

































Inserting Text

Description	Keystroke	Function	Note
<div>Inserting Text</div> <ul style="list-style-type: none">Insert quote char, Greek letterFile/Directory nameSoftware License, lice, spdxTo-do noteDate/TimeAutomatic Time StampingAutomatic Copyright noteCommented linesSmart Dash modeSmartparens modeText tempo skeletonsYasnippet See also: 🔗 Abbreviations	The commands described in this table insert specialized text at point (cursor) location. This includes: <ul style="list-style-type: none">Customization to control automatic insertion of time stamp, update of copyright notice. <ol style="list-style-type: none">Simple command based text insertions:<ul style="list-style-type: none">PEL specialized commands to insert formatted text like time stamps, file path , file name, copyright notice, Greek letters, commented lines, etc... The lice  activated by pel-use-lice, and spdx activated by  pel-use-spdx user options, can be used to insert open source licence text. The smart-dash external package  activated by pel-use-smart-dash, used to automatically convert dash into underscore when typing. The smartparens external package  activated by pel-use-smartparens to provide automatic insertion of balanced block pairs for code.Specialized template-based text insertion:<ul style="list-style-type: none">PEL tempo skeletons based templates for generic & specialized boilerplate file sections: file, class, function header, document section header. The yasnippet external package  activated by pel-use-yasnippet to insert code from predefined snippets. The yasnippet-snippets external package  activated by pel-use-yasnippet-snippets which provides a large amount of snippets.Hydra-based insertion of Greek letters:<ul style="list-style-type: none"> The hydra external package.  activated by pel-use-hydra and pel-activate-hydra-for-greek.		
Last updated on:	2026-02-25	See also 🔗 T Templates	
Open this PDF file. See also: 🔗 Help/Info	<ul style="list-style-type: none"><code><f11> i <f1></code><code><f11> y <f1></code><code><f11> _ <f1></code>	(pel-help-pdf & optional OPEN-WEB-PAGE)	Open the 🔗 Inserting Text 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 Text Insertions control	<code><f11> i <f2></code>	(pel-customize-pel & optional OTHER-WINDOW)	Customize PEL text insertion support: lice, smart-dash, smartparens, tempo, time-stamp, yasnippet. Also pel-activate-f9-for-greek . If OTHER-WINDOW is non-nil (use C-u) , display in other window.
🔗 Customize Emacs Text Insertions control	<code><f11> i <f3></code>	(pel-customize-library & optional OTHER-WINDOW)	Customize Emacs text insertion support: lice, smart-dash, tempo, time-stamp, yasnippet
Insert <i>quoted</i> char	C-q	(quoted-insert ARG)	Read next input character and insert it. Useful for inserting control characters.
	<ul style="list-style-type: none">With argument, insert ARG copies of the character.If the first character you type is an octal digit, the sequence of one or more octal digits you type is interpreted to specify a character code. Any character that is not an octal digit terminates the sequence. If the terminator is a RET, it is discarded; any other terminator is used itself as input and is inserted.		
Insert Greek Letter See also: 🔗 Input Method  In macOS you can also: add Greek as a Keyboard Input Sources and temporary select it to enter Greek characters.	<ul style="list-style-type: none"><code><f9></code> <code><f6> g</code> 	Insert a greek letter: type <code><f9></code> followed by a key in [a-zA-Z] range inserts the Unicode character for the equivalent Greek letter. Examples: <code><f9> a</code> inserts α <code><f9> b</code> inserts β <code><f9> A</code> inserts Α <code><f9> B</code> inserts Β <code><f9> 1</code> inserts λ	
	The insertions work everywhere insert is allowed, including in minibuffer. Use <code><f9> C-h</code> or which-key mode and type <code><f9></code> to see all keys.  The <code><f9></code> key binding is only available when the pel-activate-f9-for-greek user-option is turned on. The <code><f6> g</code> binding is always available.  It's not a command bound to a key: it's an additional set of bindings added to Emacs key-translation-map .		
	<code><f7> G</code>	Start the Greek letter Hydra . <ul style="list-style-type: none">After typing <code><f7> G</code> type one of the Meta letter keys in the hydra to insert a Greek character.Type any other character to insert them, latin letters, digits, punctuation characters, and Meta-char to inter the greek character, etc...In terminal mode the cursor keys may not work because they are often encoded using Esc keys with is mapped to Meta. Use C-b, C-f, C-n and C-p instead.While this hydra is active, use <code><f11> u u</code> for undo.Exit the hydra by typing <code><f7></code>The key bindings are shown by the hydra:	
 Requires the hydra external package.  activated by pel-use-hydra .  You must also set pel-activate-hydra-for-greek to t to activate this hydra.	[M-a]: α, [M-b]: β, [M-c]: χ, [M-d]: δ, [M-e]: ε, [M-f]: φ, [M-g]: γ, [M-h]: η, [M-i]: ι, [M-j]: ϕ, \[M-k]: κ, [M-l]: λ, [M-m]: μ, [M-n]: ν, [M-o]: ο, [M-p]: π, [M-q]: θ, [M-r]: ρ, [M-s]: σ, [M-t]: τ, \[M-u]: υ, [M-w]: ω, [M-x]: ξ, [M-y]: ψ, [M-z]: ζ, [M-A]: Α, [M-B]: Β, [M-C]: Χ, [M-D]: Δ, [M-E]: Ε, \[M-G]: Γ, [M-H]: Η, [M-I]: Ι, [M-J]: Φ, [M-K]: Κ, [M-L]: Λ, [M-M]: Μ, [M-N]: Ν, [M-O]: Ο, [M-P]: Π, \[M-Q]: Θ, [M-R]: Ρ, [M-S]: Σ, [M-T]: Τ, [M-U]: Υ, [M-W]: Ω, [M-X]: Ξ, [M-Y]: Ψ, [M-Z]: Ζ, [<f7>]: ca\ncel.		
Insert file/directory name			
The following commands insert the name of the file or directory of the current file or file in specified window.			
Change filename root for the current buffer.  See also:  PEL Environment Variables	<code><f6> <f4> f</code>	(pel-set-insert-filename-root)	Change the root directory that is stripped off the absolute path of the file name inserted by pel-insert-filename .
	This change is buffer specific and overrides the default value imposed by the pel-insert-filename-root user option or the value of the PEL_INSERT_FILENAME_ROOT environment variable. The value is taken in the following priority order (starting with the highest priority): <ul style="list-style-type: none">value set by this command , PEL_INSERT_FILENAME_ROOT environment variable value, pel-insert-filename-root user option		
Customize to-do note format	<code><f11> i <f4> f</code>	(pel-customize-insert-filename)	Open the customize buffer to change the 'pel-insert-filename-root' user-option.
Insert, at point, name of: <ul style="list-style-type: none">current filename by defaultname of file in window identified by command numeric argument <ul style="list-style-type: none">Append line number of point in visited file <ul style="list-style-type: none">If file is in user home, use ~ at the beginning	<ul style="list-style-type: none"><code><f11> i f</code><code><f6> f</code>	(pel-insert-filename & optional N USE-TILDE DIR-ONLY)	Insert the file name of the currently edited file at point. <ul style="list-style-type: none">By default, or with 1, insert filename of current buffer with complete absolute path.
	<ul style="list-style-type: none"><code><f11> i F</code><code><f6> F</code>	(pel-insert-filename-and-line & optional N)	Same as above but also append a colon and a line number, followed by a line end. 👉 Useful for manually building a list of files inside a buffer. Later use the pel-open-at-point command M-<f6> (see 🔗 File-mngt) to open that file and move point to that line.
	<ul style="list-style-type: none"><code><f11> i M-f</code><code><f6> M-f</code>	(pel-insert-filename-wtilde & optional N)	Same as first command above, except that if the file is located in the current user home, insert the Unix-style tilde character ~ in place of the user home directory name.
	<ul style="list-style-type: none"><code><f11> i C-f</code><code><f6> C-f</code>	(pel-insert-dirname & optional N USE-TILDE)	Insert the directory path name of the currently edited file at point. <ul style="list-style-type: none">By default, or with 1, insert directory name of file in current buffer.
<ul style="list-style-type: none">If file is in user home, use ~ at the beginning	<ul style="list-style-type: none"><code><f11> i C-M-f</code><code><f6> C-M-f</code>	(pel-insert-dirname-wtilde & optional N)	Same as the above command, except that <i>if the file is located in the current user home</i> , insert the Unix-style tilde character ~ in place of the user home directory name.
	<code><f11> ? F</code>	(pel-show-function & optional INSERT-IT)	Display the name of the current “function” at point in the mini-buffer. See also: 🔗 Help/Info <ul style="list-style-type: none">With any argument, like C-u , also insert the “function” name at point.
Insert software license			
 The lice command, requires the lice external package  PEL activates it if pel-use-lice user option is t.			
Insert software license text	<ul style="list-style-type: none"><code><f11> i L</code><code><f6> L</code>	(lice NAME)	Insert license and headers at point. Prompts for license NAME, which is a license template name like “mit”, “gpl-3.0”, etc... Support TAB completion to get the complete list of templates.
Insert SPDX software license	<ul style="list-style-type: none"><code><f11> i M-l</code><code><f6> M-l</code>C-c i l	(spdx-insert-spdx)	Insert a SPDX license header. Prompts for the SPDX license type. Supports tab completion.  Requires spdx activated by  pel-use-spdx user option.
Insert to-do note			
The default to-do note format is: <code>[:todo (DATE) , by (USER):]</code> Automatically inserts the date and user name. Other popular prefix TODO :			
Insert To-Do note	<ul style="list-style-type: none"><code><f11> i n</code><code><f6> n</code>	(pel-insert-todo-note & optional UTC)	Insert a to-do note template comment. The format is configured by the pel-todo-note-text and the pel-todo-note-date-format user options.
	<ul style="list-style-type: none">If “(USER)” is in pel-todo-note-text, it is replaced by the user name. If it has “(DATE)”, it is replaced by the date using the format specified pel-todo-note-date-format. By default the local date is entered. Use the C-u prefix to use UTC instead. Point is placed one character before the end of the inserted text.		
Customize to-do note format	<code><f11> i <f4> n</code>	(pel-customize-todo-note)	Open the customize buffer to change the to-do note format user options.

Description	Keystroke	Function	Note												
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 • ☞ Filling/Justification	<ul style="list-style-type: none">• <f11> i 1• <f6> 1	(pel-insert-line &optional LINELEN)	Insert a (commented) line before/at current line. <ul style="list-style-type: none">• 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:<ul style="list-style-type: none">• 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 adornment 1-9	<f11> _ ☺	(pel-commented-adorn-1)	Insert a commented level-x reST line adornment at point. <ul style="list-style-type: none">• ☺ := 1 to 9 for levels 1 to 9												
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	<ul style="list-style-type: none">• 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. <div>📦 Requires the smart-dash external package.  PEL activates it when pel-use-smart-dash is set to t.</div> <div> To activate smart-dash-mode automatically:<ul style="list-style-type: none">• for major modes supported by PEL, add smart-dash-mode to the pel-<MODE>activates-minor-mode user-option for the specific mode.• for other modes, add the mode name to the pel-modes-activating-smart-dash-mode user-option.</div>														
Toggle smart-dash mode See also: • ☞ Numkeypad • ☞ Text Modes • ☞ Mode Line	<f11> i -	(smart-dash-mode &optional ARG)	Toggle the smart-dash-mode on/off. <ul style="list-style-type: none">• 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. <div>⚠ Normally when smart-dash-mode is active the numeric dash key (<kp-subtract>) acts as a smart-dash only.</div> <div>👉 However, with PEL, the behaviour of the keypad '-' is only partly affected when the smart-dash-mode is active and it depends on the Numlock state:<ul style="list-style-type: none">• In Numlock OFF:<table><tr><td>with no marked area:</td><td>insert a dash. Numeric argument for multiple insertion is not supported.</td></tr><tr><td>with an area marked:</td><td>kill marked area</td></tr><tr><td>with area marked with er/expand-region:</td><td>kill marked area</td></tr></table>• In Numlock ON:<table><tr><td>with no marked area:</td><td>insert an underscore after letter, number or underscore, dash otherwise</td></tr><tr><td>with area marked normally:</td><td>Ignore the marked area; insert a dash at point</td></tr><tr><td>with area marked with er/expand-region:</td><td>Reduces the marked area semantically as controlled by er/expand-region</td></tr></table></div> <div>For more information on the NumLock control and key support, see ☞ Numkeypad.</div> <div>👉 With PEL type <f11> t m ? to display the status of text modes including dash-mode.</div> <div>👉 With PEL, when pel-use-delight is turned on, a short lighter of a green dash is showing in the mode line when smart-dash-mode is active.</div>	with no marked area:	insert a dash. Numeric argument for multiple insertion is not supported.	with an area marked:	kill marked area	with area marked with er/expand-region:	kill marked area	with no marked area:	insert an underscore after letter, number or underscore, dash otherwise	with area marked normally:	Ignore the marked area; insert a dash at point	with area marked with er/expand-region:	Reduces the marked area semantically as controlled by er/expand-region
with no marked area:	insert a dash. Numeric argument for multiple insertion is not supported.														
with an area marked:	kill marked area														
with area marked with er/expand-region:	kill marked area														
with no marked area:	insert an underscore after letter, number or underscore, dash otherwise														
with area marked normally:	Ignore the marked area; insert a dash at point														
with area marked with er/expand-region:	Reduces the marked area semantically as controlled by er/expand-region														
Smartparens Mode • Smartparens manual	Simplify insertion of matching pairs with the smartparens minor mode. PEL binds a set of keys, described below, to toggle activation of that mode. <div>📦 This uses the smartparens external package.  PEL activates it when pel-use-smartparens is set to t.</div> <ul style="list-style-type: none">• Smartparens enhances the behaviour of certain keys, namely those that are part of any pair or tag.• Mode line lighter: smartparens-mode: SP smartparens-strict-mode: SP/s														
Help on smartparens	<f11> (?	(sp-cheat-sheet &optional ARG)	Generate a cheat sheet of all the smartparens interactive functions. Shows inside Emacs buffer. <ul style="list-style-type: none">• Without a prefix argument, print only the short documentation and examples.• With non-nil prefix argument ARG, show the full documentation for each function.• You can follow the links to the function or variable help page.<ul style="list-style-type: none">• To get back to the full list, use M-x help-go-back.• You can use 'beginning-of-defun' and 'end-of-defun' to jump to the previous/next entry.• Examples are fontified using the 'font-lock-string-face' for better orientation.												
Describe user system	<f11> (M-?	(sp-describe-system STARTERKIT)	Describe user's system. Prompt for starter kit: Evil, Spacemacs, Vanilla. <ul style="list-style-type: none">• The output of this function can be used in bug reports.												
Toggle smartparens mode	<f11> (((smartparens-mode &optional ARG)	Toggle smartparens mode.												
Toggle smartparens-strict mode	<f11> ()	(smartparens-strict-mode &optional ARG)	Toggle the strict smartparens mode. <ul style="list-style-type: none">• When strict mode is active, 'delete-char', 'kill-word' and their backward variants will skip over the pair delimiters in order to keep the structure always valid (the same way as 'paredit-mode' does). This is accomplished by remapping them to 'sp-delete-char' and 'sp-kill-word'. There is also function 'sp-kill-symbol' that deletes symbols instead of words, otherwise working exactly the same (it is not bound to any key by default).• When strict mode is active, this is indicated with "/s" after the smartparens indicator in the mode list												
Toggle smartparens mode	<f11> (M-((smartparens-global-mode &optional ARG)	Toggle Smartparens mode in all buffers. <ul style="list-style-type: none">• With prefix ARG, enable Smartparens-Global mode if ARG is positive; otherwise, disable it.• Smartparens mode is enabled in all buffers where 'turn-on-smartparens-mode' would do it.												
Toggle smartparens-strict mode	<f11> (M-)	(smartparens-global-strict-mode &optional ARG)	Toggle Smartparens-Strict mode in all buffers. <ul style="list-style-type: none">• With prefix ARG, enable Smartparens-Global-Strict mode if ARG is positive; otherwise, disable it.• Smartparens-Strict mode is enabled in all buffers where 'turn-on-smartparens-strict-mode' would do it.												

Description	Keystroke	Function	Note
Text and code skeletons <ul style="list-style-type: none"> tempo skeletons Generic skeletons 	Several mechanisms have been developed to allow easy insertion of predefined text in Emacs. <ul style="list-style-type: none"> Emacs provides the built-in skeleton mechanism and the tempo skeletons. PEL supports both. They are used a little bit differently. <ul style="list-style-type: none"> 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. PEL provides generic 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.		
ⓘ Customize PEL Text Insertions control	<f6> <f2>	(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). <ul style="list-style-type: none"> If OTHER-WINDOW is non-nil (use C-u) , display in other window.
Insert generic file module header block — Language agnostic <p>After inserting the template, navigate through areas that must be filled with:</p> <ul style="list-style-type: none"> tempo-forward-mark: C-c . tempo-backward-mark: C-c , See examples in manual	<f6> h	(pel-generic-file-header)	Insert a file header block at the top of the file. Works only for buffer visiting a file. <p>Supports all text file types.</p> <ul style="list-style-type: none"> 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. <p>👉 Specify the format of the header via the user-options in the pel-pkg-generic-code-style customization group accessible via <f6> <f2></p> <ul style="list-style-type: none"> 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-sh-script-skeleton-control customization group, which is accessible as a child of the main group. <p>👉 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.</p> <p>⚠️ The command key binding <f6> h is available only 1 second after Emacs has started.</p>
Toggle pel-tempo-mode	<f6> SPC	(pel-tempo-mode &optional ARG)	Toggle PEL tempo mode on/off. <p>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.</p> <p>👉 The pel-generic-file-header command inserts the text using a tempo skeleton: the PEL tempo mode is automatically activated by typing <f6> h.</p>
Jump to next tempo mark	<ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> These key key bindings are only available when pel-tempo-mode is active.
Jump to previous tempo mark	<ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> 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 <p>Example:</p>	<p>👉 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.</p> <p>Although the default settings of pel-generic-skel-module-section-titles 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:</p> <pre> ;;; Directory Local Variables ;;; For more information see (info "(emacs) Directory Variables") ((nil . ((pel-generic-skel-with-copyright . t) (pel-generic-skel-with-license . "MIT"))) (sh-mode . ((pel-generic-skel-module-section-titles . ("Description" "Script"))))) </pre>		
Entering Templated Text with Tempo Skeletons <p>See also:</p> <ul style="list-style-type: none"> Major mode specific: <ul style="list-style-type: none"> ⚡ - C ⚡ - C++ ⚡ - Emacs Lisp ⚡ - Erlang ⚡ reStructuredText 	Emacs built-in support includes the tempo skeletons . <p>PEL implements extension to the tempo skeleton Emacs built-in package under two prefix keys:</p> <ul style="list-style-type: none"> 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 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 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> <f12> key maps as much as possible. The tempo skeletons provided by PEL can be quite complex and their formats are controlled by user options. PEL currently only support this key prefix with for the following major modes (more are planned): <ul style="list-style-type: none"> C, C++, Emacs Lisp, Erlang reStructuredText 		
Major-mode specific Tempo Templates Prefix	<f12> <f12>	Key prefix sequence to the list of tempo skeleton commands. <ul style="list-style-type: none"> 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 language specialized reference table. 	

Description	Keystroke	Function	Note									
<div>Entering Templated Test with Yasnippet</div> <div><div>See also:Customize</div><div>Activate a yasnippet by identifying the directory holding the snippet files. ➡</div></div>	<div>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.</div> <div><div>• To use yasnippets, you must type the snippet abbreviation and then hit the TAB key to expand the text.</div><div>📦 Requires yasnippet  activated when pel-use-yasnippet is set to t or to use-from-start.</div><div>📦 Requires yasnippet-snippets  activated when pel-use-yasnippet-snippets is set to t.</div><div><div>• Use the key <f11> i <f2> to access the PEL Insertion customization buffer to customize these user options.</div><div>⚠️ For yasnippet to work and find your snippets you must identify the directories holding the snippets in the yas-snippet-dirs user-option.</div><div><div>• The name of the directories identified in the list must have the name snippets and must hold sub-directories that are named after the major mode. These sub-directories hold the snippet files that will become available in the major mode (and their derived modes).</div><div>• Type <f11> y <f3> to open the yasnippet customization buffer and access that user-option.</div></div><div>When the yasnippet-mode is enabled, the list of snippets available in the current buffer is:</div><div><div>• listed in the menu bar Yasnippet menu entry (see Menus) and</div><div>• can also be listed using the yas-describe-tables command (which PEL binds to <f11> y t).</div></div><div>PEL binds the following yasnippet commands to keys in the pel: key prefix, shown below.</div></div></div>											
Customize PEL yasnippet use	<f11> y <f2>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL Yasnippet text insertion support. <div><div>• If OTHER-WINDOW is non-nil (use C-u) , display in other window.</div></div>									
Customize Emacs yasnippet control	<f11> y <f3>	(pel-customize-library &optional OTHER-WINDOW)	Customize Yasnippet groups: yasnippet, yasnippet-snippets, yas-minor									
Prints Yasnippet version info	<f11> y ?	(yas-about)	Prints yasnippet version information in the mini buffer.									
Toggle YASnippet minor mode on/off	<f11> y y	(yas-minor-mode &optional ARG)	Toggle YaSnippet mode.									
	<div><div><div>• When YASnippet mode is enabled, ‘yas-expand’, normally bound to the TAB key, expands snippets of code depending on the major mode.</div><div>• With no argument, this command toggles the mode. Positive prefix argument turns on the mode. Negative prefix argument turns off the mode.</div><div>• YASnippet mode key bindings:</div></div><table><tr><td>key</td><td>binding</td></tr><tr><td>-----</td><td>-----</td></tr><tr><td>C-c & C-n</td><td>yas-new-snippet</td></tr><tr><td>C-c & C-s</td><td>yas-insert-snippet</td></tr><tr><td>C-c & C-v</td><td>yas-visit-snippet-file</td></tr></table></div>			key	binding	-----	-----	C-c & C-n	yas-new-snippet	C-c & C-s	yas-insert-snippet	C-c & C-v
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. <div><div>• With prefix ARG, enable Yas-Global mode if ARG is positive; otherwise, disable it.</div></div>									
Display all snippets for current major mode	<f11> y t	(yas-describe-tables &optional WITH-NONACTIVE)	Display snippets for each table. Each table is associated with a major mode (and all the major-modes that derived from it). The tables show the group, the state and name, the abbreviation key(s) and bindings (if any). The names are button that open the actual snippet file.									
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. <div><div>• This key binding is only active when the YASnippet mode is active. Once the snippet was expanded the TAB key normal behaviour is restored.</div></div>									
Write a new snippet	<div><div>• <f11> y n</div><div>• C-c & C-n</div></div>	(yas-new-snippet &optional NO-TEMPLATE)	Pops a new buffer for writing a snippet. <div><div>• Expands a snippet-writing snippet, unless the optional prefix arg NO-TEMPLATE is non-nil.</div></div>									
Prompt for snippet & insert it	<div><div>• <f11> y s</div><div>• C-c & C-s</div></div>	(yas-insert-snippet &optional NO-CONDITION)	Choose a snippet to expand, pop-up a list of choices according to ‘yas-prompt-functions’. <div><div>• With prefix argument NO-CONDITION, bypass filtering of snippets by condition.</div></div>									
Visit a snippet file	<div><div>• <f11> y v</div><div>• C-c & C-v</div></div>	(yas-visit-snippet-file)	Choose a snippet to edit, selection like ‘yas-insert-snippet’. <div><div>• Only success if selected snippet was loaded from a file. Put the visited file in ‘snippet-mode’.</div></div>									

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.