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**.		
Text Insertions	`<f11> <f1> I`	(**pel-cfg-insertions** &optional OTHER-WINDOW)	Customize PEL text insertion support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window and open insertion related and supported groups: **lice**, **smart-dash**, **yasnippet** and **yasnippet-snippets**.
Time-stamps			
Insert current date	`<f11> i d`	(**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`	(**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`	(**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, if the numeric argument is negative the path is omitted. ☛ When creating keyboard macros that must insert text that includes a file name, you can open 2 windows and use this facility to enter the name of the file that is located in the other window, making the keyboard macro more general.
Insert time stamp	`<f11> i t`	(**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`	(**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. 📦 Requires the **lice** external package. PEL activates it if **pel-use-lice** user option is t.
Updating file time stamp			
Update file time stamp (See Also: ∑ File mngt)	`<f11> f t`	(**time-stamp**)	Force update the time stamp string(s) in the current buffer. The time stamp is updated if the one of the following strings is found in the first 8 lines of the file: • Time-stamp: <> • Time-stamp: “ “ ☝ If you want time stamps updated automatically, write the following inside your init.el file: `(add-hook 'before-save-hook 'time-stamp)`
Toggle time stamp automatic update		(**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.
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 (See Also: ∑ File mngt)	`<f11> i C`	(**copyright** &optional STR ARG)	Insert a copyright by $ORGANIZATION notice at cursor. • If the ORGANIZATION environment variable is not available, Emacs prompts for it.
Update file’s copyright notice		(**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. • If non-nil, INTERACTIVEP tells the function to behave as when it’s called interactively. ⚠ 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 to be prompted automatically to update an existing but out-of-date copyright notice, write the following inside your init.el file: `(add-hook 'before-save-hook ‘copyright-update)`
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: ∑ Comments)**	• `<f11> i l` • `<f6> l`	(**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. ☛ fill-column is customizable and can be used as a file or directory variable.
Comment-underline current line with level 1 adornment	`<f11> _ 1`	(**pel-commented-adorn-1**)	Insert a commented level-1 reST line adornment at point.
Comment-underline current line with level 2 adornment	`<f11> _ 2`	(**pel-commented-adorn-2**)	Insert a commented level-2 reST line adornment at point.
Comment-underline current line with level 3 adornment	`<f11> _ 3`	(**pel-commented-adorn-3**)	Insert a commented level-3 reST line adornment at point.
Comment-underline current line with level 4 adornment	`<f11> _ 4`	(**pel-commented-adorn-4**)	Insert a commented level-4 reST line adornment at point.
Comment-underline current line with level 5 adornment	`<f11> _ 5`	(**pel-commented-adorn-5**)	Insert a commented level-5 reST line adornment at point.
Comment-underline current line with level 6 adornment	`<f11> _ 6`	(**pel-commented-adorn-6**)	Insert a commented level-6 reST line adornment at point.
Comment-underline current line with level 7 adornment	`<f11> _ 7`	(**pel-commented-adorn-7**)	Insert a commented level-7 reST line adornment at point.
Comment-underline current line with level 8 adornment	`<f11> _ 8`	(**pel-commented-adorn-8**)	Insert a commented level-8 reST line adornment at point.
Comment-underline current line with level 9 adornment	`<f11> _ 9`	(**pel-commented-adorn-9**)	Insert a commented level-9 reST line adornment at point.

Description	Keystroke	Function	Note
Comment-underline current line with level 10 adornment	`<f11> _ 0`	(**pel-commented-adorn-10**)	Insert a commented level-10 reST line adornment at point.
Smart Dash Mode	📦 This uses the **smart-dash** external package. 🔄 PEL activates it when **pel-use-smart-dash** is set to **t**. 🔄 The **pel-modes-activating-smart-dash-mode** 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 **very** useful mode. More information is available in the **author's page**.		
Toggle smart dash mode (See also: ∑⌨ Numkeypad)	`<f11> M--`	(**smart-dash-mode** &optional ARG)	Toggle the smart-dash-mode on/off. • 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.
Text and code skeletons and snippets	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**. • The popular **yasnippet** external library is more recent and provides a very flexible and powerful mechanism to create code templates and several people created so-called snippets for various programming and markup languages. PEL supports both. They are used a little bit differently. PEL provides key bindings to the tempo skeletons, not for yasnippets. To use yasnippets, you must type the snippet abbreviation and then hit the TAB key to expand the text.		
Entering Templated Text with Tempo Skeletons (See also: • 𝔓 - C • 𝔈𝔓 - Emacs Lisp • 𝔓 - Erlang • 𝔐 reStructuredText)	Emacs built-in support includes the **tempo skeletons**. PEL implements extension to the tempo skeleton Emacs built-in package for some major modes. The currently supported modes are: • C, Emacs Lisp, Erlang • reStructuredText PEL creates key bindings to invoke the skeletons in the supported major modes, using the same key prefix sequence for each mode: `<f12> <f12>`, with the same key bindings for equivalent concepts (such as file header block) as much as possible. The tempo skeletons provided by PEL can be quite complex and their formats are controlled by user options. ☝ Emacs user options by default take effect globally. But by using file and directory variables (see ∑ File/Directory Variables) 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 and does not affect what you actually type.		
Tempo Templates Prefix	`<f12> <f12>`		Key prefix sequence to the list of tempo skeleton commands. See the complete list of commands in the table describing the key binding of the supported major mode.
Entering Templated Test with Yasnippet	PEL also supports the popular **yasnippet** external package which provides another way to insert templated text, and **yasnippet-snippets** external package which provides a large set of code snippets for a large set of major modes. 📦 Requires **yasnippet** 🔄 activated when **pel-use-yasnippet** is set to **t** or to **use-from-start**. 📦 Requires **yasnippet-snippets** 🔄 activated when **pel-use-yasnippet-snippets** is set to **t**. • Use the key `<f11> <f1> I` to access the PEL Insertion customization buffer to customize these user options (see above, first row). • The list of snippets available in the current buffer is listed in the menu bar (see ∑ Menus) and can also be listed using the yas-describe-tables command (which PEL binds to **<f11> y t**). PEL bins the following yasnippet commands to keys in the **pel:** key prefix, shown below.		
Customize Yasnippet	`<f11> y <f1>`	(**pel-customize-yasnippet**)	Open the yasnippet customization buffer.
Toggle YASnippet minor mode on/off	`<f11> y y`	(**yas-minor-mode** &optional ARG)	Toggle YASnippet mode. • When YASnippet mode is enabled, 'yas-expand', normally bound to the TAB key, expands snippets of code depending on the major mode. • With no argument, this command toggles the mode. Positive prefix argument turns on the mode. Negative prefix argument turns off the mode. • YASnippet mode key bindings: key binding `C-c & C-n` yas-new-snippet `C-c & C-s` yas-insert-snippet `C-c & C-v` yas-visit-snippet-file
Toggle YASnippet global mode on/off	`<f11> y Y`	(**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	`TAB`	(**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 & C-n`	(**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` • `C-c & C-s`	(**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.
Visit a snippet file	• `<f11> y v` • `C-c & C-v`	(**yas-visit-snippet-file**)	Choose a snippet to edit, selection like 'yas-insert-snippet'. • 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`	(**yas-describe-tables** &optional WITH-NONACTIVE)	Display snippets for each table.
Prints Yasnippet version info	`<f11> y ?`	(**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.