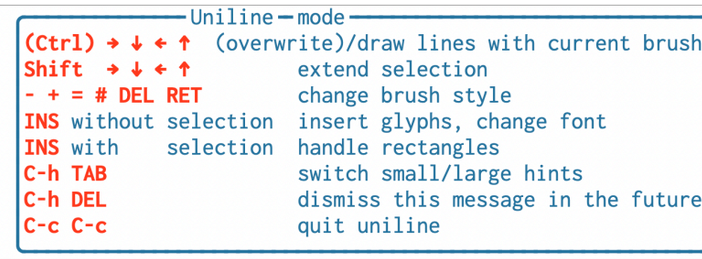
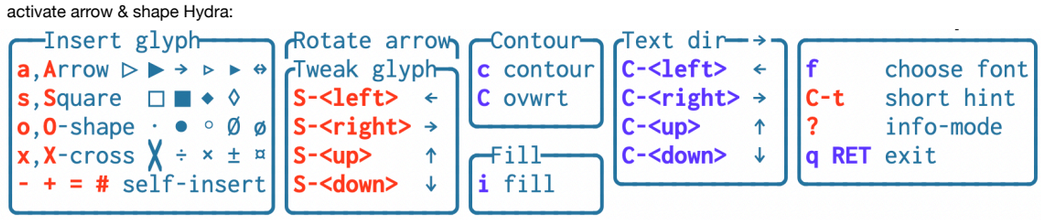
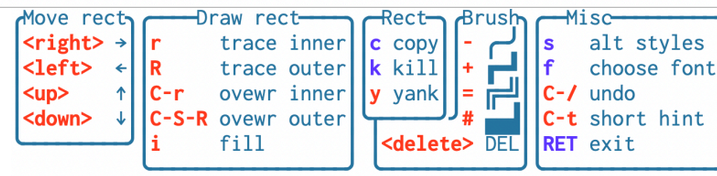
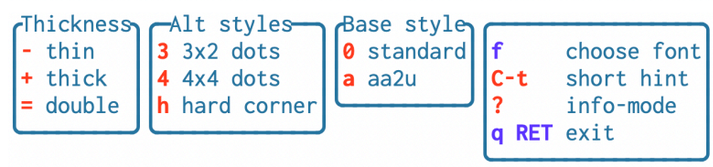
↖ Topic Index

Drawing Text in Emacs

Description	Keystroke	Function	Note
Drawing Text Emacs ○ **Help & Customize** • syntree • artist mode • **picture mode** ○ picture-mode rectangle • edit tabular data • uniline ○ **Reference**	Emacs provides the **picture-mode** and **artist-mode** to draw ASCII-based pictures. Both are available when Emacs runs in graphics and terminal mode. ☝These 2 modes work better in GUI modes as you can use the mouse to draw. Attempt to draw with. the mouse inside terminal-based Emacs failed. PEL also provides support for the following external packages, activated by the corresponding pet-use customizable user-option.		
	📦 **syntree**	**pel-use-syntree**	A major mode where you input a tree in S-expression form which generates a printed tree output. MAde for linguists but can be used to draw any kind of tree.
	📦 **uniline**	**pel-use-uniline**	Draw ▶—UNICODE diagrams—◀ within ▶—your texts—◀ in Emacs ☝Excellent in terminal
	📦 **ascii-art-to-unicode**	**pel-use-ascii-art-to-unicode** (activated when **pel-use-uniline** is **t**)	Convert ASCII drawings to Unicode. Can be used via **uniline** .
Last updated on:	2025-11-22		
Open this PDF file. See also: **Help/Info**	**<f11> D <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **Drawing** 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 Drawing support. See also: **Customize**	**<f11> D <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL drawing mode support. • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Customize Emacs Drawing control **See also: Customize**	**<f11> D <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs support for the display packages: artist, picture, syntree, online.
Create a new syntree diagram	**<f11> D s**	(**syntree-new**)	Create new frame with input and output buffers for syntree: Printing a tree from writing S-expression. See **syntree manual**. • Type **C-h** m to list the bindings of this major mode. 📦 Requires **syntree** activated by **pel-use-syntree**
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 but it can be used in terminal mode for simple things. See this **Youtube video on using Emacs Artist mode**.		
Toggle artist mode	**<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. ☝Very useful to type text in vertical fashion when for example, writing reStructuredText table.		
Enter picture mode See also: **Text Modes**	• **<f11> D p** • **<f11> t p**	(**picture-mode**)	Switch to Picture mode, in which a quarter-plane screen model is used. • Type **C-c C-c** to exit picture-mode and return to the mode previously used.
Picture Mode Commands	While in picture mode the following commands help type text in ways that help "*drawing*" text. It's possible, for example to type text vertically going down or going up or horizontally toward the left (without changing the input mode). This is very useful to type rectangular shapes that can be used to make UML drawings or just tables for reStructuredText markup for example. Or just to create vertically lined-up comments. To get get the full list of commands, simply type **<f1> m** as it is the case for all mode.		
Picture Motion	The following 12 commands set the direction of insertion. As long as you stay in picture mode and don't issue another of these commands insertions continue in that direction. The direction is displayed in the mode line.		
Move left	• **C-c <** • **C-c <left>**	(**picture-movement-left**)	Move left after self-inserting character in Picture mode. With PEL when ***pel-use-winner*** user option is **t** the **C-c <left>** is used by winner and is not available for picture movement.
Move right	• **C-c >** • **C-c <right>**	(**picture-movement-right**)	Move right after self-inserting character in Picture mode. With PEL when ***pel-use-winner*** user option is **t** the **C-c <right>** is used by winner and is not available for picture movement.
Move up	• **C-c ^** • **C-c <up>**	(**picture-movement-up**)	Move up after self-inserting character in Picture mode.
Move down	• **C-c .** • **C-c <down>**	(**picture-movement-down**)	Move down after self-inserting character in Picture mode.
Move northwest (nw)	**C-c `**	(**picture-movement-nw** &optional ARG)	Move up and left after self-inserting character in Picture mode.
Move northeast (ne)	**C-c '**	(**picture-movement-ne** &optional ARG)	Move up and right after self-inserting character in Picture mode.
Move southwest (sw)	**C-c /**	(**picture-movement-sw** &optional ARG)	Move down and left after self-inserting character in Picture mode.
Move southeast (se)	**C-c **	(**picture-movement-se** &optional ARG)	Move down and right after self-inserting character in Picture mode.
Move westnorthwest (wnw)	**C-u C-c `**	(**picture-movement-nw** &optional **ARG**)	Move up and two-column left after self-inserting character in Picture mode.
Move eastnortheast (ene)	**C-u C-c '**	(**picture-movement-ne** &optional **ARG**)	Move up and two-column right after self-inserting character in Picture mode.
Move westsouthwest (wsw)	**C-u C-c /**	(**picture-movement-sw** &optional **ARG**)	Move down and two-column left after self-inserting character in Picture mode.
Move eastsouthwest (ese)	**C-u C-c **	(**picture-movement-se** &optional **ARG**)	Move down and two-column right after self-inserting character in Picture mode.
Move in Picture Mode	The following commands move the point freely, even in "*void*" space, past the end of the current line or past the last line in the buffer, extending the whitespace as necessary and converting hard tabs to spaces when necessary. ☝ These commands override standard navigation motion commands, but other available navigation commands in described in **Navigation**.		
Move up	• **C-p** • **<up>**	(**picture-move-up** ARG)	Move vertically up, making whitespace if necessary. • With argument, move that many lines.
Move down	• **C-n** • **<down>**	(**picture-move-down** ARG)	Move vertically down, making whitespace if necessary. • With argument, move that many lines.
Move to column following last non-whitespace character	**C-e**	(**picture-end-of-line** &optional ARG)	Position point after last non-blank character on current line. • With ARG not nil, move forward ARG - 1 lines first. • If scan reaches end of buffer, stop there without error.
Move right	• **C-f** • **<right>**	(**picture-forward-column** ARG &optional INTERACTIVE)	Move cursor right, making whitespace if necessary. • With argument, move that many columns.
Move left	• **C-b** • **<left>**	(**picture-backward-column** ARG &optional INTERACTIVE)	Move cursor left, making whitespace if necessary. • With argument, move that many columns.
Move in direction of current picture motion	**C-c C-f**	(**picture-motion** ARG)	Move point in direction of current picture motion in Picture mode. • With ARG do it that many times. Useful for delineating rectangles in conjunction with diagonal picture motion.
Move in direction opposite to current picture motion	**C-c C-b**	(**picture-motion-reverse** ARG)	Move point in direction opposite of current picture motion in Picture mode. • With ARG do it that many times. Useful for delineating rectangles in conjunction with diagonal picture motion.

Description	Keystroke	Function	Note
Picture Mode Rectangle	The following commands allow drawing rectangles in the buffer as well as copy & kill them as storing/restoring to/from registers.		
Draw rectangle around region	`C-c C-r`	(**picture-draw-rectangle** START END)	Draw a rectangle around region.
Clear & save rectangle	`C-c C-k`	(**picture-clear-rectangle** START END &optional KILLP)	Clear and save rectangle delineated by point and mark. • The rectangle is saved for yanking by C-c C-y and replaced with whitespace. The previously saved rectangle, if any, is lost. • With prefix argument, the rectangle is actually killed, shifting remaining text.
Clear reactangle	`C-c C-w`	(**picture-clear-rectangle-to-register** START END REGISTER &optional KILLP)	Clear rectangle delineated by point and mark into REGISTER. • The rectangle is saved in REGISTER and replaced with whitespace. • With prefix argument, the rectangle is actually killed, shifting remaining text.
Yank and overlay saved rectangle	`C-c C-y`	(**picture-yank-rectangle** &optional INSERTP)	Overlay rectangle saved by C-c C-k • The rectangle is positioned with upper left corner at point, overwriting existing text. • With prefix argument, the rectangle is inserted instead, shifting existing text. • Leaves mark at one corner of rectangle and point at the other (diagonally opposed) corner.
Overlay rectangle saved in register	`C-c C-x`	(**picture-yank-rectangle-from-register** REGISTER &optional INSERTP)	Overlay rectangle saved in REGISTER. • The rectangle is positioned with upper left corner at point, overwriting existing text. • With prefix argument, the rectangle is inserted instead, shifting existing text. • Leaves mark at one corner and point at the other (diagonally opposed) corner.
Edit Tabular Data	The following commands help manage tabular data using tab.		
Move to next tab stop	`<tab>`	(**picture-tab** &optional ARG)	Tab transparently (just move point) to next tab stop. • With prefix arg, overwrite the traversed text with spaces. • The tab stop list can be changed by C-c TAB and M-x edit-tab-stops. • See also documentation for variable 'picture-tab-chars'.
Set tab stops according to context of current line	`C-c <tab>`	(**picture-set-tab-stops** &optional ARG)	Set value of 'tab-stop-list' according to context of this line. • This controls the behavior of TAB. A tab stop is set at every column occupied by an "interesting character" that is preceded by whitespace. • Interesting characters are defined by the variable 'picture-tab-chars', see its documentation for an example of usage. • With ARG, just (re)set 'tab-stop-list' to its default value. • The tab stops computed are displayed in the minibuffer with ':' at each stop.
Delete backward	`<del>`	(**picture-backward-clear-column** ARG)	Clear out ARG columns before point, moving back over them.
Kill rest of line	• `C-k` • `<f11> - e`	(**picture-clear-line** ARG)	Clear out rest of line; if at end of line, advance to next line. • Cleared-out line text goes into the kill ring, as do newlines that are advanced over. • With argument, clear out (and save in kill ring) that many lines.
Insert new line, leave point before it See also: **Whitespace**	`C-o`	(**open-line** N)	Insert a newline and leave point before it. • If there is a fill prefix and/or a 'left-margin', insert them on the new line if the line would have been blank. • With arg N, insert N newlines.
Draw Lines with Unicode characters. ★★	Requires **uniline** **pel-use-uniline** activates it. This also requires and activates the **hydra** external package.		
Toggle uniline mode	`<f11> D u`	(**uniline-mode** &optional ARG)	Toggle minor mode to draw lines, boxes, & arrows using UNICODE characters.
Exit uniline mode	• `C-c C-c` • `<f11> D u`		
uniline mode drawing keys	The following keys are available to draw lines while the uniline mode is active. Use control arrows to overwrite an existing line with another.		
uniline mode insertion keys The **uniline mode insert key** used to insert the following characters is identified by the **uniline-key-insert** customizable user-option. • PEL adds the `<f8>` key to this list when the mode is invoked because the default keys are not available on some keyboards.	• `<insertchar>` • `<insert>` • `<f8>`	(**uniline-launch-interface**)	Choose between two Hydras based on selection. • When selection is active, most likely user wants to act on a rectangle. Therefore the rectangle hydra is launched. • Otherwise, the arrows & shapes hydra is invoked.
	Selection **not** active:	activate arrow & shape Hydra:	
	Selection is **active**:	activate the rectangle Hydra:	
	With the rectangle Hydra: • pressing **s** opens the alternate style Hydra • that allows changing the style of the rectangle. • it also provides running the **ascii-art-to-unicode** command which translate an ASCII drawing to Unicode. • PEL automatically requests installation of **ascii-art-to-unicode** when **pel-use-uniline** is **t**		
	• pressing **f** opens the font Hydra • In GUI mode, that allows to change the font used to display the drawing, mostly for testing when a font might not render the drawing properly.		

Drawing — References

Topic & Link	Notes
Poor Man's UML / Emacs Artist Mode and Ditaa Demo - Youtube video	Video demo of Emacs artist mode. Shows how to draw UML diagram.