↖ Topic Index

Text Modes

Operation	Keystroke	Function	Note

Text Modes
○ Help & Customize
• Show state of text modes
• Toggle special text modes
○ smart-dash mode
• text whitespace modes
• text enriched mode
• Drawing in ASCII
○ References
	Emacs support navigation and also search for words and symbols, and the concept of "words" can be modified to include or exclude underscores and hyphens. It supports the following special mode:
	• **superword-mode** treats words separated by hyphen and underscores as single entity. Useful for programming languages using snake_case like C, C++, Erlang.
	• **subword-mode** treats sections of camelCase and PascalCase as distinct words. Useful when editing portions of these longer symbols.
	  • As of Emacs 23.2 the CC Mode c-subword-mode is obsolete and has been replaced by the more general subword-mode.
	• **glasses-mode** transforms the way unreadableSymbolsLikeThisOneUsingCamelCase popular in some programming languages into something that some people find more visually pleasing like unreadable_Symbol_like_This_One_Using_Camel_Case using underscores *virtual* separators. You can use other characters as separators through customization of the glasses customization group.
	• **smart-dash-mode** (see smart-dash-mode section below) where the typed dash character may be replaced by underscore depending on the context.
	
	🎛 PEL provides the ability to activate these modes automatically for various major modes by identifying the major modes in the following user option lists:
	  • **pel-modes-activating-superword-mode** **pel-modes-activating-subword-mode** **pel-modes-activating-glasses-mode**
	  • **pel-modes-activating-whitespace-mode** **pel-modes-activating-smart-dash-mode** **pel-modes-activating-electric-quote-local-mode**
	
	Emacs also has:
	• Several modes that deal with whitespace.
	• A prettify-symbol-mode where some strings can be replaced by single symbol character.
	• Drawing modes that allow you to draw in text and navigate over *'void' space*.
	  • Normally you cannot place the cursor after the end of a line. With drawing modes you can place the cursor (point) anywhere in the window.
	• Different way of handling the concept of marking text in a buffer.
	The commands to control all those are listed in this table, starting with the commands to get help in Emacs and customize these features.		

Last updated on:	2026-04-23	See also: Σ **Search/Replace**, Σ **Whitespace**. For drawing, see also: Ⓜ **Graphviz Dot**, Ⓜ **PlantUML**	
Open this PDF file. See also: Σ **Help/Info**	**<f11> t m <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the Σ **Text Modes** 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.
Σ **Customize Text Mode support** See also: Σ **Inserting Text** • Σ **Filling/Justification**	**<f11> t m <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL cross-reference support: what modes automatically activate superword-mode, subword-mode, auto-fill-mode and smart-dash-mode. Also control keypad special binding. • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Σ **Customize Emacs text modes**	• **<f11> t <f3>** • **<f11> t m <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs text mode related groups: editing-basics, glasses, whitespace.
Σ **Customize Emacs whitespace management**	**<f11> t w <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs handling of **whitespace**.

Show state of text modes
• Are hard tabs used for indentation, tab-width
• electric-quote-mode
• delete-selection-mode
• enriched-mode
• overwrite-mode
• case folding
• subword, superword, glass modes
• visible-mode, smart-dash-mode
• paragraph definition
See also: Σ **Indentation**
	<f11> t m ?	(**pel-show-text-modes**)	Display the state of the various text modes for the current buffer in the mini buffer.
			• When **indent-tabs-mode** is active, Emacs inserts a number of hard tabs and spaces.
			  • 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 for a buffer in the supported major modes.
			`-UUU:---  F1  *scratch*  All  (4,0)  (Lisp Interaction WK Anzu LY Fly/-- ² ElDoc Fill) 12:16 1.42 ----------`
			`- Local newline does align : void  . Automatically activated by modes (<f11> t a <f2>): (c-mode c++-mode sh-mode)`
			`- Local electric-quote-mode: off  , electric-quote-local-mode: not loaded.`
			`- whitespace-mode  : not loaded, show-trailing-whitespace : off  , indicate-empty-lines: off.`
			`- enriched-mode  : not loaded.`
			`- overwrite mode  : off  , delete-selection-mode  : off.`
			`- case-fold-search  : on  , sort-fold-case  : not loaded.`
			`- subword mode  : off  , superword mode  : on  , glass-mode: not loaded.`
			`- visible-mode  : off  , smart-dash-mode  : not loaded.`
			`- Sentences end with 2 space characters.`
			`- paragraph-start  : "^L\\|[  ]*$"`
			`- paragraph-separate: "[  ^L]*$"`

Switch Insert/Overwrite mode **See also:** Σ⌨ **Numkeypad**	• **<Esc><insert>** • **<Esc><kp-0>** • **<f11> t o** • **<f11> `**	(**overwrite-mode** &optional ARG)	Toggles the overwrite mode on/off • With a prefix argument ARG, enable Overwrite mode if ARG is positive, and disable it otherwise. • The **<insert>** key is not available in macOS keyboards where it registers as **<kp-0>** instead, but it's the same key on the numeric keypad: the numeric keypad 0 key. ⚠ If using Emacs on a Linux host accessed through ssh, **<Esc><kp-0>** may not work. Try setting **pel-keypad-esc-0-is-overwrite-mode** to t.
Binary file overwrite only mode. ☝ nhexl mode must be activated to edit in binary.	**<f11> t O**	(**nhexl-overwrite-only-mode** &optional ARG)	Minor mode where text is only overwritten: Insertion/deletion is avoided where possible and replaced by overwriting existing text, if needed with 'nhexl-overwrite-clear-byte'. 📦 Requires the **nhexl-mode** package. 🛠 PEL activates it if **pel-use-nhexl** user option is **t**.
Make Info control text visible/invisible (toggle visible mode)	**<f11> t m v**	(**visible-mode** &optional ARG)	Toggle making all invisible text temporarily visible (Visible mode). With a prefix argument ARG, enable Visible mode if ARG is positive, and disable it otherwise. • Useful for editing info files, where some characters are not visible by default. • Same in Org-mode (for example to show everything, or to show the syntax of links without expanding anything).
Toggle subword-mode See also: Σ **Search/Replace**	• **<f11> t m b**	(**subword-mode** &optional ARG)	Toggle subword-mode: a minor mode that treats sections of camelCase and PascalCase as distinct words. With a ARG **>** 0: enable subword-mode mode , ARG **<** 0 : disable it.
	• **<f12> M-b** • **M-<f12> M-b**		• PEL provides the **<f12> M-b** key for the programming language modes where camelCase and PascalCase are popular.
Toggle superword-mode See also: Σ **Search/Replace**	• **<f11> t m p**	(**superword-mode** &optional ARG)	Toggle superword-mode: a minor mode that treats snake_case as one word. In Lisp, '-' and '_' are treated part of words. With a ARG **>** 0: enable superword mode , ARG **<** 0 : disable it. • It impacts word operations such as forward-word **M-f**, backward-word **M-b**, transpose-words **M-t**.
	• **<f12> M-p** • **M-<f12> M-p**	For convenience, PEL provides the **<f12> M-p** key for the programming language modes where snake_case is popular (Emacs Lisp, C, C++, Erlang, Python, etc…).	

Toggle glasses-mode
 	
• See Making CamelCase Readable with Glasses-Mode
	• **<f11> t m g**	(**glasses-mode** &optional ARG	Minor mode for making identifiers likeThisLongCamelCaseName readable.
			• With a prefix argument ARG, enable the mode if ARG is positive, and disable it otherwise.
			• When glasses-mode is active, it tries to add virtual separators (like underscores) at places they belong to.
			  • It does **not** changes the content of the buffer, just the way the text looks. However, if you type the separator, it **is** inserted into the buffer.
			  • By default it uses underscore as separator. It can be modified through customization of the user-options in the **glasses** customization group.

Toggle sentence separators between 1 or 2 spaces	**<f11> t m s**	(**pel-toggle-sentence-end**)	Toggle definition of end of sentence between 2 and 1 space character (to help text filling). This has an impact on the commands that deal with sentences (navigation such as (backward-sentence) and (forward-sentence), kill such as (kill-sentence) and (backward-kill-sentence).
Toggle local electric quote mode	**<f11> t m '**	(**electric-quote-local-mode** &optional ARG)	Toggle 'electric-quote-mode' only in this buffer. Useful to insert nicer-looking quote characters. • To toggle the mode inside all buffers use: **M-x** electric-quote-mode
Toggle prettify-symbols-mode More info: here (on SO) and here (on Reddit).	**<f11> t m y**	(**prettify-symbols-mode** &optional ARG)	Toggle Prettify Symbols mode. • When Prettify Symbols mode and font-locking are enabled, symbols are prettified (displayed as composed characters) according to the rules in 'prettify-symbols-alist' (which see), which are locally defined by major modes supporting prettifying. To add further customizations for a given major mode, you can modify 'prettify-symbols-alist'.
Mark and Region See also: Σ **Cut & Paste** Σ **Marking**	With most editors when you type over a "selected" region, the text in the selection is automatically replaced by the new text. By default Emacs does not behave like this; instead it allows typing text while there is an active a marked region. If you want Emacs behave like other editors and automatically replace the text activate the "*delete-selection-mode*" with the following command.		
Toggles delete selection mode	**<f11> t m d**	(**delete-selection-mode**)	Toggles delete selection-mode on/off. In delete-selection-mode typing a character while a region is active replaces the entire region with what is typed. By default delete selection-mode is off.

Operation	Keystroke	Function	Note
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**. 📦 Requires the **smart-dash** external package. PEL activates it when **pel-use-smart-dash** is set to **t**. To activate smart-dash-mode automatically: • 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.
Toggle smart-dash mode See also: • **Numkeypad** • **Inserting Text** • **Mode Line**	`<f11> i -`	(**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. ⚠️ Normally when smart-dash-mode is active the numeric dash key (**`<kp-subtract>`**) acts as a smart-dash only. ☝ 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: • In **Numlock OFF**: • 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 • In **Numlock ON**: • 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 For more information on the NumLock control and key support, see **Numkeypad**. ☝ With PEL type `<f11> t m ?` to display the status of text modes including dash-mode. ☝ 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.
Text Whitespace Modes			The following Emacs command control how whitespace is shown or hidden. The following commands are also described in the **Whitespace** table. The whitespace mode can be used to highlight: space characters, hard tabs, end of line and lines that are too long.
Toggle Whitespace Mode See also: **Whitespace**	• `<f11> t m w` • `<f11> t w m`	(**whitespace-mode** &optional ARG)	Toggle whitespace visualization (Whitespace mode). ARG>0: enable, ARG<0: disable. The kind of whitespace visualized is determined by the list variable ***whitespace-style***, ***whitespace-newline***.
Hide/Show trailing whitespaces	`<f11> t w T`	(**pel-toggle-show-trailing-whitespace**)	Toggle highlight of the trailing whitespaces in current buffer. • Toggles the value of the variable ***show-trailing-whitespace***.
Hide/Show trailing empty lines	`<f11> t w e`	(**pel-toggle-indicate-empty-lines**)	Toggle highlight of empty lines. • Toggles the value of the variable ***indicate-empty-lines***.
Toggle individual elements of whitespace-style See also: **Whitespace**	`<f11> t w o`	(**whitespace-toggle-options** ARG)	• If local whitespace-mode is off, toggle the ARG option and turn on local whitespace-mode. • If local whitespace-mode is on, toggle ARG option and restart local whitespace-mode. The argument, which is a single character and must be typed following the `<f11> t w o`, can be: • **`f`** toggle face visualization • **`s`** toggle SPACE and HARD SPACE visualization • **`l`** toggle "long lines" visualization • **`n`** toggle NEWLINE visualization • **`C-i`** toggle indentation SPACEs visualization (via 'indent-tabs-mode') • **`I`** toggle indentation SPACEs visualization • **`C-t`** toggle big indentation visualization • **`A`** toggle SPACEs after TAB: SPACEs visualization • **`C-b`** toggle SPACEs before TAB visualization (via 'indent-tabs-mode') • **`B`** toggle SPACEs before TAB: SPACEs visualization • **`?`** Show the above list of options. **`t`** toggle TAB visualization **`r`** toggle trailing blanks visualization **`L`** toggle "long lines" tail visualization **`e`** toggle empty line at bob and/or eob visualization **`i`** toggle indentation TABs visualization **`C-a`** toggle SPACEs after TAB visualization (via 'indent-tabs-mode') **`a`** toggle SPACEs after TAB: TABs visualization **`b`** toggle SPACEs before TAB: TABs visualization
Enriched Mode See also: • **Enriched Text** • **Filling/Justification** Use it to store coloured logs!			The enriched-mode is a buffer-local minor mode. When active text properties are used to modify the way the buffer renders text. • When saving a buffer with enriched-mode to a file, the is saved in 'text/enriched' format. Several text properties are saved to the file, including the fonts, colours, indentation and justification. If Emacs opens the file later it can automatically re-active the enriched-mode to display these properties. • If Emacs fails to recognize that the file is in 'text/enriched' format, execute **`M-x format-decode-buffer`**. ☝💡 This can be quite useful to store the content of a log file that uses different fonts and colours to indicate commands and errors in a file. Later you can re-open the file with Emacs and see the same fonts and colours!
Toggle enriched text mode	• `<f11> t e e` • `<f11> t m e`	(**enriched-mode** &optional ARG)	Toggle the enriched-text minor mode for editing text/enriched files. • These are files with embedded formatting information in MIME standard text/enriched format. • With a prefix argument ARG, enable the mode if ARG is positive, and disable it otherwise. If called from Lisp, enable the mode if ARG is omitted or nil. • Turning the mode on or off runs 'enriched-mode-hook'.
GNU Screen Log File			GNU Screen log files contain ANSI escape codes normally interpreted by the shell but not by Emacs. Use the following command to render the codes.
Fix the rendering of the log file created by GNU Screen See also: • **Buffers** • **Shells/Terminals Comparisons** for info on **GNU Screen**	`<f11> t s`	(**pel-screen-log-fix-rendering**)	Fix rendering of buffer created by the GNU Screen log. • It converts the marked area of a buffer, if it is marked, otherwise it processes the entire or the narrowed portion of the buffer. It renders the escape codes, converts the line endings to Unix-style line endings. • In some cases the log file created by GNU Screen injects one extra line per line. If the '**pel-screen-log-delete-all-consecutive-blank-lines**' user-option is set, the function removes them. • Some artifact main remain after these transformations. To fix them automatically, identify a set of regular expression/replacement string pairs in the '**pel-screen-log-fix-regexp-pairs**' user-option. ⚠️ This removes the escape codes from the buffer but renders them using color and other attributes. If you save the file these will no longer be visible unless you first activates the enriched text mode in the buffer to encode the text attributes in a way Emacs will be able to interpret later.
Drawing ASCII in Emacs See also: **Graphviz Dot** **PlantUML**			Emacs provides the picture-mode and artist-mode to draw ASCII-based pictures. Both are available when Emacs runs in graphics and terminal mode. However, I have not been able to use artist-mode with the mouse, even with xterm-mouse-mode active: each click just prints an ANSI sequence code. Two modes are available: 1) Artist Mode 2) Picture Mode ☝ The Picture mode can be very useful to type text in vertical fashion when for example, writing **reStructuredText** table or writing code in tabular fashion.
Artist Mode			Although you can get some commands to work in terminal mode, it's best to use artist-mode when running Emacs in graphics mode. See **demo**.
Toggle artist mode See also: **Drawing**	`<f11> D a`	(**artist-mode** &optional ARG)	Toggle Artist mode. • With argument ARG, turn Artist mode on if ARG is positive. • Artist lets you draw lines, squares, rectangles and poly-lines, ellipses and circles with your mouse and/or keyboard.
Picture Mode			Emacs supports the picture mode that allow you to move your cursor freely anywhere inside the window, which greatly simplify creating rectangular shapes for tables or even *drawing* ASCII-art. This work well in both graphics and terminal mode.
Enter picture mode See also: **Drawing**	• `<f11> D p` • `<f11> t p`	(**picture-mode**)	Switch to Picture mode, in which a quarter-plane screen model is used. ☝ Very useful to type text in vertical fashion when for example, writing **reStructuredText** table. • Type **`C-c C-c`** to exit picture-mode and return to the mode previously used in the buffer.

Text Modes — References

Topic & Link	Notes
GNU Emacs Manual: Text - Words	
GNU Emacs Manual: Text - Sentences	
GNU Emacs Manual: Text - Paragraphs	
GNU Emacs Manual: Text - Quotation Marks	
GNU Emacs Manual: Modes - Minor Modes	
GNU Emacs Manual: Programs - Other Features Useful for Editing Programs	
GNU Info Manual: Getting Started - Invisible text in Emacs Info	