




## Indenting & Tab

Description	Keystroke	Function	Note
<a href="#">Indentation under Emacs</a>	Emacs controls indentation via major modes and the tab key may take different and surprising behaviour for people just starting with Emacs. More information is available in the tables of the various major modes.		
<b>Open this PDF file.</b> See also: <a href="#">🔗 Help/Info</a>	<f11> <tab> <f1>	<a href="#">(pel-help-pdf &amp;optional OPEN-WEB-PAGE)</a>	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="#">🔗 Customize Emacs indentation control</a>	<f11> <tab> <f3>	<a href="#">(pel-customize-library &amp;optional OTHER-WINDOW)</a>	Customize Emacs indentation. Opens the indent customization group. <ul style="list-style-type: none"> <li>If OTHER-WINDOW is non-nil (use <b>C-u</b>), display in another window.</li> </ul>
<a href="#">Insert Literal Tab</a>	<b>C-q</b> <tab>	<b>(quoted-insert ARG)</b> <tab>	Inserts a hard tab inside the file but moves the cursor to the column that represents the next multiple of <b>tab-width</b> .
		👉 <b>With PEL</b> , 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.	
<a href="#">Behaviour of Tab</a>	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.		
<b>Indent current line (or region)</b>	<tab>	<b>(indent-for-tab-command &amp;optional ARG)</b>	Indent the current line or region, or insert a tab, as appropriate.
		<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>	
		⚠️ 🛠️ 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.	
		<b>indent-for-tab-command</b> &optional ARG)	In <b>Lisp</b> related modes. <ul style="list-style-type: none"> <li>indent-line-function = indent-relative.</li> <li>tab-always-indent = t</li> </ul>
		🛠️👉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 <b>very</b> 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.	
		<b>(c-indent-line-or-region</b> &optional ARG REGION)	In C related modes: Indent active region, current line, or block starting on this line. <ul style="list-style-type: none"> <li>In Transient Mark mode, when the region is active, reindent the region.</li> <li>Otherwise, with a prefix argument, rigidly reindent the expression starting on the current line.</li> <li>Otherwise reindent just the current line.</li> </ul>
<b>Indent lines of list after point</b>  <b>(CLBC s3.lisp)</b>	<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>
<b>Insert spaces or tabs to next defined tab-stop column</b> See also: <ul style="list-style-type: none"> <li><a href="#">🔗 I - C</a></li> <li><a href="#">🔗 I - C++</a></li> <li><a href="#">🔗 I - D</a></li> </ul>	<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> <li>👉 <b>With PEL</b>, 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.</li> </ul>
<b>Insert an indented line below current line</b>  See also: <a href="#">🔗 Align</a>	<ul style="list-style-type: none"> <li><b>M-RET</b></li> <li>&lt;f11&gt; &lt;tab&gt; <b>RET</b></li> </ul>	<a href="#">(pel-newline-and-indent-below)</a>	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 &lt;f11&gt; <b>M-RET</b>.</li> <li>To see the current behaviour use &lt;f11&gt; <b>t a ?</b> : it displays whether the <b>M-RET</b> command aligns text or not.</li> </ul>
<b>Toggle text alignment on pel-newline-and-indent-below</b> See also: <a href="#">🔗 Align</a>	<f11> <b>M-RET</b>	<a href="#">(pel-toggle-newline-indent-align)</a>	Toggle variable <i>pel-newline-does-align</i> for the local buffer. This toggles the way function 'pel-newline-and-indent-below' operates. <ul style="list-style-type: none"> <li>If <i>pel-newline-does-align</i> is t, it aligns several syntactic element in the current block: the comments, the assignments.</li> <li>👁️ Identify modes where <i>pel-newline-does-align</i> is automatically activated (set to t) by adding the major mode to the list in the <b>pel-modes-activating-align-on-return</b> user option.</li> <li>This affects the behaviour of the following commands: <ul style="list-style-type: none"> <li>pel-cc-newline (assigned to <b>RET</b> in CC modes like c-mode, c++-mode and d-mode).</li> <li>pel-newline-and-indent-below (assigned the <b>M-RET</b>)</li> </ul> </li> </ul>
<b>Show state of pel-newline-and-indent-below</b> See also: <a href="#">🔗 Align</a>	<f11> <b>t a ?</b>	<a href="#">(pel-show-if-newline-aligns)</a>	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.
<b>Change the tab stops</b>	<b>M-x</b> <code>edit-tab-stops</code>		Opens a <b>*Tab Stops* buffer</b> . 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,
<b>Change the tab width</b>	<b>M-:</b> <code>(setq tab-width N)</code>	The variable <i>tab-width</i> 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). <ul style="list-style-type: none"> <li>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.</li> <li>👉 <b>With PEL</b>, 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.</li> </ul>	
<b>Make &lt;tab&gt; insert space/tab</b>	<b>M-:</b> <code>(setq indent-tabs-mode nil/t)</code>		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.

Description	Keystroke	Function	Note
Next line, indented	C–j	(electric-newline-and-maybe-indent)	Add new line and indent next line. Indentation is controlled by the variable <b>left-margin</b> . Pressing <b>Tab</b> 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. <ul style="list-style-type: none"> <li>A numeric prefix argument specifies a column: indent each line to that column.</li> <li>With no prefix argument, the command chooses one of these methods and indents all the lines with it: <ol style="list-style-type: none"> <li>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.</li> <li>If ‘indent-region-function’ is non-nil, call that function to indent the region.</li> <li>Indent each line via ‘indent-according-to-mode’.</li> </ol> </li> </ul>
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: <ul style="list-style-type: none"> <li>⌘ Cut &amp; Paste</li> <li>⌘ Whitespace</li> </ul>	M–^	(delete-indentation &optional ARG)	Join this line to previous and fix up whitespace at join. <ul style="list-style-type: none"> <li>If there is a fill prefix, delete it from the beginning of this line.</li> <li>With argument, join this line to following line.</li> </ul>
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.
	<p>The following line shows the indentation points in this line.</p> <pre> The following line shows the indentation points in this line. </pre> <ul style="list-style-type: none"> <li>If FIRST-ONLY is non-nil (ie. using <b>C–u</b> prefix) then only the first indent point is considered.</li> <li>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.</li> <li>If there isn’t a previous nonblank line and UNINDENTED-OK is nil, call ‘tab-to-tab-stop’.</li> </ul> <p>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.</p>		
Indenting and un-indenting rigidly	<p>The following commands provide non-semantic indentation of the current line or marked region.</p> <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>		
Indent/Unindent rigidly  See also: ⌘ Key-Chords          See also: <ul style="list-style-type: none"> <li>⌘ I - C</li> <li>⌘ I - C++</li> <li>⌘ I - D</li> <li>⌘ reStructuredText</li> </ul>	<ul style="list-style-type: none"> <li>C–x &lt;tab&gt;</li> <li>&lt;f11&gt; &lt;tab&gt; &lt;tab&gt;</li> <li>&lt;tab&gt;q</li> </ul>	(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. <ul style="list-style-type: none"> <li><b>If a region is marked</b>, it uses ‘indent-rigidly’ and provides the same prompts to control indentation changes.</li> <li><b>If no region is marked</b>, it operates on current line(s) identified by the numeric argument N (or if not specified N=1): <ul style="list-style-type: none"> <li>N = [-1, 0, 1] : operate on current line</li> <li>N &gt; 1 : operate on the current line and N-1 lines below.</li> <li>N &lt; -1 : operate on the current line and (abs N) -1 lines above.</li> </ul> </li> </ul>  With PEL, the <b>&lt;tab&gt;q</b> key-chord is also available when pel-use-key-chord is non-nil. See ⌘ Key-Chords.  Command numeric prefix <b>is available</b> 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. <ul style="list-style-type: none"> <li>If called interactively with no prefix argument, activate a transient mode in which the indentation can be adjusted interactively by typing <b>&lt;left&gt;</b>, <b>&lt;right&gt;</b>, <b>&lt;S-left&gt;</b>, or <b>&lt;S-right&gt;</b>.</li> </ul>
	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 <i>indent-rigidly-map</i> with by default provides: <ul style="list-style-type: none"> <li><b>S–&lt;right&gt;</b> indent-rigidly-right-to-tab-stop</li> <li><b>S–&lt;left&gt;</b> indent-rigidly-left-to-tab-stop</li> <li><b>&lt;right&gt;</b> indent-rigidly-right</li> <li><b>&lt;left&gt;</b> indent-rigidly-left</li> </ul> Typing any other key deactivates the transient mode. <ul style="list-style-type: none"> <li>The <b>S–&lt;right&gt;</b> and <b>S–&lt;left&gt;</b> keys indent/de-indent to the next tab-stop position, which is controlled by the <b>tab-width</b> user option. <ul style="list-style-type: none"> <li>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 <b>pel-c-tab-width</b> is automatically stored into tab-width when the buffer is opened.</li> </ul> </li> </ul>  If you use the cua-mode: the cua-mode uses <b>C–x</b> , to invoke this command when cua-mode is active, type it really fast or type <b>C–x C–x &lt;tab&gt;</b> (or use the PEL binding <b>&lt;f11&gt; &lt;tab&gt; &lt;tab&gt;</b> ).		
Indent line(s) rigidly	<ul style="list-style-type: none"> <li>&lt;f6&gt; &lt;tab&gt;</li> <li>&lt;f11&gt; &lt;tab&gt; c</li> </ul>	(pel-indent-lines &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>&lt;backtab&gt;</li> <li>&lt;f6&gt; &lt;backtab&gt;</li> <li>&lt;f11&gt; &lt;tab&gt; C</li> </ul>	(pel-unindent-lines &optional N)	<ul style="list-style-type: none"> <li>Un-indent current line or marked lines by N indentation levels.</li> <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>

Description	Keystroke	Function	Note
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	<f11> t w I	(pel-toggle-indent-tabs-mode &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  See also: <a href="#">§ Whitespace</a>	<f11> t w SPC	(untabify 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>
Replace multiple spaces with tabs in a region  See also: <a href="#">§ Whitespace</a>	<f11> t w <tab>	(tabify 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>

## 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. 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