline and leave point before it. With arg N, insert N newlines. • If there is a fill prefix and/or a 'left-margin', insert them on the new line if the line would have been blank.

Comment/un-comment
 	
★★
See also:Σ **Comments**
	`M-;`	(**pel-c-comment-dwim** ARG)	Comment line or region with **//** or **/* */** style comments depending on the comment style currently used in the buffer.
			• When no marked region and no comment:
			  • On empty line: insert comment starter at the proper indentation level.
			  • Typed again: move it toward end of line.
			  • On line with code: insert comment starter after the code for an end-of-line comment
			• With marked un-commented region:
			  • 🎛 Comment region with style selected by **pel-c-multiline-comments** user-option:
			  • default (like comment-dwim): each line is commented with a **/* */**
			  • 1: single start multi-line comment (see example in box above)
			  • 2: double star multi-line comment (see example in the box above)
			• With marked commented region:
			  • removes the comment.
			• When a prefix ARG is specified, call 'comment-kill'. Else, call 'comment-indent'.

Comment/un-comment See also:Σ **Comments**	**`C-c C-c`**	(**comment-region** BEG END &optional ARG)	Comment or uncomment each line in the region. • With just **`C-u`** prefix arg, uncomment each line in region BEG .. END.
	• Numeric prefix ARG means use ARG comment characters. If ARG is negative, delete that many comment characters instead. • The strings used as comment starts are built from '**comment-start**' and '**comment-padding**'; the strings used as comment ends are built from '**comment-end**' and 'comment-padding'. By default, the '**comment-start**' markers are inserted at the current indentation of the region, and comments are terminated on each line (even for syntaxes in which newline does not end the comment and blank lines do not get comments). This can be changed with '**comment-style**'.		

Fill current paragraph
See also:
Σ **Filling/Justification**
	• **`M-q`**
	• **`<f12> F`**
	• **`M-<f12> F`**
		(**c-fill-paragraph** &optional ARG)	Like **`<f11> t f p`**.
			• If any of the current line is a comment or within a comment, fill the comment or the paragraph of it that point is in, preserving the comment indentation or line-starting decorations.
			• If point is inside multiline string literal, fill it. This currently does not respect escaped newlines, except for the special case when it is the very first thing in the string. The intended use for this rule is in situations like the following:
			
			`char description[] = "\`
			`A very long description of something that you want to fill to make nicely formatted output.";`
			
			• If point is in any other situation, i.e. in normal code, do nothing.
			• Optional prefix ARG means justify paragraph as well.

Toggle subword-mode See also: • Σ **Text Modes**	• **`<f11> t m b`** • **`<f12> M-b`** • **`M-<f12> M-b`**	(**subword-mode** &optional ARG)	Toggle subword-mode: a minor mode that treats sections of camelCase and PascalCase as distinct words. • With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise.
HIde/Show comments See also: • Σ **Comments**	**`<f11> ; ;`**	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see Σ **Comments**). 🔄 PEL activates it when the **pel-use-hide-comnt** user option is **t**.

Description	Keystroke	Function	Note

Hungry Deletion of Whitespace	The CC mode provides two commands that can perform "hungry whitespace deletion" that can also be used in every mode.
	• ☝ PEL provides the convenient keys with the **<f11>** prefix keys for those 2 commands, available in **all** modes.
	• In modes compatible with the CC Mode (e.g. for AWK, C, C++, D, Java, Pike, etc..) it is also possible to activate the Hungry Delete Mode to modify the behaviour of the simple **<DEL>** and **C-d**, to perform hungry deletions. That's not currently supported in other modes.
	  • When the Hungry Delete Mode is on, the mode-line displays a 'h' to the right of the '//l' indication of electric mode.
	  • The Hungry Mode also activates the key prefixes below that start with **C-c**. They are listed but remember they are only available once the Hungry state mode is activated (and that can only be done in modes that are CC Mode compatible).
	  • In modes derived from CC Mode you can also activate the hungry state to make standard delete commands delete hungrily, but that does not work for other modes. PEL provides the **<f12> M-DEL** key for those modes (like C).
	• Toggle hurry deletion mode of the **DEL** and **C-d** key for the current buffer with **c-toggle-hungry-state** (**<f12> M-DEL**).		

Delete preceding char or all preceding whitespace. See also: • Σ **Cut & Paste**	• **C-c DEL** • **C-c ⌫** • **C-c C-⌫** • **C-c C-<backspace>** • **C-c C-DEL** • **<f11> ⌫ ⌫** • **<f11> DEL DEL**	(**c-hungry-delete-backwards**)	Delete the preceding character or all preceding whitespace back to the previous non-whitespace character. ☞ In terminal mode, even though **C-⌫**, **C-<backspace>** and **C-DEL** are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. ☝ With PEL, the **<f11> ⌫ ⌫** binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.
Delete next char or all following whitespace. See also: • Σ **Cut & Paste**	• **C-c C-d** • **C-c ⌦** • **C-c C-⌦** • **C-c C-<delete>** • **<f11> ⌦**	(**c-hungry-delete-forward**)	Delete the following character or all following whitespace up to the next non-whitespace character. ☞ In terminal mode, even though **C-⌦** and **C-<delete>** are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. ☝ With PEL, the **<f11> ⌦** binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.
Indentation	All syntactic indentation control for AWK is controlled by the CC-Mode state, the style and whether electric mode for some characters is active. See CC Mode behaviour control section above. You can also explicitly request indentation using the commands below. • The first set of commands perform syntactic indentations s controlled by the CC Mode. • Rigid indentation commands are also available and listed at the end of this list. They are also listed in the Σ **Indentation** table.		

Indent current line or region
 	
See also:
• Σ **Indentation**
☝ This might seem strange for new Emacs users, but it ends up being very useful. You can type **<tab>** anywhere in the line to adjust the indentation of the current line or everything in the marked area if a block is marked.
	<tab>	(**c-indent-line-or-region** &optional ARG REGION)	Indent active region, current line, or block starting on this line.
			• Behaviour depends on syntactic-indentation mode (enabled by default but can be toggled on/off with the **<f12> M-i** key):
			  • With syntactic-indentation on (the default):
			  • In Transient Mark mode, when the region is active, reindent the region.
			  • Otherwise, with a prefix argument, rigidly reindent the expression starting on the current line.
			  • Otherwise reindent just the current line.
			  • With syntactic-indentation off:
			  • **<tab>** always indent current line by one level
			  • **C-u - <tab>** or **M-- <tab>** always un-indent current line by one level.
			  • Indenting marked region is done without syntax knowledge and at the same level as previous line.
			☝ If you want to indent rigidly you can use:
			• **pel-indent-rigidly,** bound to **C-x <tab>** and to **<f11> <tab><tab>** to indent the line or region rigidly.
			• **tab-to-tab-stop**, bound to **M-i** to insert spaces to the next tab stop column.

Indent lines of list after point See also: • Σ **Indentation**	**C-M-q**	(**indent-pp-sexp** &optional ARG)	Indent each line of the list starting just after point, or pretty-print it. • A prefix argument (**C-u**) specifies pretty-printing. Pretty-printing essentially uses more lines as it places the beginning of each list on a new line.
Indent current function or class	**C-c C-q**	(**c-indent-defun**)	Indent the content of the current top-level function or class. Leaves point unchanged.

Indent a region	**C-M-**	(**indent-region** START END &optional COLUMN)	Indent each nonblank line in the region.
			• A numeric prefix argument specifies a column: indent each line to that column.
			• With no prefix argument, the command chooses one of these methods and indents all the lines with it:
			  1. If 'fill-prefix' is non-nil, insert 'fill-prefix' at the beginning of each line in the region that does not already begin with it.
			  2. If 'indent-region-function' is non-nil, call that function to indent the region.
			  3. Indent each line via 'indent-according-to-mode'.
			☝ When a region is marked you can also use the simple **<tab>** to do the same when syntactic-indentation is active.

Non Syntactic Indentation	Emacs provides the following command to indent without regards to semantics. More information on indentation is available in the Σ **Indentation** table. 🍀 For most editing scenarios, it's best to set **pel-c-tab-width** and **pel-c-indent-width** to the same value: the first 2 commands use the value of pel-c-tab-width while the other 2 use pel-c-indent-width.		
Insert spaces or tabs to next defined tab-stop column See also: • Σ **Indentation**	**M-i**	(**tab-to-tab-stop**)	Insert spaces or tabs to next defined tab-stop column. • The exact location of the next tab stop is identified by the value of the **tab-stop-list** and **tab-width** for the current buffer. • With PEL, the tab-stop interval is controlled by the value of **pel-c-tab-width.** • PEL sets **tab-width** to the value of pel-c-tab-width for each c-mode buffer.

Indent/Unindent rigidly
 	
See also:
• Σ **Indentation**
• Σ **Key-Chords**
	• **C-x <tab>**
	• **<f11> <tab>**
	<tab>
	• **<tab>q**
		(**pel-indent-rigidly &optional N**)	Indent rigidly the marked region or current line N times tab-width columns.
			• **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.

	✂ PEL rebinds this key, but it extends the functionality: pel-indent-rigidly uses the original indent-rigidly.
	indent-rigidly 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>**.
	Both of 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-<left>** indent-rigidly-left-to-tab-stop **S-<right>** indent-rigidly-right-to-tab-stop
	• **<left>** indent-rigidly-left **<right>** indent-rigidly-right
	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.
	  • With PEL, the tab-stop interval is controlled by the value of **pel-awk-tab-width.**
	  • PEL sets **tab-width** to the value of pel-c-tab-width for each c-mode buffer.
	⚠ To invoke this command when **cua-mode** is active, type it really fast or type **C-x C-x <tab>** (or use the PEL binding **<f11> <tab> <tab>**).		

Description	Keystroke	Function	Note

Indent line(s) rigidly
 	
See also:
• **Indentation**
	• **<f6> <tab>**
	• **<f11> <tab> c**
		(**pel-indent-lines** &optional N)	Indent current or marked lines by N indentation levels controlled by **pel-c-indent-width**.
			• Works with point anywhere on the line.
			• All lines touched by the region are indented. A special argument N can specify more than one indentation level. It defaults to 1. If a negative number is specified, 'pel-unindent-lines' is used. If a region is marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all characters in all affected lines. Use **C-g** to de-activate the region.
			• Handles presence of hard tabs:
			  • If indent-tabs-mode is non-nil the indentation is created with a mix of hard-tabs and space characters.
			  • If indent-tabs-mode is nil, any hard tab in the indentation of the marked lines is replaced by the proper number of spaces. Hard tabs after first non-whitespace character on the line are left.

Un-indent line(s) rigidly
 	
See also:
• **Indentation**
	• **<backtab>**
	• **<f6> <backtab>**
	• **<f11> <tab> C**
		(**pel-unindent-lines** &optional N)	• Un-indent current line or marked lines by N indentation levels controlled by **pel-awk-indent-width**.
			• Works with point is anywhere on the line.
			• All lines touched by the region are un-indented. If region was marked, the function does not deactivate it to allow repeated execution of the command.
			• If a region was marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all characters in all affected lines. Use **C-g** to de-activate the region.
			• Handles presence of hard tabs: If **indent-tabs-mode** is:
			  • **non-nil** the indentation is created with a mix of hard-tabs and space characters.
			  • **nil**, any hard tab in the indentation of the marked lines is replaced by the proper number of spaces. Keeps hard tabs after first non-whitespace character.

Open file at point
 	
See also: **File mngt**
⚠ After changing user-options you currently must restart PEL by executing **pel-init** or restarting Emacs! ⚠
			The following command allow opening files from the file name taken at point (the cursor location).
			• In a awk-mode buffer the command is specialized to be more useful for AWK programming and has the extra capability of searching files where header files are stored. The search method is controlled by the following user-options:
			  • **pel-awk-file-finder-method** : identifies one of 4 supported method of identifying the header files. See their descriptions below.
			  • **pel-awk-file-searched-extra-dir-trees** : List of extra directory trees also searched by the tool identified by **pel-ffind-executable** user-option.
			  • **pel-awk-file-finder-ini-tool-name** : The name of a tool chain ***TTT***, to select one of the ***TTT-c-path*** tool-chain key inside the [file-finder] section of the ***pel.ini*** file, a INI-format configuration file. The value mapped to that key identifies the list of directories to search for that tool-chain. The name of the tool chain can be overridden by the value of the environment variable PEL_CC_FIND_TOOLCHAIN, which is read and used when Emacs starts up (or pel-init is executed). Use the command **pel-cc-set-file-finder-ini-tool-name** to change the currently used tool chain name.
			☝ Note that when using the Ido completion mode, it is possible to instruct Ido to use a file name at point as the basis for the file name to open.
			  • This Ido behaviour is controlled by the **ido-use-filename-at-point** user-option. With PEL you can control it globally or locally with **<f11> f M-.**

Show active file finder setup for current buffer	• **<f12> <f4> f** • **<M-f12> <f4>**	(**pel-cc-find-show-status** &optional APPEND)	Print AWK specific PEL file finding control user-options and variables info inside a *pel-cc-ffind-status* help-mode buffer. • Prints current state and values of relevant user-options and variables as buttons you can use to get more info and change the values of the user options. • Clear previous buffer content by default. Use prefix arg (like **C-u**) to append instead.
Set name of Tool-Chain specific include path. • (with **pel-ini-file** search method)	• **<f12> <f4> M-<f6>** • **<f12> <f4> <f54>**	(**pel-cc-set-file-finder-ini-tool-name** &optional TOOL-NAME)	Change activate value of tool-chain name key identified by value of **pel-c-file-finder-ini-tool-name** user-option. The change is not persistent. Requires **pel-use-ini** set to t. Used when the **pel-c-file-finder-method** is set to **pel-ini-file.** In that case it effectively select a new set of tool-chain specific directories to search by **pel-open-at-point**. The directories are identified by the corresponding ***TTT-c-path*** *key in the* **[file-finder]** section of the ***pel.ini*** file.

Open file or web-page whose name is at point
★★
 	
☝
AWK file inclusion finding control ☛
 	
pel-use-ini ☛
 	
Command is also specialized for:
• **reStructuredText**
• **C**
• **C++**
• **Erlang**
• **UNIX Shell**
 	
Generic Delimiting characters
 	
⚠ The complete file detection heuristic is described in the **File mngt** description of the same command.
 	
Select target window ☛
 	
N>20 : open the directory ☛
 	
See function docstring for more info.
	• **M-***
	• **<f11> f .**
	• **6y**
		(**pel-open-at-point** &optional N)	Open the file, library or the URL, named at point, with potential line & column #s.
			• If necessary will search source code files in current project as specified by **pel-filename-at-point-finders** user-option. Type **<f12> <f4> ?** to **show current file search method**.
			Supports glob characters, partial directory path. When multiple files are found it prompts using the method selected by **pel-prompt-read-method** user-option.
			The **6y** key-chord is available if **pel-use-key-chord** is t. See **Key-Chords**.
			
			This command works generically but is also specialized for AWK major mode: it opens the header file identified by the `@include` statement.
			Aside from generic method described below, the command searches for the file to open using the method identified by the **pel-awk-file-finder-method** and the **pel-awk-file-searched-extra-dir-trees** user-options. The first one identifies one of the following search method, the other identifies extra directory tree(s) to search using the search tool identified by the **pel-ffind-executable** user-option:
			• **generic**: the command searches, in current directory and its parents, for a file identifying the parent root directory; a file with a name identified in the **pel-project-root-identifiers** user-option. Something like .git, .hg, .project or .pel-project by default. Then searches for files inside that directory tree.
			• **pel-ini-file**: the command searches inside directories identified by lists defined in the ***pel.ini*** file which PEL identifies for the project like it does for project marker. The ***pel.ini*** file is a **.INI file format**. When found, it is opened and information inside the file identifies where to search.
			• The file must contain a **[file-finder]** section with:
			  • The **project-path** key. The value is a list of directories to search recursively.
			  • One or several ***TTT-awk-path*** key(s), where ***TTT*** is a tool-chain name. The value is a list of directories to search recursively for that tool-chain.
			  • The currently used tool chain is identified by the following values in order (first one takes priority on startup):
			  • The content of the PEL_CC_FIND_TOOLCHAIN environment variable, if it exists.
			  • The content of the **pel-awk-file-finder-ini-tool-name** user-option; which identifies the name of a ***TTT-awk-path*** key.
			  • The paths identified in the two lists may use environment variables inside the path strings. Use the `$VARNAME` format to identify them.
			  • You can modify this tool chain name anytime during an editing session by typing **<f12> <f4> M-<f6>** and specifying another name.
			  • With several ***TTT-awk-path*** keys inside the ***pel.ini*** file, you can adjust the include path dynamically for various tool chains.
			• **environment variable name**: the name of an environment variable (like INCLUDE) that holds a list of directory names to search files in.
			  • Directories are not searched recursively for the last 2 options.
			• **explicit lists**: two lists of directory names: one list holds the project directory names, the other hold the tool and library directory names. The lists may identify directory names indirectly via environment variables. The `$VARNAME` format must be used. Directories are not searched recursively.
			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.
			• The generic mode extraction works by identifying the beginning & end of the file/directory/library/URL name string by delimiter characters, one of: tab, newline and: **"`'|()[]{}<>** ‘’“” 「」 〔〕 〈〉 《》 【】 〖〗 «»‹›❮❯❬❭ 〘〙 ·。
			
			When finding several file names, 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.
			  • ☝ Use this to detect potential duplication in header file names in large include paths.
			
			The command opens the file in the window selected by the following logic controlled by presence or absence of typed numerical prefix arguments:
			• **Select target window:**
			  • Without argument:
			  • If file or directory 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 according to 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 prefix numeric argument N:
			  • N < 0 : create a new window and use that.
			  • (abs N) > 20: then open the **directory** instead of the file. Interpret the window position from the N value adjusted: N-20 (or N+20 if N is negative)
			  • 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: force **opening the file in the OS associated application** (with N=29 or N=-29, open the file's directory with the OS associated application (eg. macOS Finder, Windows Explorer). If this is a URL, open it in the OS default web browser.
			  • Selecting Minibuffer, inexistent or dedicated window is not allowed.

Description	Keystroke	Function	Note
Inserting code			
Insert Parentheses	`M-(`	(**insert-parentheses** &optional ARG)	For C++: insert a parenthesis pair '()', leaving point after open-paren. • A positive ARG encloses the following ARG sexps in parenthesis if they are balanced. • A negative ARG encloses the preceding ARG sexps instead. • No argument is equivalent to zero: just insert '()' and leave point between. • PEL makes 'parens-require-spaces' buffer local and set it to nil in C++ mode buffers, allowing the use of this command to insert the argument parentheses following a function (and without placing a space between the function name and the opening parenthesis. • If region is active, insert enclosing characters at region boundaries. • This command assumes point is not in a string or comment.
Marking	Emacs provides the following command to quickly mark the whole content of the current function. More mark commands exists, see the Σ **Marking** table.		
Mark the complete function body See also: Σ **Marking**	`C-M-h`	(**c-mark-function**)	Mark complete function. • Put mark at end of the current top-level declaration or macro, point at beginning. • If point is not inside any then the closest following one is chosen. Each successive call of this command extends the marked region by one function. • A mark is left where the command started, unless the region is already active (in Transient Mark mode). • As opposed to C-M-a and C-M-e, this function does not require the declaration to contain a brace block.
Getting Syntactic Information	Use the following commands to extract syntactic information from the source code.		
Display name of current function	• `C-c C-z` • `<f12> f` • `M-<f12> f`	(**c-display-defun-name** &optional ARG)	Display the name of the current CC mode defun and the position in it. • With a prefix arg, push the name onto the kill ring too.
Search Support	In C++ mode, the superword mode can be useful since snake_case is often used. Using superword-mode helps searching. PEL activates the superword mode by default in C++ mode. To change this use the `<f11> t <f2>` to access the customize buffer.		
Toggle superword-mode See also: • Σ **Text Modes** • Σ **Search/Replace**	• `<f11> t m p` • `<f12> M-p`	(**superword-mode** &optional ARG)	Toggle superword-mode: a minor mode that treats snake_case as one word. In C++ '_' are treated as part of words. • With a prefix argument ARG, enable superword mode if ARG is positive, and disable it otherwise. • PEL provides the `<f12> M-p` key for the programming language modes where snake_case is popular (Emacs Lisp, C, C++, Erlang, Python, etc…)
Highlighting blocks	The following commands can be used to activate or toggle useful modes to highlight blocks of (), {}, and []. • show-paren-mode, which highlights the parens that matches the one before or after point. • rainbow delimiters mode, where matching nested parens are highlighted with the same colour.		
Toggle show-paren mode on/off See also: Σ **Highlight**	• `<f12> M-9` • `M-<f12> M-9` • `<f11> h (`	(**show-paren-mode** &optional ARG)	Toggle visualization of matching parens (Show Paren mode). • With prefix argument ARG, enable Show Paren mode if ARG is positive, disable it otherwise. • Show Paren mode is a global minor mode. When enabled, any matching parenthesis is highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.
Enable/Disable coloured highlight of nested blocks (),{},[] See also: Σ **Highlight**	• `<f12> M-r` • `M-<f12> M-r` • `<f11> h R`	(**rainbow-delimiters-mode** &optional ARG)	Highlight nested parentheses, brackets, and braces with colours according to their depth. • Customize the depth and colours with **M-x customize-group rainbow-delimiters** **Requires:** rainbow-delimiters.el PEL activates this when the **pel-use-rainbow-delimiters** user option is set to **t**.
Navigation in AWK	This current list below describe the specialized commands only. See the others inside Σ **Navigation** Note that navigation in AWK is similar to navigation in C.		
• **By definitions**	Move to the definition of function or type at point. See Σ **Xref** for more information to activate the various engines that support cross referencing for C code.		
Find definition of identifier at point See also: Σ **Xref**	`M-.`	(**xref-find-definitions** IDENTIFIER)	Grab symbol at point and move cursor to its definition. • If there are more than one match, prompt in the *xref* buffer. • To search for a symbol entered manually, type **`C-u M-.`** • With dumb-jump this performs a search using ag, ripgrep or git grep if available.
Go back to where M-. was last issued	`M-,`	(**xref-pop-marker-stack**)	• Pop back to where M-. was last invoked. • Marker depth is controlled by the **xref-marker-ring-length** user option.
• **By statements**	Move to beginning /end of statement or comment.		
Go to beginning of statement (backward)	`M-a`	(**c-beginning-of-statement** &optional COUNT LIM SENTENCE-FLAG)	Go to the beginning of the innermost statement. • With prefix arg, go back N - 1 statements. • If already at the beginning of a statement then go to the beginning of the closest preceding one, moving into nested blocks if necessary (use **`C-M-b`** to skip over a block). • If within or next to a comment or multiline string, move by sentences instead of statements.
Go to the end of statement (forward)	`M-e`	(**c-end-of-statement** &optional COUNT LIM SENTENCE-FLAG)	Go to the end of the innermost statement. • With prefix arg, go forward N - 1 statements. • Move forward to the end of the next statement if already at end, and move into nested blocks (use **`C-M-f`** to skip over a block). • If within or next to a comment or multiline string, move by sentences instead of statements.
Go to start of current switch statement	`<f6> t w s`	(**pel-cc-to-switch-begin**)	Move point to the start { of current switch statement, if any. • If point is inside switch statement, mark position before moving point. Move it back with **`M-``**. • If point is not inside a switch statement, issue a user error.
Go to end of current switch statement	`<f6> t w e`	(**pel-cc-to-switch-end**)	Move point just past the end } of current switch statement, if any • If point is inside switch statement, mark position before moving point. Move it back with **`M-``**. • If point is not inside a switch statement, issue a user error.

Description	Keystroke	Function	Note

• **By blocks**
  • **functions**
  • **structures**
	• Move to beginning /end of function definition blocks or structure definition blocks.
	• Move across C++ statements and C++ scope blocks, or any group of ( ), [ ], { } or < > blocks. Jump over comments.
	• ☝ When point is located before opening brace or right after closing brace and **show-paren-mode** is on, the matching parentheses are highlighted.
	• ☝ Both **<f12>** and **<M-f12>** key prefixes are available for several bindings to ease typing some sequences. The one easier to type is identified in **bold**.		

Move block forward See also: • Σ **Navigation** ☝ • Use this to move to end of next syntax element or to end of block when already **outside** the block. • Use **C-M-u** to exit a block **(see below)**.	• **<f12> <right>** • <M-f12> <right> • **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. • ⚠ 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, otherwise it does something else. ❖ **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.
Forward block/list See also: Σ **Navigation**	**C-M-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.
Move block backward See also: • Σ **Navigation**	• **<f12> <left>** • <M-f12> <left> • **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. • ⚠ 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, otherwise it does something else. ❖ **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.
Backward block/list See also: Σ **Navigation**	**C-M-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.
Backward to beginning of current top-level function or struct	**C-M-a**	(**c-beginning-of-defun** &optional ARG)	Move backward to the beginning of a function or type definition. • With a positive argument, move backward that many functions or structures. A negative argument -N means move forward to the Nth following beginning.
	• **<f12> <up>** • <M-f12> <up>	(**beginning-of-defun** &optional ARG)	Move backward to the beginning of function or type definition. • Move point before the function type or the struct or typedef keyword. • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. ☛ Shift marking is available. With **<f6>** and **<f12>** hit Shift after function key, before cursor key. ⚠ This command moves to the beginning go the next function or of the same nesting level of the current location. It skips the functions that are more deeply nested.
	C-M-<home>		
Forward to end of current top-level function or struct.	**C-M-e**	(**c-end-of-defun** &optional ARG)	Move forward to the end of a top level declaration. • With argument, do it that many times. Negative argument -N means move back to Nth preceding end.
	• **<f12> <down>** • <M-f12> <down>	(**end-of-defun** &optional ARG)	Move forward to the end of next function or type definition. With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. ☛ Shift marking is available. With **<f6>** and **<f12>** hit Shift after function key, before cursor key. ⚠ This command moves to the end of the next **top-level** function. It skips nested functions.
	C-M-<end>		
Backward to end of previous top level function or struct	• <f12> <M-up> • **<M-f12> <M-up>**	(**pel-end-of-previous-defun** &optional SILENT DONT-PUSH_MARK)	Move backwards to the end of the previous function or type 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>**. ☛ Shift marking is available. With **<f6>** and **<f12>** hit Shift after function key, before cursor key. ⚠ In some cases it fails to detect the end of the previous block and fails. 🚧
Forward to start of next top level function or struct ☝ • Use this to move from the top of the file to the first block.	• <f12> <M-down> • **<M-f12> <M-down>**	(**pel-beginning-of-next-defun** &optional SILENT DONT-PUSH_MARK)	Move forward to the beginning of the next function or type definition. • Move point before the function type or the struct or typedef keyword. • 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>**. ☛ Shift marking is available. With **<f6>** and **<f12>** hit Shift after function key, before cursor key. ☝ 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.
• **in/out of blocks**	• Move in or out of C scope blocks, or any group of ( ), [ ], { } or < > blocks.		
Backward Up/ outside sexp hierarchy See also: • Σ **Navigation**	• **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 or nested blocks. • 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-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.

Description	Keystroke	Function	Note
Forward Up/outside sexp hierarchy See also: **Navigation**	`C-M-]`	(**up-list** &optional ARG ESCAPE-STRINGS NO-SYNTAX-CROSSING)	Move forward out of one level of parentheses or nested blocks. • Also work on other parentheses-like expressions defined by the current language mode. • With ARG, do it that many times. Negative arg means move backward but to a less deep spot.
Down/inside sexp/ block See also: • **Navigation**	• **`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. • Also work on other parentheses-like expressions defined by the current language mode. • With ARG, do it that many times. Negative arg mans move backward but still go down a level. • This command assumes point is not in a string or comment. • ⚠ With PEL: if you want to use **`Esc C-<down>`** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. • **`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.
Programming Help	PEL has bindings for the following commands that are useful when editing source code, markup files or any file that has a mode that supports imenu.		
Show what completion mode is currently used.	• **`<f11> ? M-c`** • **`<f11> M-c ?`**	(**pel-show-active-completion-mode**)	Display the completion mode currently used, and the Ido prompt geometry when appropriate. Show key bindings for changing other aspects of input completion.
Show function at point	**`<f11> ? F`**	(**pel-show-function**)	Display the name of the current "*function*" at point in the mini-buffer.

Toggle which-function-mode to display name of current function at point
 	
See also:
• **Menus**
• **Mode Line**
 	
☝ The concept of "*function*" is major mode specific. For example, in C++ mode, if point is inside a class definition it shows the name of the class.
	• **`<f11> ? f`**
	• **`<f11> M-d f`**
		(**which-function-mode** &optional ARG)	Toggle mode line display of current function (Which Function mode).
			• With a prefix argument ARG, enable Which Function mode if ARG is positive, and disable it otherwise.
			
			• The **which-function-mode** is a global minor mode. When enabled, the current function name is continuously displayed in the mode line.
			⚠ Detection of functions and variables depend on the **imenu** functionality. If you modify the content of a buffer, you need to force a menu rescan to get proper results. You can force a rescan with **pel-imenu-rescan**, bound to **`<f11> <f10> r`**.
			🎛 Identify major modes that automatically active the mode with **which-function-mode** user-option.
			• Use **`M-x customize-option which-function-mode`** to open the relevant customization buffer.
			• With PEL you can use:
			  • **`<f11> ? <f3>`** to access the which-func customization group. It will provide access to the customization group even when the feature has not yet been loaded, something that Emacs does not do by default.
			  • **`<f11> <f2> o which-function-mode RET`** to access the user-option directly.