










Indenting & Tab

Description	Keystroke	Function	Note
Indentation under Emacs <ul style="list-style-type: none"> 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 Smart-shift Smart-tabs <div>Last updated on:</div>	<p>Emacs controls indentation according to various rules controlled by the buffer major mode.</p> <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> 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 <i>display rendering</i> of hard tabs by the tab-width variable. <ul style="list-style-type: none"> 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. <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">  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).  The smart-tabs external package  PEL activates it when the pel-use-smart-tabs user-option is turned on (set to t). <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> <div>2025-10-30</div> <div>👉 See Showing Information About Indentation and Hard Tab Control under PEL</div>		
Open this PDF file. See also: 🔗 Help/Info	<code><f11> <tab> <f1></code>	(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	<code><f11> <tab> <f2></code>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL support for indentation management <ul style="list-style-type: none"> If OTHER-WINDOW is non-nil (use C-u), display in other window.
🔗 Customize Emacs indentation control	<code><f11> <tab> <f3></code>	(pel-customize-library &optional OTHER-WINDOW)	Customize Emacs indentation control groups: indent, indent-tools , smart-shift . <ul style="list-style-type: none"> If OTHER-WINDOW is non-nil (use C-u), display in another window.
Use 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 . <ul style="list-style-type: none"> Like several Emacs variable this variable has global impact, but this can be overridden by directory local value, file local value <div>See also: 🔗 Whitespace</div>		
Toggle use of hard tabs for indentation in the current buffer	<ul style="list-style-type: none"> <code><f11> <tab> m</code> <code><f11> t w I</code> 	(pel-toggle-indent-tabs-mode &optional ARG)	Toggle whether indentation can insert hard tab characters in the current buffer. <ul style="list-style-type: none"> 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. This uses Emacs indent-tabs-mode function and provides more feedback.
Hard-tab “width”	The current-buffer value of tab-width affects the <i>visual rendering</i> of the hard-tab character. It does not affect the content of the buffer or file.		
Set visual rendering of hard tabs for the current buffer <ul style="list-style-type: none"> Does not change buffer/file content 	<code><f11> <tab> w</code>	(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. <ul style="list-style-type: none"> 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-tab-width' user-option variable instead. See the documentation of each major mode for more information.
Tab stops	Emacs keeps track of a set of “ <i>tab-stop</i> ” columns that can be used as reference points to align text. Something similar to typewriter tab-stop .		
Change the tab stops <ul style="list-style-type: none"> used by M-i 	<code><f11> <tab> e</code>	(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. <ul style="list-style-type: none"> The tab stop take effect at the top of the buffer, as used by M-i
Show Indentation settings <div>For several major modes, a PEL mode-specific user-option controls the the value of the tab-width variable for the mode. For example, pel-c-tab-width is used for c-mode buffers.</div> <div>Note however, that indentation for C buffer is controlled by the c-basic-offset user-option.</div> <div>Other major modes may provide mode specific control variables and the printed information will differ.</div>	<ul style="list-style-type: none"> <code><f11> <tab> ?</code> <code><f11> ? <tab></code> 	(pel-show-indent &optional APPEND)	Print info about indentation control in a *pel-indent-info* help-mode buffer. <ul style="list-style-type: none"> 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. <div> <div>-----Indentation Control from alloc.c --- Wednesday, April 30, 2025 @ 08:21:59 -----</div> <div> <div> <div>- pel-c-indent-width : 4</div> <div>- pel-c-tab-width : 4</div> <div>- pel-c-use-tabs : nil</div> </div> <div>-----</div> <div>The above major-mode specific user options take precedence over the following global ones (unless they are set by file variables):</div> <div> <div>- c-basic-offset : 4</div> <div>- tab-width : 4</div> <div>-> Use pel-set-tab-width to change locally and have tabs rendered with a different width.</div> <div>- indent-tabs-mode : nil</div> <div>- standard-indent : 4</div> <div>- tab-always-indent : t</div> <div>- tab-stop-list : nil</div> </div> <div> <div>-UUU:%*- F1 *pel-indent-info* All (1,0) (Help WK Anzu) 08:22 2.34 -----</div> <div>f11 TAB ?</div> </div> </div> <div> <div>These are help buttons! Click on them in graphic mode or text with mouse enabled. In text-mode without mouse support, <tab> to navigate and <ret> to open help.</div> </div> </div>
Toggle text alignment on pel-newline-and-indent-below See also: 🔗 Align	<code><f11> M-RET</code>	(pel-toggle-newline-indent-align)	Toggle variable <i>pel-newline-does-align</i> for the local buffer. <p>It affects the way function 'pel-newline-and-indent-below' operates.</p> <ul style="list-style-type: none"> If <i>pel-newline-does-align</i> is t, it aligns several syntactic element in the current block: the comments, the assignments.  set modes that automatically activates <i>pel-newline-does-align</i> 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) <ul style="list-style-type: none"> See the list with <code><f11> t a ?</code>
Show state of text modes <ul style="list-style-type: none"> 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 <div>Output example ➡</div>	<code><f11> t m ?</code>	(pel-show-text-modes)	Display the state of the various text modes in the mini buffer. <ul style="list-style-type: none">  Quickly show several settings inside the mode line: the tab settings, text alignment on newline, whitespace mode, etc... When indent-tabs-mode is active, Emacs inserts a number of hard tabs and spaces. <ul style="list-style-type: none"> The number of hard tabs instead depends on the amount of characters required for indentation and the tab-width. If indent-tabs-mode is not active, then Emacs inserts only space characters.  PEL provides user-options of the form pel-<mode>-use-tabs which is used to initialize the indent-tabs-mode supported major modes buffers. <div>Text Modes Status:</div> <div> <div>- Local indent-tabs-mode : off: use spaces, Tab width = 8</div> <div>- Local newline does align : off . Automatically activated by modes (<f11> t a <f2>): (c-mode c++-mode sh-mode)</div> <div>- Local electric-quote-mode: off , electric-quote-local-mode: not loaded.</div> <div>- whitespace-mode : not loaded, show-trailing-whitespace : off , indicate-empty-lines: off.</div> <div>- enriched-mode : not loaded.</div> <div>- overwrite-mode : off , delete-selection-mode : off.</div> <div>- case-fold-search : on , sort-fold-case : not loaded.</div> <div>- subword mode : off , superword mode : on , glass-mode: not loaded.</div> <div>- visible-mode : off , smart-dash-mode : not loaded.</div> <div>- Sentences end with 2 space characters.</div> <div>- paragraph-start : "^L\\[]*\$"</div> <div>- paragraph-separate: "[^L]*\$"</div> </div>

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. <ul style="list-style-type: none"> It is implemented as a minor or global minor mode that must be enabled first. 		
ⓘ Customize with: <f11> <tab> s <f2>	<ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> Activate the line-shift key bindings listed below, in the current buffer. <ul style="list-style-type: none"> 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)	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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).		
Toggle smart-tabs mode	<f11> <tab> M-s	(smart-tabs-mode &optional ARG)	Toggle smart-tabs minor mode. <ul style="list-style-type: none"> 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?? 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"> Use only hard-tab for indentation. Uncontrolled use of tabs or spaces for alignment. Use only space characters for indentation. Popular in C like languages. Also popular in Python. Use hard tabs for indentation, and space character for alignment. <ul style="list-style-type: none"> 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 <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. 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"> To use this method in other programming languages, you can use the smart-tabs-mode explained in the Smart-Tabs Emacs Wiki page. <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"> 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. <p>PEL provides access to the smarttabs package but it's not yet fully configured for programming modes nor tested. 🚧 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>	
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