








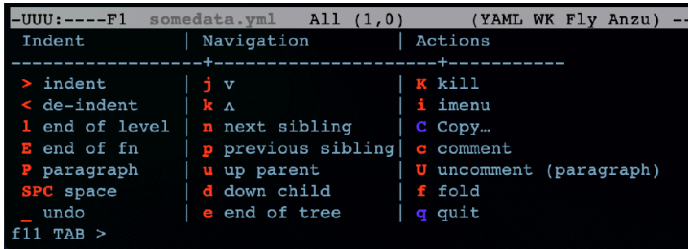





# Indenting & Tab

Description	Keystroke	Function	Note
<p><b>Indentation under Emacs</b></p> <ul style="list-style-type: none"> <li>Indentation with PEL</li> <li>Behaviour of tab key</li> <li>Text alignment</li> <li>Indent region</li> <li>Indenting /un-indenting rigidly</li> <li>Controlling use of tabs and spaces for indentation</li> <li>Replacing tab with spaces or vice-versa</li> <li>indent-tools</li> <li>smart-shift</li> </ul>	<p>Emacs controls indentation according to various rules controlled by the buffer <b>major mode</b>.</p> <ul style="list-style-type: none"> <li>Furthermore the behaviour of the tab key is also controlled by the major mode; it may have surprising behaviour for people learning Emacs.</li> <li>The standard behaviour may be modified by the use of major and <b>minor modes</b>. <ul style="list-style-type: none"> <li>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.</li> <li>Several major modes identify a variable that sets the indentation level. Refer to the information on the programming language major mode.</li> <li>Some programming languages (<b>such as Go</b>) impose hard-tab for indentation, using tab for indentation and space for alignment (works very nicely).</li> <li>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.</li> <li>Emacs can support anything. It can tabify or untabify source code. Impose the use of hard tab or prevent it.</li> <li>Emacs controls the <i>display rendering</i> of hard tabs by the <b>tab-width</b> variable. <ul style="list-style-type: none"> <li>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.</li> <li>The indentation width is often independent from the tab width but not always. Again it depends on the major mode used.</li> </ul> </li> </ul> </li> </ul> <p>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:</p> <ul style="list-style-type: none"> <li> The <b>indent-tools</b> external package  PEL activates it when the <b>pel-use-indent-tools</b> user-option is turned on (set to t).</li> <li> The <b>smart-shift</b> external package  PEL activates it when the <b>pel-use-smart-shift</b> user-option is turned on (set to t).</li> </ul> <p>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.</p>		
<p><b>Open this PDF file.</b> See also: <a href="#">↶ Help/Info</a></p>	<b>&lt;f11&gt; &lt;tab&gt; &lt;f1&gt;</b>	<b>(pel-help-pdf &amp;optional OPEN-WEB-PAGE)</b>	Open the <a href="#">↶ Indentation</a> local PDF. If the prefix argument (like <b>C-u</b> or <b>M--</b> ) is used, then it opens the remote GitHub hosted raw PDF instead. If the <b>pel-flip-help-pdf-arg</b> user-option is set it's the other way around.
<a href="#">↶</a> <b>Customize PEL highlighting control</b>	<b>&lt;f11&gt; &lt;tab&gt; &lt;f2&gt;</b>	<b>(pel-customize-pel &amp;optional OTHER-WINDOW)</b>	Customize PEL support for indentation management <ul style="list-style-type: none"> <li>If OTHER-WINDOW is non-nil (use <b>C-u</b>), display in other window.</li> </ul>
<a href="#">↶</a> <b>Customize Emacs indentation control</b>	<b>&lt;f11&gt; &lt;tab&gt; &lt;f3&gt;</b>	<b>(pel-customize-library &amp;optional OTHER-WINDOW)</b>	Customize Emacs indentation control groups: indent, <a href="#">indent-tools</a> , <a href="#">smart-shift</a> . <ul style="list-style-type: none"> <li>If OTHER-WINDOW is non-nil (use <b>C-u</b>), display in another window.</li> </ul>
 <b>Indentation with PEL</b>	For several major modes, the value of the <b>tab-width</b> variable is controlled by a mode specific user options variable, like <b>pel-c-tab-width</b> 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. See with <b>&lt;f11&gt; &lt;tab&gt; ?</b>		
<b>Show Indentation settings</b>	<b>&lt;f11&gt; &lt;tab&gt; ?</b>	<b>(pel-show-indent &amp;optional APPEND)</b>	Print info about indentation control in a "pel-indent-info" <b>help-mode</b> buffer. <ul style="list-style-type: none"> <li>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.</li> <li>Clear previous buffer content. Use prefix arg (like <b>C-u</b>) to append instead.</li> </ul>
<b>Set visual rendering of hard tabs for the current buffer</b>	<b>&lt;f11&gt; M-t</b>	<b>(pel-set-tab-width N)</b>	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.
	<ul style="list-style-type: none"> <li>This modifies a buffer local value of the the <b>tab-width</b> user-option. The change is temporary and affects the current buffer only.</li> <li>PEL provides a specialized user-option to set the default value of <b>tab-width</b> for several major modes. For example, to change the tab width used for all Go source code files, change the '<b>pel-go-tab-width</b>' user-option variable instead.</li> </ul> <p>See the documentation of each major mode for more information.</p>		
<p><b>Show state of text modes</b></p> <ul style="list-style-type: none"> <li>whether hard tabs are used for indentation, tab-width</li> <li>electric-quote-mode</li> <li>delete-selection-mode</li> <li>enriched-mode</li> <li>overwrite-mode</li> <li>case folding</li> <li>subword, superword, glass modes</li> <li>visible-mode, smart-dash-mode</li> <li>paragraph definition</li> </ul>	<b>&lt;f11&gt; t m ?</b>	<b>(pel-show-text-modes)</b>	Display the state of the various text modes in the mini buffer.
	 This command is specially useful to see if hard-tabs or spaces are inserted for indentation: <ul style="list-style-type: none"> <li>When <b>indent-tabs-mode</b> is active, Emacs inserts a number of hard tabs and spaces. <ul style="list-style-type: none"> <li>The number of hard tabs instead depends on the amount of characters required for indentation and the tab-width.</li> </ul> </li> <li>If indent-tabs-mode is not active, then Emacs inserts only space characters.</li> <li> PEL provides user-options of the form <b>pel-&lt;mode&gt;-use-tabs</b> which is used to initialize the indent-tabs-mode for a buffer in the supported major modes.</li> </ul>		<pre> -UU-:----F1  keymaps.c      Top (1,0)      Git-master (C/*1a WK Anzu- Modes status: - Local indent-tabs-mode      : off: use spaces.  Tab width = 8 - Local electric-quote-mode: off. - delete-selection-mode      : off. - enriched-mode               : not loaded. - overwrite mode              : off. - case-fold-search: on, sort-fold-case: not loaded. - subword mode: off, superword mode: on, glass-mode: not loaded. - visible-mode: off, smart-dash-mode: on. - Sentences end with 2 space characters. - paragraph-start   : "[                         ]*\\(//+\\ \\ *\\ \\ )                         ]*\$\\ \\ ^"L" - paragraph-separate: "[                         ]*\\(//+\\ \\ *\\ \\ )                         ]*\$\\ \\ ^"L" </pre>
<b>Insert Literal Tab</b>	<b>C-q &lt;tab&gt;</b>	<b>(quoted-insert ARG) &lt;tab&gt;</b>	Inserts a <u>hard tab</u> inside buffer. Render text according to value of <b>tab-width</b> .
<b><a href="#">Behaviour of Tab Key</a></b>	<p>In Emacs the behaviour of the &lt;tab&gt; key depends on the major mode of the current buffer. This key is rebound by several major mode.</p> <ul style="list-style-type: none"> <li>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.</li> </ul>		
<p><b>Indent current line (or region)</b></p> <ul style="list-style-type: none"> <li>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 ‘<b>indent-line-function</b>’.</li> <li>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.</li> <li>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.</li> <li>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.</li> </ul> <p>  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.</p>	<b>&lt;tab&gt;</b>	<b>(indent-for-tab-command &amp;optional ARG)</b>	Indent the current line or region, or insert a tab, as appropriate.
	<b>&lt;tab&gt;</b>	<b>indent-for-tab-command &amp;optional ARG)</b>	In <b>Lisp</b> related modes. indent-line-function = indent-relative. <ul style="list-style-type: none"> <li>tab-always-indent = t</li> </ul>
	 Several major modes adjust the behaviour of the tab key to perform semantically aware indentation, such as what is being done in Lisp.		
	<b>&lt;tab&gt;</b>	<b>(c-indent-line-or-region &amp;optional ARG REGION)</b>	In C related modes: Indent active region, current line, or block starting on the line. <ul style="list-style-type: none"> <li>In Transient Mark mode, when the region is active, reindent the region.</li> <li>With prefix argument, rigidly reindent the expression starting on current line.</li> <li>Otherwise reindent just the current line.</li> </ul>
	<b>C-M-q</b>	<ul style="list-style-type: none"> <li>(<b>indent-sexp</b> &amp;optional ENDPOS)</li> <li>(<b>c-indent-exp</b> &amp;optional SHUTUP-P)</li> </ul>	Indent each line of the list starting just after point. <ul style="list-style-type: none"> <li>The command used depends on the major mode of the current buffer.</li> </ul>
<p><b>Insert spaces or tabs to next defined tab-stop column</b> <a href="#">↶</a> - C , <a href="#">↶</a> - C++, <a href="#">↶</a> - D</p>	<b>M-i</b>	<b>(tab-to-tab-stop)</b>	Insert spaces or tabs to next defined tab-stop column.
	<ul style="list-style-type: none"> <li>The exact location of the next tab stop is identified by the value of the <b>tab-stop-list</b> and <b>tab-width</b> for the current buffer.</li> </ul> <p>With PEL, for several major modes, the value of the tab-width variable is controlled by a mode specific user options variable, like <b>pel-c-tab-width</b> for buffers in c-mode. In those buffers the value of tab-width is set by PEL to the mode specific value when opening the buffer. See with <b>&lt;f11&gt; &lt;tab&gt; ?</b></p>		
<p><b>Insert an indented line below current line</b></p> <p>See also: <a href="#">↶ Align</a></p>	<ul style="list-style-type: none"> <li><b>M-RET</b></li> <li><b>&lt;f11&gt; &lt;tab&gt; RET</b></li> </ul>	<b>(pel-newline-and-indent-below)</b>	Insert an indented line just below current line. <ul style="list-style-type: none"> <li>The command can also align text vertically if this special mode was activated for the buffer with the <b>&lt;f11&gt; M-RET</b>.</li> <li>To see the current behaviour use <b>&lt;f11&gt; t a ?</b> : it displays whether the <b>M-RET</b> command aligns text or not.</li> </ul>

[illegible]

Description	Keystroke	Function	Note
Indent line(s) rigidly	<ul style="list-style-type: none"><li>• <b>&lt;f6&gt; &lt;tab&gt;</b></li><li>• <b>&lt;f11&gt; &lt;tab&gt; c</b></li></ul>	( <b>pel-indent-lines</b> &optional N)	Indent current or marked lines by N indentation levels
	<ul style="list-style-type: none"><li>• Works with point anywhere on the line.</li><li>• All lines touched by the region are indented.</li><li>• A special argument N can specify more than one indentation level. It defaults to 1.</li><li>• If a negative number is specified, 'pel-unindent-lines' is used.</li><li>• 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.</li><li>• Use <b>C–g</b> to de-activate the region.</li><li>• Handles presence of hard tabs:<ul style="list-style-type: none"><li>• If indent-tabs-mode is non-nil the indentation is created with a mix of hard-tabs and space characters.</li><li>• 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.</li></ul></li></ul>		
Un-indent line(s) rigidly	<ul style="list-style-type: none"><li>• <b>&lt;backtab&gt;</b></li><li>• <b>&lt;f6&gt; &lt;backtab&gt;</b></li><li>• <b>&lt;f11&gt; &lt;tab&gt; C</b></li></ul>	( <b>pel-unindent-lines</b> &optional N)	<ul style="list-style-type: none"><li>• Un-indent current line or marked lines by N indentation levels.</li></ul>
	<ul style="list-style-type: none"><li>• Works with point is anywhere on the line.</li><li>• All lines touched by the region are un-indented.</li><li>• If region was marked, the function does not deactivate it to allow repeated execution of the command.</li><li>• 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</li><li>• Use <b>C–g</b> to de-activate the region.</li><li>• Handles presence of hard tabs:<ul style="list-style-type: none"><li>• If indent-tabs-mode is non-nil the indentation is created with a mix of hard-tabs and space characters.</li><li>• 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.</li></ul></li></ul>		
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 <b>indent-tabs-mode</b> . Like several Emacs variable this variable has <a href="#">global impact</a> , but this can be overridden by <a href="#">directory local</a> value, <a href="#">file local</a> value and <a href="#">buffer local</a> value allowing fine control over set of files and buffers. PEL provides the following related commands. See also: <a href="#">⌘ Whitespace</a>		
Toggle use of hard tabs and only spaces for indentation in the current buffer	<b>&lt;f11&gt; t w I</b>	( <b>pel-toggle-indent-tabs-mode</b> &optional ARG)	Toggle use of hard tabs or spaces for indentation in current buffer. <ul style="list-style-type: none"><li>• Beep on each change to warn user of the change and display new value.</li><li>• If ARG is positive set to use hard tabs, otherwise force use of spaces only.</li></ul>
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	<b>&lt;f11&gt; t w SPC</b>	( <b>untabify</b> START END &optional ARG)	Convert all tabs in region to multiple spaces, preserving columns. <ul style="list-style-type: none"><li>• If called interactively with prefix ARG, convert for the entire buffer.</li><li>• First select a region (Use C-x h for selecting the whole file). Then use the <i>untabify</i> function to replace all tabs by spaces in that region.</li></ul>
See also: <a href="#">⌘ Whitespace</a>			
Replace multiple spaces with tabs in a region	<b>&lt;f11&gt; t w &lt;tab&gt;</b>	( <b>tabify</b> START END &optional ARG)	Convert multiple spaces in region to tabs when possible. <ul style="list-style-type: none"><li>• A group of spaces is partially replaced by tabs when this can be done without changing the column they end at.</li><li>• If called interactively with prefix ARG, convert for the entire buffer.</li></ul>
See also: <a href="#">⌘ Whitespace</a>			
Indent-tools	<p>The <a href="#">indent-tools</a> external package provides several commands to indent, un-indent and navigate across indented text levels.</p> <ul style="list-style-type: none"><li>• It provides a minor mode and a key <a href="#">hydra</a> that provides all of these commands.</li></ul> <p> The <a href="#">indent-tools</a> external package  PEL activates it when the <a href="#">pel-use-indent-tools</a> user-option is turned on (set to <b>t</b>).</p> <ul style="list-style-type: none"><li>• This also automatically activates the <a href="#">hydra</a> external package.</li></ul> <p> PEL provide a global key binding to its key <a href="#">hydra</a> and provides the ability to activate the proposed key binding globally and for python mode:</p> <ul style="list-style-type: none"><li>• <b>pel-indent-tools-key-bound</b> : activates the <b>C–c &gt;</b> key binding either globally or for python-mode only.</li></ul>		
Open the indent-tools hydra	<b>&lt;f11&gt; &lt;tab&gt; &lt;f7&gt;</b>	(indent-tools-hydra/body)	Activate the "indent-tools-hydra" hydra.
	<b>C–c &gt;</b>		<p> With PEL, this key binding is only available when:</p> <ul style="list-style-type: none"><li>• globally, when <b>pel-indent-tools-key-bound</b> is set to <b>globally</b>,</li><li>• in python-mode only when <b>pel-indent-tools-key-bound</b> is set to <b>python</b>.</li><li>• The actual key is selected by indent-tools <b>indent-tools-keymap-prefix</b> user-option, the default is <b>C–c &gt;</b></li></ul>
See also: <a href="#">⌘I - Python</a>			
See also: <a href="#">⌘ Hide/Show</a>	The heads for the associated hydra are:		
	<pre>&gt;: 'indent-tools-indent', &lt;: '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</pre>		
			

Description	Keystroke	Function	Note
<a href="#">Smart-shift</a>	<p>The <a href="#">smart-shift</a> external package simplifies shifting a complete line or region of lines right or left but also up or down.</p> <ul style="list-style-type: none"> <li>It is implemented as a minor or global minor mode that must be enabled first.</li> <li>You can identify the smart-shift-mode inside one of the <b>pel-&lt;mode&gt;-activates-minor-modes</b> 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.</li> <li>PEL controls it through customization user-options: <ul style="list-style-type: none"> <li> The <a href="#">smart-shift</a> external package</li> <li> PEL activates it when the <b>pel-use-smart-shift</b> user-option is turned on (set to t).</li> </ul> </li> <li> PEL also provides the <b>pel-smart-shift-keybinding</b> 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 <b>C-c</b> as a key prefix. With PEL you can also use a control key for the cursor or change the prefix key to use the <b>&lt;f9&gt;</b> 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.</li> </ul>		
Toggle smart-shift mode in current buffer	<b>&lt;f11&gt; &lt;tab&gt; s</b>	( <b>smart-shift-mode</b> &optional ARG)	Activate/de-activate the smart-shift mode in the current buffer. <ul style="list-style-type: none"> <li>Activate the line-shift key bindings listed below, in the current buffer. <ul style="list-style-type: none"> <li>With PEL, the actual key binding selected for the line shift commands depend on the value of the <b>pel-smart-shift-keybinding</b> user-option.</li> </ul> </li> </ul>
Toggle smart-shift mode globally	<b>&lt;f11&gt; &lt;tab&gt; S</b>	( <b>global-smart-shift-mode</b> &optional ARG)	<ul style="list-style-type: none"> <li>Toggle Smart-Shift mode in all buffers.</li> <li>With prefix ARG, enable Global Smart-Shift mode if ARG is positive; otherwise, disable it.</li> <li>Smart-Shift mode is enabled in all buffers where ‘smart-shift-mode-on’ would do it.</li> </ul>
Shift line or region right	<ul style="list-style-type: none"> <li><b>C-c &lt;right&gt;</b></li> <li><b>C-c &lt;C-right&gt;</b></li> <li><b>&lt;f9&gt; &lt;right&gt;</b></li> </ul>	( <b>smart-shift-right</b> &optional ARG)	Shift the line or region to the ARG times to the right. <p>👉 With PEL <b>one</b> of the extra key bindings can be enabled via the <b>pel-smart-shift-keybinding</b> user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.</p>
Shift line or region left	<ul style="list-style-type: none"> <li><b>C-c &lt;left&gt;</b></li> <li><b>C-c &lt;C-left&gt;</b></li> <li><b>&lt;f9&gt; &lt;left&gt;</b></li> </ul>	( <b>smart-shift-left</b> &optional ARG)	Shift the line or region to the ARG times to the left. <p>👉 With PEL <b>one</b> of the extra key bindings can be enabled via the <b>pel-smart-shift-keybinding</b> user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.</p>
Shift line or region up	<ul style="list-style-type: none"> <li><b>C-c &lt;up&gt;</b></li> <li><b>C-c &lt;C-up&gt;</b></li> <li><b>&lt;f9&gt; &lt;up&gt;</b></li> </ul>	( <b>smart-shift-up</b> &optional ARG)	Shift the line or region to the ARG times to the upwards. <p>👉 With PEL <b>one</b> of the extra key bindings can be enabled via the <b>pel-smart-shift-keybinding</b> user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.</p>
Shift line or region down	<ul style="list-style-type: none"> <li><b>C-c &lt;down&gt;</b></li> <li><b>C-c &lt;C-down&gt;</b></li> <li><b>&lt;f9&gt; &lt;down&gt;</b></li> </ul>	( <b>smart-shift-down</b> &optional ARG)	Shift the line or region to the ARG times to the downwards <p>👉 With PEL <b>one</b> of the extra key bindings can be enabled via the <b>pel-smart-shift-keybinding</b> user-option. So unlike other cells only one of the last 2 key bindings is available in the smart-shift minor mode.</p>

## Indentation — References

Title & URL	Description
<a href="#">Understanding GNU Emacs and Tabs</a>	Overview description of how Emacs handle the Tab key, often used for strict indentation in many editors. In Emacs it can do much more.
<a href="#">GNU Emacs Manual - Indentation</a>	
<a href="#">GNU Emacs Manual - Indentation for Programs</a>	
<a href="#">Indentation Basic Concepts Tutorial @ XEmacs</a>	A tutorial on indentation written by <a href="#">KaiGrossjohann</a>
<b>Tabs or space for indentation??</b> There are several views on the use of hard-tab and space characters for indenting source code. They are: <ol style="list-style-type: none"> <li>Use only hard-tab for indentation. Uncontrolled use of tabs or spaces for alignment.</li> <li>Use only space characters for indentation. Popular in C like languages. Also popular in Python.</li> <li>Use hard tabs for indentation, and space character for alignment.</li> </ol> <ul style="list-style-type: none"> <li>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.</li> <li>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 <i>size</i> 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.</li> <li>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. <ul style="list-style-type: none"> <li>To use this method in other programming languages, you can use the smart-tabs-mode explained in the <a href="#">Smart-Tabs Emacs Wiki page</a>.</li> </ul> </li> </ul> <p>Emacs support all modes. It has 2 different buffer local variables that are important and control the rendering of hard-tabs and the indentation:</p> <ul style="list-style-type: none"> <li><b>tab-width:</b> How many columns a hard-tab occupies, the distance between tab-stops.</li> <li>indentation offset variable: a variable for each major mode, like <b>c-basic-offset</b> for CC modes (C, C++, Java, etc...), that identifies the number of columns per indentation level.</li> </ul> <p>PEL does not yet integrate the smarttabs package. 🚧 For CC modes it provides PEL user-options that control the indentation using method 2.</p> <p>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.</p>	
<a href="#">Smarttabs @ GitHub</a>	Starttabs source code repository.
<a href="#">Indentation Styles for Curly Bracket Languages</a>	
<a href="#">Indentation Styles @ Wikipedia</a>	
<a href="#">StackOverflow - Emacs BSD/Allman Style with 4 Space Tabs?</a>	
<a href="#">GNU Emacs Manual - Styles</a>	
<a href="#">Emacs BSD/Allman Style with 4 Space Tabs?</a>	
<a href="#">Emacs: Linux Kernel Style but with Allman/BSD Style Braces?</a>	
<a href="#">Emacs Wiki - Indenting C</a>	
<a href="#">Indent preprocessor directives as C code in emacs</a>	Does not fully address the way I want to have multi-indentations for pre-processor
<a href="#">elisp code - ppindent.el</a>	Implements pre-processor indentation with the # always in the first column. Not yet exactly what I want.
<a href="#">Demystify C++ Metaprograms using Emacs</a>	
<a href="#">Programming in C++, Rules and Recommendations</a>	ellemtel style