












Indenting & Tab

Description	Keystroke	Function	Note
Indentation under Emacs	<p>Emacs controls indentation via major modes and the tab key may take different and surprising behaviour for people just starting with 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.</p> <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. 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). 		
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 <ul style="list-style-type: none"> 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 . <ul style="list-style-type: none"> If OTHER-WINDOW is non-nil (use C-u), display in another window.
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	<p>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.</p> <p>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.</p>		
Indent current line (or region)	<tab>	(indent-for-tab-command &optional ARG)	Indent the current line or region, or insert a tab, as appropriate.
			<ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> 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 this line. <ul style="list-style-type: none"> In Transient Mark mode, when the region is active, reindent the region. Otherwise, with a prefix argument, rigidly reindent the expression starting on the current line. Otherwise reindent just the current line.
Indent lines of list after point (CLBC s3.lisp)	C-M-q	<ul style="list-style-type: none"> (indent-sexp &optional ENDPOS) (c-indent-exp &optional SHUTUP-P) 	Indent each line of the list starting just after point. <ul style="list-style-type: none"> The command used depends on the major mode of the current buffer.
Insert spaces or tabs to next defined tab-stop column See also: <ul style="list-style-type: none"> 🔗 I - C 🔗 I - C++ 🔗 I - D 	M-i	(tab-to-tab-stop)	Insert spaces or tabs to next defined tab-stop column. <ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> M-RET <f11> <tab> RET 	(pel-newline-and-indent-below)	Insert an indented line just below current line. <ul style="list-style-type: none"> 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.
Toggle text alignment on pel-newline-and-indent-below See also: 🔗 Align	<f11> M-RET	(pel-toggle-newline-indent-align)	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"> If <i>pel-newline-does-align</i> is t, it aligns several syntactic element in the current block: the comments, the assignments.  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 pel-modes-activating-align-on-return user option. This affects the behaviour of the following commands: <ul style="list-style-type: none"> 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 <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"> 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. 	

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 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. <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.
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. <ul style="list-style-type: none"> • 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 <i>untabify</i> 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. <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • 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). <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: 🐍 - 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: <ul style="list-style-type: none"> • 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		
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. 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: <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 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. <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.
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. 🙌 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	<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. 🙌 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	<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. 🙌 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	<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 🙌 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
<p>Tabs or space for indentation??</p> <p>There are several views on the use of hard-tab and space characters for indenting source code. They are:</p> <ol style="list-style-type: none">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. <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 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.<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 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>	
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