## **Inserting Text**

Description	Keystroke	Function	Note		
Inserting Text	The commands described in this table insert specialized text at point (cursor) location.  The first sections of the table show commands that are not template-based.  The bottom section describe the flexible template system supported by PEL: tempo skeletons and yasnippet.				
Open this PDF file. See also: <u>∑ Help/Info</u>	• <f11> i <f1> • <f11> y <f1> • <f11> _ <f1></f1></f11></f1></f11></f1></f11>	(pel-help-pdf &optional OPEN-WEB-PAGE)	Open the <u>Normal Text</u> 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.		
<u>∑ Customize</u> PEL Text Insertions control	<f11> i <f2></f2></f11>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL text insertion support: lice, smart-dash, tempo, time-stamp, yasnippet.  • If OTHER-WINDOW is non-nil (use <b>C-u</b> ), display in other window.		
<u>∑ Customize</u> Emacs Text Insertions control	<f11> i <f3></f3></f11>	(pel-customize-library &optional OTHER-WINDOW)	Customize Emacs text insertion support: lice, smart-dash, tempo, time-stamp, yasnippet		
Insert time & file info	The following commar	nds insert time stamps of speci	ific formats and name of the current file.		
Insert current date	<f11> i d</f11>	(pel-insert-current-date &optional UTC)	Insert current date (only, no time) at point.  • Local by default, UTC if C-u prefix used.		
Insert current date & time	<f11> i D</f11>	(pel-insert-current-date- time &optional UTC)	Insert current date and time at point.  • Local by default, UTC if C-u prefix used.		
Insert current filename	• <f11> i f • <f6> f</f6></f11>	(pel-insert-filename &optional N)	Insert the file name of the currently edited file at point.  • By default: insert filename of current buffer with complete absolute path.  • With a numeric argument you can select the file name of the current buffer or the buffers in the 4 surrounding windows. 8: up, 2: down, 4: left, 6: right. Any other number identifies the current window.  • When the numeric argument is positive the file with complete absolute path is inserted,  • With negative numeric argument the path is omitted.		
Insert time stamp	<f11> i t</f11>	(pel-insert-iso8601- timestamp &optional UTC)	Insert ISO 8601 conforming abbreviated YYYY-MM-DD hh:mm:ss format timestamp.  • Local by default, UTC if C-u prefix is used.		
Insert software license text	• <f11> i L • <f6> L</f6></f11>	(lice NAME)	Insert license and headers at point.  Prompts for license NAME, which is a license template name like "mit", "gpl-3.0", etc The list is available with TAB completion: hit TAB on prompt to get the complete list of templates.  PEL activates it if pel-use-lice user option is t.		
Automatic File Time Stamp on file save  References:  TimeStamps @ EmacsWiki  Change time stamp format in:  markdown file  reStucturedText file  See also: File mngt	Emacs has a built-in_automatic time-stamping of files, it must be activated by adding the time-stamp function to the before-save-hook variable. This can either be done via Emacs customization system or explicitly inside your init file with the following code:     (add-hook 'before-save-hook 'time-stamp)  The time stamp will be added to files that contain, inside their first 8 lines, a line that looks like one of the following:     'Time-stamp: "  You can, however change these defaults and get Emacs to update all sorts of time stamp formats, even inside source code statements:     time-stamp pattern consists of 4 parts, each one controlled by a variables:     time-stamp-pattern consists of 4 parts, each one controlled by a variable:     time-stamp-limit : identifies where in the file the time stamp can be located. Defaults to 8: the first 8 lines.     time-stamp-limit identifies the text pattern that precedes the time stamp.     time-stamp-ormat specifies the format of the time stamp.     time-stamp-pattern consists of a 021 * (3021 * 30				
Update file time stamp See also: ∑ File mngt	<f11> f t</f11>	(time-stamp)	Force update the time stamp string(s) in the current buffer.  • Updates a time stamp of format recognized by <i>Emacs current settings</i> even when automatic time-stamp update is off.  • More information about the " <i>Emacs current settings</i> " in the description block above.		
Toggle time stamp automatic update	<f11> f M-t</f11>	(time-stamp-toggle-active &optional ARG)	Toggle 'time-stamp-active', setting whether <f11> f t updates a buffer.  • With ARG, turn time stamping on if and only if arg is positive.</f11>		
Inserting & Automatically Updating Copyrights	Emacs has built-in support for insertion and update of copyright notices inside files.  • Two commands, shown below, are provided to manually insert or update the file's copyright notice.  • The copyright notice can be automatically updated by adding the copyright-update function to the list of before-save-hook variable with the following code:  (add-hook 'before-save-hook 'copyright-update)  To be automatically updated, the copyright notice must be placed within an area at the beginning of the file specified by the value of the copyright-limit variable, normally defined as the first 2000 characters. This variable is customizable.				
Insert copyright notice at point	<f11> i c</f11>	(copyright &optional STR ARG)	Insert a copyright by \$ORGANIZATION notice at cursor.  If the ORGANIZATION environment variable is not available, Emacs prompts for it.		
See also: <u><b>∑ File mngt</b></u>					
Update file's copyright notice	<f11> i M-c</f11>	(copyright-update &optional ARG INTERACTIVEP)	Update copyright notice to indicate the current year.  • With prefix ARG, replace the years in the notice rather than adding the current year after them. If necessary, and 'copyright-current-gpl-version' is set, any copying permissions following the copyright are updated as well.  ▲ Even when used interactively copyright-update does not warn if there is no copyright in the current buffer to update. It does not create a missing notice.  ☑ If you want automatic copyright notice updates when a modified buffer is saved, set the pel-update-copyright user option to t.  • Without PEL add the following inside your init.el file:  (add-hook 'before-save-hook 'copyright-update)		

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>		
Insert Commented Lines	The following commands help insert commented lines or just underlines the current line of text using the character corresponding to one of the adornment level used for reStructuredText sections. The strings are commented according to the major mode of the current buffer. If the buffer has no identified comment strings, the command prompts for them the first time it is used in that type of buffer.  The following commands are also listed in the Comments table.				
Insert commented line See also: <u>▼ Comments</u>	• <f11> i 1 • <f6> 1</f6></f11>	(pel-insert-line &optional LINELEN)	Insert a (commented) line before/at current line.  If point is at the beginning of the line insert it there.  If point is in the middle of a line, move point at beginning of line before inserting it.  The number of dash characters of the line is specified by LINELEN:  If LINELEN is not specified the buffer's fill-column value is used.  It supports several programming and markup language and uses the comment style identified by the file extension. If the comment style is unknown the command prompts for one.  fill-column is customizable and can be used as a file or directory variable.		
Comment-underline current line with level 1 adornment	<f11> _ 1</f11>	(pel-commented-adorn-1)	Insert a commented level-1 reST line adornment at point.		
Comment-underline current line with level 2 adornment	<f11> _ 2</f11>	(pel-commented-adorn-2)	Insert a commented level-2 reST line adornment at point.		
Comment-underline current line with level 3 adornment	<f11> _ 3</f11>	(pel-commented-adorn-3)	Insert a commented level-3 reST line adornment at point.		
Comment-underline current line with level 4 adornment	<f11> _ 4</f11>	(pel-commented-adorn-4)	Insert a commented level-4 reST line adornment at point.		
Comment-underline current line with level 5 adornment	<f11> _ 5</f11>	(pel-commented-adorn-5)	Insert a commented level-5 reST line adornment at point.		
Comment-underline current line with level 6 adornment	<f11> _ 6</f11>	(pel-commented-adorn-6)	Insert a commented level-6 reST line adornment at point.		
Comment-underline current line with level 7 adornment	<f11> _ 7</f11>	(pel-commented-adorn-7)	Insert a commented level-7 reST line adornment at point.		
Comment-underline current line with level 8 adornment	<f11> _ 8</f11>	(pel-commented-adorn-8)	Insert a commented level-8 reST line adornment at point.		
Comment-underline current line with level 9 adornment	<f11> _ 9</f11>	(pel-commented-adorn-9)	Insert a commented level-9 reST line adornment at point.		
Comment-underline current line with level 10 adornment	<f11> _ 0</f11>		Insert a commented level-10 reST line adornment at point.		
Smart Dash Mode	This uses the <u>smart-dash</u> external package. PEL activates it when <b>pel-use-smart-dash</b> is set to t.  The <b>pel-modes-activating-smart-dash-mode</b> user option identifies the major modes where PEL automatically activates the smart-dash-mode.				
	Anyone that has been writing Lisp code for a while knows that using dash as word separator instead of underscore is more natural and faster to type. Unfortunately most programming languages (all non-Lisp?) have restrictions on the characters available in identifiers and underscore is often used. Typing underscore requires hitting the Shift key and it annoys some people that enjoyed writing Lisp code. This is where the smart-dash-mode helps. You can insert underscore in text by typing the dash key without hitting the Shift key! A <b>very</b> useful mode. More information is available in the <b>author's page</b> .				
Toggle smart dash mode	<f11> M</f11>	(smart-dash-mode &optional ARG)	Toggle the smart-dash-mode on/off.		
See also:  • ∑ Numkeypad  • ∑ Text Modes	When smart-dash-mode is active, it redefines the dash key to insert an underscore within C-style identifiers and a dash otherwise. This allows you to type all_lowercase_c_identifiers as comfortably as you would lisp-style-identifiers.  While Smart-Dash mode is active, you can type C-q — or use the minus key on the numeric keypad to override it and insert a dash after a C-style identifier character. You might need to do this if you want to type a cramped-looking expression like x-5.  If Smart-Dash mode is activated while in a C-like mode (c-mode, c++-mode, and objc-mode by default, customizable with 'smart-dash-c-modes') it will also activate Smart-Dash-C mode, which translates "_>" into "->" and "" into "" automatically so that struct pointer member access and postfix-decrement aren't made more difficult by Smart-Dash mode's tendency to insert underscores at the tail ends of identifiers whether you want it to or not. Note that this will necessitate that you type literal underscores if you want more than one underscore in a row.  With PEL, the keypad '-' is not affected, allowing the insertion of dash character as long as Emacs is operating in numlock ON:  in numlock ON mode the keypad '-' inserts a dash character,  in numlock OFF it kills the current line.  For more information, see ∑ Numkeypad.  Also note that as soon as dash character is before point typing '-' from any key will produce a dash character.				
	With PEL type <f11> t m ? to display the status of text modes including dash-mode.</f11>				
Text and code skeletons	Several mechanisms have been developed to allow easy insertion of predefined text in Emacs.  • Emacs provides the built-in skeleton mechanism and the tempo skeletons.				
tempo skeletons	PEL supports both. They are used a little bit differently.  • PEL provides key bindings to the tempo skeletons: the generic code templates, accessible via the <f6> prefix key, and the language-specific code templates, accessible via the <f12> key prefix.</f12></f6>				
Generic skeletons	PEL provides <b>generic</b> tempo skeletons as well as some specialized for specific programming languages. The generic skeletons are less powerful but often good enough for most types of files. They support all types of files recognized by Emacs as long as Emacs understands the way comments work for the file type which is normally the case. If Emacs does not know the file type the commands assume the file uses a comment start only and will prompt for that string.				
<u>∑ Customize</u> PEL Text Insertions control	<f6> <f2></f2></f6>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL generic tempo skeleton customization groups that control the format of the various skeletons including the generic skeleton used by the <f6> h key (se below).  • If OTHER-WINDOW is non-nil (use C-u), display in other window.</f6>		
Insert generic file module header block — Language agnostic  After inserting the template, navigate though areas that must be filled with:  • tempo-forward-mark: C-c.  • tempo-backward-mark: C-c,	<f6> h</f6>	(pel-generic-file-header)	Insert a file header block at the top of the file. Works only for buffer visiting a file.  Supports all text file types.  • Supports all programming and markup language files that have a dedicated major mode. It is also available in buffers for major modes explicitly supported by the <f12> <f12> key prefix. This way, those modes can use two different commands to insert file header blocks, each having its own different format.  It supports several programming and markup language and uses the comment style identified by the file extension. If the comment style is unknown the command prompts for one.  • The layout of the entered text is controlled by user options. It is possible to create a user-specified skeleton this command will used instead of the one provided by PEL.  • Specify the format of the header via the user-options in the pel-pkg-generic-code-style customization group accessible via <f6> <f2>  • The files that have no extensions are often used in Unix-like OS shell scripts. These files are also supported as Emacs can recognize them if they are stored in a bin directory. PEL also has special support for them and is controlled by the pel-shell-script-skeleton-control customization group, which is accessible as a child of the main group.  • After inserting the template you can use the tempo-forward-mark and tempo-backward-mark to move point to the beginning of each section that must be filled.</f2></f6></f12></f12>		
			The command key binding <f6> h is available only 1 second after Emacs has started.</f6>		

Description	Kovotvolica	Function	Note		
<u>Description</u>	Keystroke	Function	Note		
Toggle pel-tempo-mode	<f6> SPC</f6>	(pel-tempo-mode &optional ARG)	Toggle PEL tempo mode on/off. PEL tempo mode activates C-c . and C-c , as well as to C-c C and C-c C-, key bindings to navigate across tempo mark hot-spots. When pel-tempo-mode is active the pel-tempo-mode lighter (‡) is shown on the status bar. The second set of keys are only available when Emacs runs in graphics mode.  The pel-generic-file-header command inserts the text using a tempo skeleton: the PEL tempo mode is automatically activated by typing <f6> h.</f6>		
Jump to next tempo mark	• C-c M-f • C-c . • C-c C	(tempo-forward-mark)	Jump to the next mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton.  • These key key bindings are only available when pel-tempo-mode is active.		
Jump to previous tempo mark	• C-c M-b • C-c , • C-c C-,	(tempo-backward-mark)	Jump to the previous mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton.  • These key binding are only available when pel-tempo-mode is active.		
Store PEL code template settings in .dir-locals.el to fine-tune layout of files in a directory tree	Emacs user options by default take effect globally. But by using file and directory variables (see <u>File/Directory Variables</u> ) they can also be used to take effect on a single file or all files inside a directory tree. So by default, the user options that control the PEL tempo template take effect globally. If you want to change the behaviour for only one file, write the user option control block at the end of that file. If you want to control the behaviour of the PEL tempo templates for all files inside a directory tree create a .dir-locals file and store the values of the relevant options variables inside that file. This allows you to control the user options affecting the format of the tempo templates precisely.				
Example:	Although the default settings of <b>pel-generic-skel-module-section-titles</b> identifies the 3 sections "Module Description", "Dependencies" and "Code" you can keep this for other files but inside a directory you can force all shell-mode files to use 2 sections: "Description" and "Script" and ensure that all files have a 1-line copyright notice with the .dir-locals.el file containing the following code:				
	<pre>;;; Directory Local Variables ;;; For more information see (info "(emacs) Directory Variables")</pre>				
	((nil . (	(pel-generic-skel-with-	copyright . t)		
	<pre>((nil . ((pel-generic-skel-with-copyright . t)</pre>				
Entering Templated Text with <u>Tempo Skeletons</u> See also:	Emacs built-in support includes the tempo skeletons.  PEL implements extension to the tempo skeleton Emacs built-in package under two prefix keys:  The commands under the <f6> prefix keys insert template text that are adapted to each major mode. They are generic in nature, and dynamically adapt to the major mode and the comment style supported by the major mode. The layout of the templates is the same for every major mode, they</f6>				
<ul><li>Major mode specific:</li><li>\$1 - C</li></ul>	differ only by the comment strings.  • The commands under the <f12> <f12> prefix key insert templates specialized for the programming or markup language of the major mode that</f12></f12>				
• ⊈®t - Emacs Lisp	support this key prefix. PEL attempts to use the same key bindings for equivalent concepts (such as file header block) inside each mode specific instance of the <f12> key maps as much as possible. The tempo skeletons provided by PEL can be quite complex and their formats are</f12>				
• <u>∯ĭ - Erlang</u> • <u>M reStructuredText</u>	controlled by user options. PEL currently only support this key prefix with for the following major modes (more are planned):  • C, Emacs Lisp, Erlang  • reStructuredText				
Major-mode specific	<f12> <f12></f12></f12>		Key prefix sequence to the list of tempo skeleton commands.		
Tempo Templates Prefix			This command prefix is available only for some major modes (see the list in the first column) of the section row above.		
			The commands under this prefix insert text specialized for their specific major mode, as opposed to the commands bound to the <f6> prefix key. For more information see the</f6>		
			language specialized reference table.		
Entering Templated Test with <u>Yasnippet</u>	PEL also supports the popular <b>yasnippet</b> external package which provides another way to insert templated text, and <b>yasnippet-snippets</b> external package which provides a large set of code snippets for a large set of major modes.				
See also: <u></u> Customize	To use yasnippets,	you must type the snippet abbr	reviation and then hit the TAB key to expand the text.		
			yasnippet is set to t or to use-from-start.		
	Use the key <f11></f11>	Requires vasnippet-snippets dativated when pel-use-yasnippet-snippets is set to t.  Use the key <f11> i <f2> to access the PEL Insertion customization buffer to customize these user options (see above, first row).</f2></f11>			
	• The list of snippets available in the current buffer is listed in the menu bar (see <u>Nenus</u> ) and can also be listed using the yas-describe-tables command (which PEL binds to <f11> y t).</f11>				
	PEL bins the following	PEL bins the following vasnippet commands to keys in the <b>pel:</b> key prefix, shown below.			
∑ Customize PEL yasnippet	<f11> y <f2> (pel-customize-pel Customize PEL Yasnippet text insertion support.</f2></f11>				
use			If OTHER-WINDOW is non-nil (use <b>C-u</b> ), display in other window.		
<u>S Customize</u> Emacs yasnippet control	<f11> y <f3></f3></f11>	(pel-customize-library &optional OTHER-WINDOW)	Customize Yasnippet groups: yasnippet, yasnippet-snippets, yas-minor		
Toggle YASnippet minor mode on/off	<f11> y y</f11>	(yas-minor-mode &optional ARG)	Toggle YaSnippet mode.		
	<ul> <li>When YASnippet mode is enabled, 'yas-expand', normally bound to the TAB key, expands snippets of code depending on the major mode.</li> <li>With no argument, this command toggles the mode. Positive prefix argument turns on the mode. Negative prefix argument turns off the mode.</li> <li>YASnippet mode key bindings:         <ul> <li>key</li> <li>binding</li> </ul> </li> </ul>				
	C-c & C-n yas-new-snippet				
		yas-insert-snippet yas-visit-snippet-file			
Toggle YASnippet global mode on/off	<f11> y Y</f11>	(yas-global-mode &optional ARG)	Toggle Yas minor mode in all buffers.  • With prefix ARG, enable Yas-Global mode if ARG is positive; otherwise, disable it.		
Expand snippet whose name is just before point	ТАВ	(yas-expand &optional FIELD)	Expand a snippet before point. If no snippet expansion is possible, do nothing.  • This key binding is only active when the YASnippet mode is active. Once the snippet was expanded the TAB key normal behaviour is restored.		
Write a new snippet	• <f11> y n • C-c &amp; C-n</f11>	(yas-new-snippet &optional NO-TEMPLATE)	Pops a new buffer for writing a snippet.  Expands a snippet-writing snippet, unless the optional prefix arg NO-TEMPLATE is non-nil.		
Prompt for snippet and insert it	• <f11> y s</f11>	(yas-insert-snippet &optional NO-CONDITION)	Choose a snippet to expand, pop-up a list of choices according to 'yas-prompt-functions'.  • With prefix argument NO-CONDITION, bypass filtering of snippets by condition.		
	• C-c & C-s	, 2 22.2			
Visit a snippet file	• <f11> y v</f11>	(yas-visit-snippet-file)	Choose a snippet to edit, selection like 'yas-insert-snippet'.		
	• C-c & C-v		Only success if selected snippet was loaded from a file. Put the visited file in 'snippet-mode'.		
Display all snippets for current major mode	<f11> y t</f11>	(yas-describe-tables &optional WITH- NONACTIVE)	Display snippets for each table.		
Prints Yasnippet version info	<f11> y ?</f11>	(yas-about)	Prints version information in the mini buffer.		

## Inserting Text - References

Topic & link	Description
GNU Emacs Manual: Time Stamps	
Smart-Dash Mode homepage	A description of this extremely useful mode and why it was created.