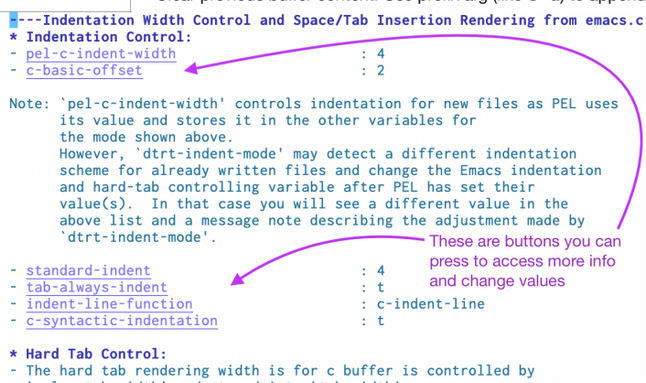
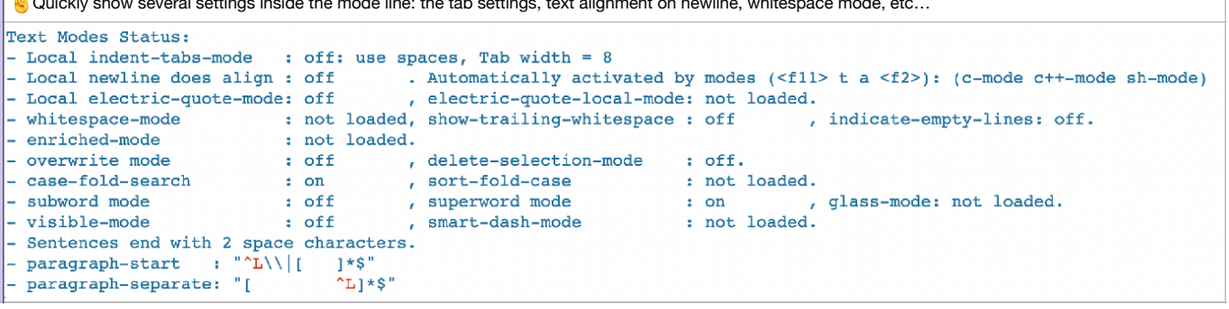
↖ Topic Index

Indenting & Tab

Description	Keystroke	Function	Note

Indentation under Emacs
○ Use hard tabs or spaces for indentation
• Set hard-tab visual width
• Adjust tab stop
• Show tab/indent settings
○ Align on return
• Insert hard-tab
• To next tab stop
• tabify & untabify
• Behaviour of tab key
• Insert newline, split line
• Indent region
• Delete indentation
• Move to 1st non-blank
○ Indenting /un-indenting rigidly
• Text alignment on newline
• Indent-tools
○ dtrt-indent, Indent-bars
• Smart-shift, Smart-tabs
○ **References**
	Emacs controls indentation and behaviour of the tab key according to various rules controlled by the buffer's **major mode**.
	• The modes default behaviour may be modified by the use of **minor modes**.
	  • Several major modes identify one or several variable that control indentation levels and behaviour. Some use default indentation variables.
	  • Some programming languages (**such as Go**) impose hard-tab for indentation, using tab for indentation and space for alignment. Works very nicely.
	  • There are several indentation schemes: hard-tab only for indentation, spaces only for indentation, hard-tabs for indentation space for alignment, free mix of hard-tabs and spaces for indentation and whitespace.
	  • Emacs can support all those schemes. It can tabify or untabify source code.
	  • Emacs controls the *display rendering* of hard tabs via the **tab-width** customizable user-option variable.
	  • The indentation width is often independent from the tab width but not always. Again it depends on the major mode used.
	PEL supports all indentation schemes and various indentation mechanisms and has its own extensions. For each explicitly supported mode you can configure the indentation with and use of hard tabs for the mode. This is used for all files you create. It also support automatic adjustment for files you edit that have been created by others and use a different indentation width and use of hard-tabs.
	• It provides activation of following external packages by turning on the corresponding pel-use customizable user-option (setting it to **t**):
	📦 The **dtrt-indent** external package — **pel-use-dtrt-indent** — Detects indentation width & use of hard-tabs in file, then adjust settings to adapt.
	📦 The **indent-bars** external package — **pel-use-indent-bars** — Provides ability to show indent bar at each indentation level.
	📦 The **indent-tools** external package — **pel-use-indent-tools** — Provides commands to alter indentation of code.
	📦 The **smart-shift** external package — **pel-use-smart-shift** — Provides ability to shift text areas left/right by indentation levels.
	📦 The **smart-tabs** external package — **pel-use-smart-tabs** — Use it when using tabs for indentation and spaces for alignment.
	🕶️ To help people that have visual problems with small indentation, PEL provides **pel-indent-with-tabs** which converts the indentation of the whole buffer to hard tabs of the same width as indentation width. Then you can change the indentation visual rendering with the **pel-set-tab-width** command. Later, revert to the conventional content with **pel-indent-with-spaces**.		

Last updated on:	2025-11-05	See **Showing Information About Indentation and Hard Tab Control under PEL**	
Open this PDF file. See also: **Σ Help/Info**	`<f11> <tab> <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **Σ Indentation** 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.
Σ Customize PEL highlighting control	`<f11> <tab> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL support for indentation management • If OTHER-WINDOW is non-nil (use `C-u`), display in other window.
Σ Customize Emacs indentation control	`<f11> <tab> <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs indentation control groups: indent, Indent-bars indent-tools, dtrt-indent, smart-shift. • If OTHER-WINDOW is non-nil (use `C-u`), display in another window.
Toggle use of hard tabs for indentation in the current buffer. Affect new code written.	The use of hard tabs or spaces for indentation is controlled by the Emacs (customizable) variable **indent-tabs-mode**. See also: **Σ Whitespace** • This variable has global impact, but can be overridden by directory local value, file local value and buffer local value.		
	• `<f11> <tab> m` • `<f11> t w I`	(**pel-toggle-indent-tabs-mode** &optional ARG)	Toggle whether indentation can insert hard tab characters in the current buffer. • If ARG is positive set to use hard tabs, otherwise force use of spaces only. • This uses Emacs **indent-tabs-mode** function and provides more feedback.
Indent whole buffer with tabs	`<f11> <tab> i <tab>`	(**pel-indent-with-tabs** &optional WITH-TAB-WIDTH)	Convert all indentation in current buffer to use tabs for indentation. • If the optional WITH-TAB-WIDTH numerical argument is specified use that otherwise prompt for the tab width to use.
Indent whole buffer with spaces	`<f11> <tab> i <space>`	(**pel-indent-with-spaces** &optional WITH-TAB-WIDTH)	Convert current buffer to use space for indentation. • Use the original scheme unless a the optional WITH-TAB-WIDTH numerical argument is specified. If an optional numerical argument is specified, use that for tab width.
Set hard-tab width: the *visual rendering* of hard tabs for the current buffer • **Does not change buffer/file content**	`<f11> <tab> w`	(**pel-set-tab-width** N)	Change the **tab-width** of the current buffer, only affecting the display rendering of hard tabs inserted in the buffer text. Prompts for a new value in the [2, 8] range.
	• This modifies a buffer local value of the the **tab-width** user-option. • PEL provides a specialized user-option to set the default value of **tab-width** for several major modes. For example, to change the tab width used for all Go source code files, change the '**pel-go-tab-width**' user-option variable instead. See the documentation of each major mode for more information.		The change is temporary and affects the current buffer only.
Tab stops	Emacs keeps track of a set of "*tab-stop*" columns that can be used as reference points to align text. Something similar to **typewriter tab-stop**.		
Change the tab stops • used by `M-i`	`<f11> <tab> e`	(**edit-tab-stops**)	Opens a ***Tab Stops* buffer**. Identify the tab stops in the first line with colons. • Use `C-c C-c` to activate and exit the buffer. • The tab stop take effect at the top of the buffer, as used by `M-i`
Show Indentation settings For several major modes, a PEL mode-specific user-option is used to set the indentation width for the major mode. Like **pel-c-indent-width** for C buffers. PEL stores the value into the indentation control variable used by the mode (like **c-basic-offset** in C buffers). These control the configured indentation for the mode which is used when new files are created. Files written by others may not conform to your customized indentation settings, but PEL can adapt when **pel-use-dtrt-indent** is on.	• `<f11> <tab> ?` • `<f11> ? <tab>`	(**pel-show-indent** &optional APPEND)	Print info about indentation control in a ***pel-indent-info*** help-mode buffer. • Buffer-specific values of relevant user-options as buttons to use to get more info and change their customized values. Includes major-mode specific ones. • Clear previous buffer content. Use prefix arg (like `C-u`) to append instead.
	• The first line shows the buffer name and date. • The first group of user options control the indentation. The PEL user-option is normally shown first if there is one for the mode. • The next group list user-options are less important but may also be used. • All user-option variables are **help buttons**; click or hit return on them to open help specific buffer about the variable. • If **pel-use-dtrt-indent** is on, the customized values may be overridden by dtrt-indent detection of indentation scheme.		
Toggle text alignment on pel-newline-and-indent-below See also: **Σ Align** • See the alignment control variable list with: `<f1> t a ?`	`<f11> M-RET`	(**pel-toggle-newline-indent-align**)	Toggle variable *pel-newline-does-align* for the local buffer.
	It affects the way function '**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. • 🎛️ set modes that automatically activates *pel-newline-does-align* by adding the major mode to **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 like c-mode, c++-mode and d-mode). **pel-newline-and-indent-below** (assigned the `M-RET`)		
Show state of text modes • whether hard tabs are used for indentation, tab-width • whether newline aligns text • electric-quote-mode • delete-selection-mode • enriched-mode • overwrite-mode • case folding • subword, superword, glass modes • visible-mode, smart-dash-mode • paragraph definition Output example ➡	`<f11> t m ?`	(**pel-show-text-modes**)	Display the state of the various text modes in the mini buffer.
	☝️ Quickly show several settings inside the mode line: the tab settings, text alignment on newline, whitespace mode, etc…		
			

Description	Keystroke	Function	Note
Align on return	The following commands control **indentation** on return. The behaviour can be modified and PEL provides a quick command to list relevant user options.		

Display current buffer's vertical alignment behaviour
See also: Σ **Align**
	`<f11> t a ?`	(**pel-align-info** &optional APPEND)	Print information about text aligning info in a *pel-align-info* **help-mode** buffer.
			• Prints current state and values of relevant user-options as buttons to help:
			  • `M-RET` behaviour in current buffer: show if that command aligns text or not.
			  • whether the modes are activating vertical alignment on return.
			• Clear previous buffer content. Use prefix arg (like **C-u**) to append instead.

Insert an indented line below current line See also: Σ **Align**	• `M-RET` • `<f11> <tab> RET`	(**pel-newline-and-indent-below**)	Insert an indented line just below current line. • Also align vertically if mode was activated for the buffer with `<f11> M-RET` . • See current behaviour with `<f11> t a ?`
Inserting hard tab	Regardless of `<tab>` key behaviour, it is possible to insert a **hard tab character** with `C-q <tab>`		
Insert Literal Tab	`C-q <tab>`	(**quoted-insert** ARG) <tab>	Inserts a hard tab inside buffer. Render text according to value of **tab-width**.
Insert whitespace to next tab-stop	Regardless of `<tab>` key behaviour, it is possible to insert spaces (or combinations of hard tab and spaces) to put point to the next "tab-stop" with `M-i` • The location of tab-stops can be modified with (**edit-tab-stop**), bound to `<f11> <tab> e` .		
Insert spaces or tabs to next defined tab-stop column	`M-i`	(**tab-to-tab-stop**)	Insert spaces or tabs to next defined tab-stop column. • Use `<f11> <tab> e` to modify position of the tab stops.
	• 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.		
Tabify & untabify	The following two commands can be used to **replace hard tabs in a file with the corresponding number of space characters** while retaining the same indentation and vice-versa.		
Replace tabs with spaces in a region See also: Σ **Whitespace**	`<f11> t w SPC`	(**untabify** START END &optional ARG)	Convert all tabs in region to multiple spaces, preserving columns. • If called interactively with prefix ARG, convert for the entire buffer. • First select a region (Use C-x h for selecting the whole file). Then use the *untabify* function to replace all tabs by spaces in that region.
Replace multiple spaces with tabs in a region See also: Σ **Whitespace**	`<f11> t w <tab>`	(**tabify** START END &optional ARG)	Convert multiple spaces in region to tabs when possible. • A group of spaces is partially replaced by tabs when this can be done without changing the column they end at. • If called interactively with prefix ARG, convert for the entire buffer.
Behaviour of Tab Key	In Emacs the behaviour of the `<tab>` key depends on the major mode of the current buffer. This key is rebound by several major mode. • In text modes, tabs default to inserting hard tabs corresponding to a alignment at every 8 columns. However, if there is text in the above lines, the tab moves to the spot under the word above. Note that if a line is full of text (without any space), then the tab stops controlled by the ruler take effect again. • In several programming modes the tab key does not insert anything, it adjusts the line indentation. according to surrounding code		
Indent current line (or region)	`<tab>`	(**indent-for-tab-command** &optional ARG)	Indent the current line or region, or insert a tab, as appropriate.
	• This function either inserts a tab, or indents the current line, or performs symbol completion, depending on 'tab-always-indent'. The function called to actually indent the line or insert a tab is given by the variable '**indent-line-function**'. • If a prefix argument is given, after this function indents the current line or inserts a tab, it also rigidly indents the entire balanced expression which starts at the beginning of the current line, to reflect the current line's indentation. • In most major modes, if point was in the current line's indentation, it is moved to the first non-whitespace character after indenting; otherwise it stays at the same position relative to the text. • If 'transient-mark-mode' is turned on and the region is active, this function instead calls 'indent-region'. In this case, any prefix argument is ignored.		
	⚠️🚧 The behaviour of the tab key vastly differ between major modes. This ranges from not moving the cursor at all if the indentation is identified as correct for the current context, to cycling through various potential positions to just what someone new to Emacs would expect. Much more has to be documented on the behaviour of that key and how it can be controlled and customized. It's quite possible that the best way to document its behaviour would be to place a description inside the table of each major mode.		
	`<tab>`	**indent-for-tab-command** &optional ARG)	In **Lisp** related modes. indent-line-function = indent-relative. • tab-always-indent = t
	☝ Several major modes adjust the behaviour of the tab key to perform semantically aware indentation, such as what is being done in Lisp.		
	`<tab>`	(**c-indent-line-or-region** &optional ARG REGION)	In C related modes: Indent active region, current line, or block starting on the line. • In Transient Mark mode, when the region is active, reindent the region. • With prefix argument, rigidly reindent the expression starting on current line. • Otherwise reindent just the current line.
Indent lines of list after point **Example: CLBC s3.lisp**	`C-M-q`	• (**indent-sexp** &optional ENDPOS) • (**c-indent-exp** &optional SHUTUP-P)	Indent each line of the list starting just after point. • The command used depends on the major mode of the current buffer.
Newline & indent			
Insert new-line and maybe indent	`C-j`	(**electric-newline-and-maybe-indent**)	Add new line and indent next line. Indentation is controlled by the variable **left-margin**. Pressing **Tab** anywhere on the line also indents the line properly.
Insert new-line and maybe indent	`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**'.
Split current line & indent	`C-M-o`	(**split-line** &optional ARG)	Split current line, moving portion beyond point vertically down. • If the current line starts with 'fill-prefix', insert it on the new line as well. • With prefix ARG, don't insert 'fill-prefix' on new line.

Indent 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'.

Indent relative to line above	`<f11> <tab> r`	(**indent-relative** &optional FIRST-ONLY UNINDENTED-OK)	Space out to under next indent point in previous nonblank line. An indent point is a non-whitespace character following whitespace.

	`The following line shows the indentation points in this line.`
	`    ^         ^    ^     ^   ^           ^       ^  ^    ^`
	• If FIRST-ONLY is non-nil (ie. using **C-u** prefix) then only the first indent point is considered.
	• If the previous nonblank line has no indent points beyond the column point starts at, then 'tab-to-tab-stop' is done, if both FIRST-ONLY and UNINDENTED-OK are nil, otherwise nothing is done.
	• If there isn't a previous nonblank line and UNINDENTED-OK is nil, call 'tab-to-tab-stop'.
	
	Essentially, this command inserts whitespace at point, until point is aligned with the first non-whitespace character on the previous line (actually, the last non-blank line). If point is already farther right than that, run tab-to-tab-stop instead—unless called with a numeric argument, in which case do nothing.		

Delete Indentation, join line to the previous one See also: Σ **Cut & Paste** Σ **Whitespace**	• `M-^` • `<f11> ⌫ 6` • `<f6> 6`	(**delete-indentation** &optional ARG)	Join this line to previous and fix up whitespace at join. • If there is a fill prefix, delete it from the beginning of this line. • With argument, join this line to following line.
Move to fist nonbank character on the line	`M-m`	(**back-to-indentation**)	Move point to the first non-whitespace character on this line.

Description	Keystroke	Function	Note
Indenting and un-indenting rigidly	The following commands provide non-semantic indentation of the current line or marked region. • The first command allows you to use further keystrokes to fine-tune the indentation back and forth using cursor keys. That's probably all you ever need to use. • Currently, PEL also provides the last 2 commands that indent or un-indent the current line or marked region. Once used, the region remains marked to allow further use of the command.		
Indent/Unindent rigidly See also: 𝕏 **Key-Chords** ✂ PEL rebinds this key, but extends the functionality: **pel-indent-rigidly** uses indent-rigidly, described below the dashed line.	• **C-x <tab>** • **<f11> <tab> <tab>** • **<tab>q**	(**pel-indent-rigidly &optional N**) - - - - - - - - - - - - - - - - ✂ PEL uses the above instead of the standard: (**indent-rigidly** START END ARG &optional INTERACTIVE)	Enter a mode to indent rigidly the marked region or current line N times. • **If a region is marked**, it uses 'indent-rigidly' and provides the same prompts to control indentation changes. • **If no region is marked**, it operates on current line(s) identified by the numeric argument N (or if not specified N=1): • N = [-1, 0, 1] : operate on current line • N > 1 : operate on the current line and N-1 lines below. • N < -1 : operate on the current line and (abs N) -1 lines above. Once the command is issued use the keys listed below to indent or un-indent by indent-width step or by single column. - 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>**.
See also: • 𝔓𝔏 - **C** • 𝔓𝔏 - **C++** • 𝔓𝔏 - **D** • 𝕄 **reStructuredText**	Both of these commands activate a transient mode where Emacs prompts for extra keys to control how to indent or un-indent. The capabilities are controlled by the variable *indent-rigidly-map* with by default provides: • **S-<right>** indent-rigidly-right-to-tab-stop **S-<left>** indent-rigidly-left-to-tab-stop • **<right>** indent-rigidly-right **<left>** indent-rigidly-left Typing any other key deactivates the transient mode. • The **S-<right>** and **S-<left>** keys indent/de-indent to the next tab-stop position, which is controlled by the **tab-width** user option. • With PEL, for several major modes, the indentation is controlled by a mode-specific user option variable . For example, for buffers in c-mode, the value of **pel-c-tab-width** is automatically stored into tab-width when the buffer is opened. ⚠️ If you use the cua-mode: the cua-mode uses **C-x**, 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>**). 📦🔄 With PEL, the **<tab>q** key-chord is available when **pel-use-key-chord** is non-nil. See 𝕏 **Key-Chords**. ☝️ Command numeric prefix **is available** with the key-chord binding.		
Indent line(s) rigidly	• **<f6> <tab>** • **<f11> <tab> c**	(**pel-indent-lines** &optional N)	Indent current or marked lines by N indentation levels

	• 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	• **<backtab>** • **<f6> <backtab>** • **<f11> <tab> C**	(**pel-unindent-lines** &optional N)	• Un-indent current line or marked lines by N indentation levels.

	• 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.
	  • 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.		

Indent-tools	The **indent-tools** external package provides several commands to indent, un-indent and navigate across indented text levels. • It provides a minor mode and a key **hydra** that provides all of these commands. 📦 The **indent-tools** external package 🔄 PEL activates it when the **pel-use-indent-tools** user-option is turned on (set to **t**). • This also automatically activates the **hydra** external package. 🎛 PEL provide a global key binding to its key hydra and provides the ability to activate the proposed key binding globally and for python mode: • **pel-indent-tools-key-bound** : activates the **C-c >** key binding either globally or for python-mode only.		
Open the indent-tools hydra See also: 𝔓𝔏 - **Python**	• **<f11> <tab> <f7>** • **<f7> <tab>** • **C-c >**	(**indent-tools-hydra/body**)	Activate the "indent-tools-hydra" hydra. 🎛 With PEL, this key binding is only available when: • globally, when **pel-indent-tools-key-bound** is set to **globally**, • in python-mode only when **pel-indent-tools-key-bound** is set to **python**. • The actual key is selected by indent-tools **indent-tools-keymap-prefix** user-option, the default is **C-c >**

The heads for the associated hydra are:

- **>:** 'indent-tools-indent',
- **<:** 'indent-tools-demote',
- **E:** 'indent-tools-indent-end-of-defun',
- **c:** 'indent-tools-comment',
- **U:** 'indent-tools-uncomment',
- **P:** 'indent-tools-indent-paragraph',
- **l:** 'indent-tools-indent-end-of-level',
- **K:** 'indent-tools-kill-tree',
- **C:** 'indent-tools-copy-hydra/body',
- **s:** 'indent-tools-select',
- **e:** 'indent-tools-goto-end-of-tree',
- **u:** 'indent-tools-goto-parent',
- **d:** 'indent-tools-goto-child',
- **S:** 'indent-tools-select-end-of-tree',
- **n:** 'indent-tools-goto-next-sibling',
- **p:** 'indent-tools-goto-previous-sibling',
- **i:** 'helm-imenu',
- **j:** 'forward-line',
- **k:** 'previous-line',
- **SPC:** 'indent-tools-indent-space',
- **_:** 'undo-tree-undo',
- **L:** 'recenter-top-bottom',
- **f:** 'yafolding-toggle-element',
- **q:** exit

```
 Indent          | Navigation          | Actions
-----------------+---------------------+-----------
 > indent        | j v                 | K kill
 < de-indent     | k ʌ                 | i imenu
 l end of level  | n next sibling      | C Copy…
 E end of fn     | p previous sibling  | c comment
 P paragraph     | u up parent         | U uncomment (paragraph)
 SPC space       | d down child        | f fold
 _ undo          | e end of tree       | q quit
```

See also: 𝕏 **Hide/Show**

☝️ The **f** key toggles the element folding. Press once to hide the sub-tree, press-again to display it back.

Description	Keystroke	Function	Note

dtrt-indent
 	
★★
 	
Detect indentation scheme of file from its content.
☝Useful when editing foreign files (files written by others) that do not conform to the indentation scheme identified by your customization.
	📦 The **dtrt-indent** external package 🔄 **pel-use-dtrt-indent**
	• A very useful global minor mode. When activated by **pel-use-dtrt-indent**, when you open a file it analyzes the indentation scheme and use of hard tabs inside the file. It compares it to the indentation width imposed by the mode specific indentation settings imposed by the appropriate customization user-options for the mode. If the current file indentation scheme differs from what you request for new files, it adjusts the settings to the current content of the file, adapting to the current style used in the files and prints a message showing what it changed.
	
	☝ This is extremely useful when editing *foreign files*, files written by others, that use an indentation scheme that differs from what you use when you create your own files:
	• All new files you create will use the indentation width and use of hard-tabs you identified in your PEL customizable user-options.
	• When you open a file written by others, files that use a different indentation scheme, dtrt-indent-mode will automatically detect the difference and will adjust the relevant user-options variables to adapt to the style, you won't have to modify your PEL settings for these files.
	  • Normally these *foreign files* should be files located inside repositories of foreign groups.
	  • Note that inside a development project and a development group you should use a consistent style and that style should be identified by your PEL customization for the relevant modes. But at least now you have a way to deal with the other files and that does not require you to continuously changing your PEL customization to do it.
	
	dart-indent also provides command to show how it infers the indentation and use of hard tabs and commands you can use to force a different indentation with or less the indentation width and hard-=tab use again. Use that after you modify the file's content.		

Show indentation style detection data	`<f11> <tab> d ?`	(**dtrt-indent-diagnosis**)	Show guess indentation for the current buffer and output diagnostics.
Guess indentation scheme again	`<f11> <tab> d d`	(**dtrt-indent-try-set-offset**)	Try adjusting the current buffer's indentation offset. • Use this after modifying the file's content, manually adjusting the indentation to the with and style you want.
Force a new indentation width	`<f11> <tab> d D`	(**dtrt-indent-set** INDENT)	Prompt for a new indentation offset (width) and force the indentation offset for the current buffer to INDENT.
indent-bars	• A minor-mode that displays vertical bars over each indentation level. The style of the bars is customizable. 📦 The **indent-bars** external package 🔄 PEL activates it when the **pel-use-indent-bars** user-option is turned on (set to **t**).		
Toggle indent-bars	`<f11> <tab> b b`	(**indent-bars-mode** &optional ARG)	Toggle showing bars that indicate indentation. A minor mode.
Reset indent bar config	`<f11> <tab> b r`	(**indent-bars-reset** &rest R)	Reset indent-bars config.
Reset style of indent bars	`<f11> <tab> b s`	(**indent-bars-reset-styles** &rest R)	Reset all styles' colors and faces. • Useful for calling after theme changes.

Smart-shift
 	
Σ **Customize** with:
`<f11> <tab> s <f2>`
	The **smart-shift** external package simplifies shifting a complete line or region of lines right or left but also up or down.
	• It is implemented as a minor or global minor mode that must be enabled first.
	
	• Automatically activate the smart-shift-mode in specified major mode by customizing the **pel-<mode>-activates-minor-modes** user-options.
	• You can also use the commands manually or through the key bindings provided by PEL to activate the smart-shift-mode in the current buffer or globally for all buffers.
	• PEL controls it through customization user-options:
	  📦 The **smart-shift** external package 🔄 PEL activates it when the **pel-use-smart-shift** user-option is turned on (set to t).
	  🎛 PEL also provides the **pel-smart-shift-keybinding** user-option that allows you to select whether the shift keys used by smart-shift mode is the default provided keys only or whether you also want to activate another set.
	  • The default are always available when smart-shift mode is active: `C-c <right>`, `C-c <left>`, `C-c <up>` and `C-c <down>`.
	  • PEL can also activate **one** of the following extra key binding sets:
	  • Using the control cursor key : `C-c C-<right>`, `C-c C-<left>`, `C-c C-<up>` and `C-c C-<down>`.
	  • Using the `<f9>` key as prefix: `<f9> <right>` , `<f9> <left>` , `<f9> <up>` and `<f9> <down>`		

Toggle smart-shift mode in current buffer	`<f11> <tab> s`	(**smart-shift-mode** &optional ARG)	Activate/de-activate the smart-shift mode in the current buffer. • Activate the line-shift key bindings listed below, in the current buffer. • With PEL, the actual key binding selected for the line shift commands depend on the value of the **pel-smart-shift-keybinding** user-option.
Toggle smart-shift mode globally	`<f11> <tab> S`	(**global-smart-shift-mode** &optional ARG)	• Toggle Smart-Shift mode in all buffers. • With prefix ARG, enable Global Smart-Shift mode if ARG is positive; otherwise, disable it. • Smart-Shift mode is enabled in all buffers where 'smart-shift-mode-on' would do it.
When smart-shift mode is active:	☝ As described above, with PEL only **one** of the extra key bindings provided by PEL can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other key binding description cells in this and other tables, only one of the last 2 key bindings is available in the smart-shift minor mode.		
Shift line or region right	• `C-c <right>` • `C-c C-<right>` • `<f9> <right>`	(**smart-shift-right** &optional ARG)	Shift the line or region to the ARG times to the right.
Shift line or region left	• `C-c <left>` • `C-c C-<left>` • `<f9> <left>`	(**smart-shift-left** &optional ARG)	Shift the line or region to the ARG times to the left.
Shift line or region up	• `C-c <up>` • `C-c C-<up>` • `<f9> <up>`	(**smart-shift-up** &optional ARG)	Shift the line or region to the ARG times to the upwards.
Shift line or region down	• `C-c <down>` • `C-c C-<down>` • `<f9> <down>`	(**smart-shift-down** &optional ARG)	Shift the line or region to the ARG times to the downwards
smart-tabs	📦 The **smart-tabs** external package 🔄 PEL activates it when the **pel-use-smart-tabs** user-option is turned on (set to **t**). • See **Smart-Tabs Emacs Wiki page** for more information.		
Toggle smart-tabs mode	`<f11> <tab> M-s`	(**smart-tabs-mode** &optional ARG)	Toggle smart-tabs minor mode. • Intelligently indent with tabs, align with spaces!

Indentation — References

Title & URL	Description
Understanding GNU Emacs and Tabs	Overview description of how Emacs handle the Tab key, often used for strict indentation in many editors. In Emacs it can do much more.
GNU Emacs Manual - Indentation	
GNU Emacs Manual - Indentation for Programs	
Indentation Basic Concepts Tutorial @ XEmacs	A tutorial on indentation written by KaiGrossjohann

Tabs or space for indentation??

Indentation and hard-tab use Style	Advantages	Disadvantages	Popularity	Support by PEL	Supported by
• Use hard-tabs for indentation. • Uncontrolled use of space and hard-tabs for indentation.	Smaller file size	May render indentation differently on printer and sites such as Gihub. The code may look misaligned when not using the proper rendering width for hard-tab characters.	Often used for C, Emacs Lisp and languages used by users of programming editors that can easily deal with this, such as Emacs.	Yes	• PEL mode-specific customizable user-options that are stored into the Emacs relevant indentation-controlling variables.
• Use hard-tabs for indentation. • Controlled use of hard-tabs for indentation, spaces for alignment.	• Ease of indentation width. • Visual rendering of indentation can be narrowed or widened by simply changing the visual rendering (the width) of a hard-tab character. • Smaller file size.	May render indentation differently on printer and sites such as Gihub.	Used by Go and some project that use C and some other programming languages.	Yes	• PEL mode-specific customizable user-options that are stored into the Emacs relevant indentation-controlling variables. • **smart-tabs** • Adjust use of hard-tabs and indentation width settings used by foreign files when dtrt-indent is used (via **pel-use-dtrr-indent**).
Use spaces only. No hard tabs.	The rendering is the same everywhere; on all printers and even on Gihub which botches support for hard tabs.	• Cannot adjust visual width rendering of indentation, a useful feature for visually challenged people: the indentation with is directly tied to the file content. • Larger file size.	Widely popular strategy for most programming languages as of 2025, despite the growing concern for the use of small indentation width (2 mostly) that is difficult to see for some people.	Yes	• PEL mode-specific customizable user-options that are stored into the Emacs relevant indentation-controlling variables. • Adjust use of hard-tabs and indentation width settings used by foreign files when dtrt-indent is used (via **pel-use-dtrr-indent**).

Smarttabs @ GitHub	Starttabs source code repository.

Indentation Styles for Curly Bracket Languages

Indentation Styles @ Wikipedia	
StackOverflow - Emacs BSD/Allman Style with 4 Space Tabs?	
GNU Emacs Manual - Styles	
Emacs BSD/Allman Style with 4 Space Tabs?	
Emacs: Linux Kernel Style but with Allman/BSD Style Braces?	
Emacs Wiki - Indenting C	
Indent preprocessor directives as C code in emacs	Does not fully address the way I want to have multi-indentations for pre-processor
elisp code - ppindent.el	Implements pre-processor indentation with the # always in the first column. Not yet exactly what I want.
Demystify C++ Metaprograms using Emacs	
Programming in C++, Rules and Recommendations	ellemtel style