Indenting & Tab

Description	Keystroke	Function	Note
Indentation under Emacs	Emacs controls indentation according to various rules controlled by the buffer **major mode**. • Furthermore the behaviour of the tab key is also controlled by the major mode; it may have surprising behaviour for people learning Emacs. • The standard behaviour may be modified by the use of major and **minor modes**. • Several major modes implement special indentation schemes, such as Lisp where indentation is inferred by the code itself as opposed to Python that uses indentation for defining scopes. • Several major modes identify a variable that sets the indentation level. Refer to the information on the programming language major mode. • Some programming languages (**such as Go**) impose hard-tab for indentation, using tab for indentation and space for alignment (works very nicely). • Most languages never identified any rule, which led in some case to all sorts of conventions: use of both tabs and spaces, spaces only, with various number of positions for the indentation level. • Emacs can support anything. It can tabify or untabify source code. Impose the use of hard tab or prevent it. • Emacs controls the *display rendering* of hard tabs by the **tab-width** variable. • The go-mode, for example, will move the first non-whitespace character location inside the buffer as you modify the tab-width as indentation is entirely controlled by hard tabs. It does not change the content of the file, just the way the file looks on the screen. • The indentation width is often independent from the tab width but not always. Again it depends on the major mode used. PEL supports various indentation mechanisms and also provides some of its own extensions. It also provides easy access to external packages that implement other behaviours, supporting various major modes. This includes the following: 📦 The **indent-tools** external package ☑ PEL activates it when the **pel-use-indent-tools** user-option is turned on (set to **t**). 📦 The **smart-shift** external package ☑ PEL activates it when the **pel-use-smart-shift** user-option is turned on (set to **t**). Information related to indentation is described in the pages related to programming major modes. The information in this page is generic and complements the mode specific information.		
Open this PDF file. See also: **∑ Help/Info**	`<f11> <tab> <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **∑ 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.
∑ 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.
∑ Customize Emacs indentation control	`<f11> <tab> <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs indentation control groups: indent, indent-tools, smart-shift. • If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.
Set visual rendering of hard tabs for the current buffer	`<f11> M-t`	(**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. • The change is temporary and affects the current buffer only. • 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-tag-width**' user-option variable instead. See the documentation of each major mode for more information.
Insert Literal Tab	**`C-q <tab>`**	(**quoted-insert** ARG) <tab>	Inserts a hard tab inside the file but moves the cursor to the column that represents the next multiple of **tab-width**.
		☝ With PEL, for several major modes, the value of the tab-width variable is controlled by a mode specific user options variable, like pel-c-tab-width for buffers in c-mode. In those buffers the value of tab-width will be set by PEL to the mode specific value when the buffer is opened.	
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. By default, in text modes, tabs are set to 8 spaces, inserting hard tabs. 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.		
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
	🚧The above values that I got in Emacs via inspection do not explain tab behaviour in the Emacs Lisp mode (which is to indent the code according to Emacs Lisp semantics, a **very** useful feature when writing Lisp code). In this mode tab corrects the indentation of the code at the current line, which may be to indent, de-indent or do nothing.		
	`<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.
Insert spaces or tabs to next defined tab-stop column See also: • **𝔓𝔏 - C** , **𝔓𝔏 - C++** • **𝔓𝔏 - D**	**`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, for several major modes, the value of the tab-width variable is controlled by a mode specific user options variable, like pel-c-tab-width for buffers in c-mode. In those buffers the value of tab-width will be by PEL to the mode specific value when the buffer is opened.
Insert an indented line below current line See also: **∑ Align**	• `M-RET` • `<f11> <tab> RET`	(**pel-newline-and-indent-below**)	Insert an indented line just below current line. • The command can also align text vertically if this special mode was activated for the buffer with the `<f11> M-RET` . • To see the current behaviour use `<f11> t a ?` : it displays whether the `M-RET` command aligns text or not.

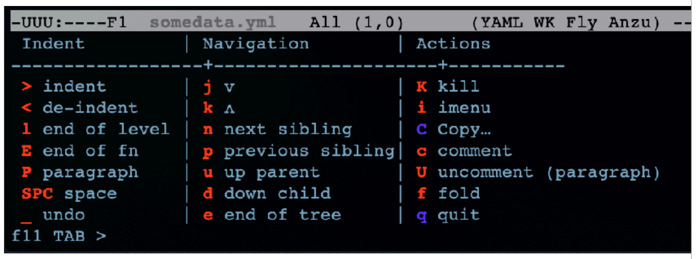
Description	Keystroke	Function	Note
Toggle text alignment on pel-newline-and-indent-below See also: ∑ Align	`<f11> M-RET`	(pel-toggle-newline-indent-align)	Toggle variable *pel-newline-does-align* for the local buffer.
	This toggles the way function ‘pel-newline-and-indent-below’ operates. • If *pel-newline-does-align* is t, it aligns several syntactic element in the current block: the comments, the assignments. • 🎛 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 like c-mode, c++-mode and d-mode). • pel-newline-and-indent-below (assigned the `M-RET`)		
Show state of pel-newline-and-indent-below See also: ∑ Align	`<f11> t a ?`	(pel-show-if-newline-aligns)	Display the behaviour of M-RET in the current buffer: show if that command aligns text or not. Print the information in the echo area.
Change the tab stops	`M-x 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. Again, the tab stop take effect at the top of the buffer,
Change the tab width	`M-: (setq tab-width N)`	The variable *tab-width* normally defaults to 8 in emacs. It can be set locally inside a buffer with (setq tab-width N) or globally with (setq-default tab-width N). The M-: keystroke allows evaluating a lisp expression interactively (as the above two). • Note that any literal tab in the buffer impact the location of the column where the next character shows. When changing the tab-width, the layout shown in the window that contains literal tabs will be modified according to the new tab-width value. • ☝ With PEL, remember that the local value of tab-width is controlled by PEL and set to the value of a mode specific user option, such as pel-c-tab-width in c-mode buffers.	
Make <tab> insert space/tab	`M-: (setq indent-tabs-mode nil/t)`		By default, pressing <tab> insert literal (hard) tabs inside the file. The indent-tabs-mode variable controls that: set to t it inserts tabs, set to nil it inserts spaces.
Next line, indented	`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.
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’.
Move to fist nonbank character on the line	`M-m`	(**back-to-indentation**)	Move point to the first non-whitespace character on this line.
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.
Delete Indentation, join this line to the previous one See also: • ∑ Cut & Paste • ∑ Whitespace	`M-^`	(**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.
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.		

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: ∑ Key-Chords See also: • 𝔓𝔩 - C • 𝔓𝔩 - C++ • 𝔓𝔩 - D • Ϻ reStructuredText	• `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)	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. 📦 With PEL, the `<tab>q` key-chord is also available when pel-use-key-chord is non-nil. See ∑ Key-Chords. ☝ Command numeric prefix **is available** with the key-chord binding. ✂ PEL rebinds this key, but it extends the functionality: pel-indent-rigidly uses indent-rigidly, described below the dashed line. — — — — — — — — — — — — — — — — — — 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-<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>`).		

Description	Keystroke	Function	Note
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.		
Controlling use of hard tabs or spaces for indentation	The use of hard tabs or spaces for indentation is controlled by the Emacs (customizable) variable **indent-tabs-mode**. Like several Emacs variable this variable has global impact, but this can be overridden by directory local value, file local value and buffer local value allowing fine control over set of files and buffers. PEL provides the following related commands. See also: **∑ Whitespace**		
Toggle use of hard tabs and only spaces for indentation in the current buffer	`<f11> t w I`	(**pel-toggle-indent-tabs-mode** &optional ARG)	Toggle use of hard tabs or spaces for indentation in current buffer. • Beep on each change to warn user of the change and display new value. • If ARG is positive set to use hard tabs, otherwise force use of spaces only.
Replacing Tabs with spaces or spaces with tabs	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: **∑ 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: **∑ 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.
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> >`
	
	`C-c >`
		(**indent-tools-hydra/body**)	Activate the e body in 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 >**

See also: **∑ Hide/Show**	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		

```
-UUU:----F1   somedata.yml    All (1,0)       (YAML WK Fly Anzu) --
 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
f11 TAB >
```

Description	Keystroke	Function	Note
Smart-shift	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. You can identify the smart-shift-mode inside one of the pel-<mode>-activates-minor-modes user-options to activate it automatically. 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 additional alternative key bindings for the smart-shift commands that shift line(s). By default the key bindings are using **`C-c`** as a key prefix. With PEL you can also use a control key for the cursor or change the prefix key to use the **`<f9>`** key. The 3 possible key bindings are shown below but only one of them will be available at any given time. The one available is the one selected by the user-option value.		
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.
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. ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.
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. ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.
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. ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.
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 ☝ With PEL **one** of the extra key bindings can be enabled via the **pel-smart-shift-keybinding** user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.

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??

There are several views on the use of hard-tab and space characters for indenting source code. They are:

1. Use only hard-tab for indentation. Uncontrolled use of tabs or spaces for alignment.
2. Use only space characters for indentation. Popular in C like languages. Also popular in Python.
3. Use hard tabs for indentation, and space character for alignment.

- Method 1 was popular originally since it reduces file size when hard tab size was always the same. But soon it became possible to identify a different number of character positions to render a hard tab. And then it became impossible to guarantee the rendering of code indentation and alignment when the number of hard-tabs did not match the indentation level of a line of source code.
- A reaction to this problem is to use Method 2 where hard-tabs are banned. The rendering is therefore always the same no matter what the *size* of a hard tab is since you don't use any. This however increases the size of files. Not a problem for storage today you'd say, but perhaps a problem for data transfer and/or power consumption.
- Method 3 is used by some programming environments. The Go programming language imposes the use of hard-tabs for indentation. And if you want to align text at the right of the indentation level, you use spaces.
  - To use this method in other programming languages, you can use the smart-tabs-mode explained in the **Smart-Tabs Emacs Wiki page**.

Emacs support all modes. It has 2 different buffer local variables that are important and control the rendering of hard-tabs and the indentation:
- **tab-width**: How many columns a hard-tab occupies, the distance between tab-stops.
- indentation offset variable: a variable for each major mode, like **c-basic-offset** for CC modes (C, C++, Java, etc…), that identifies the number of columns per indentation level.

PEL does not yet integrate the smarttabs package. 🚧 For CC modes it provides PEL user-options that control the indentation using method 2.
Using method 3 requires a better understanding from all developers working on the source code with all their editors being able to handle the mix of hard tab and space characters correctly.

Title & URL	Description
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