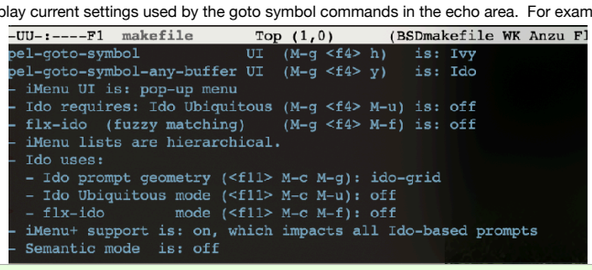
↖ Topic Index

Emacs support for Make Files

Description	Keystroke	Function	Note
Make support Last updated on:	• Emacs natively supports several Make dialect modes as listed below. • PEL adds several commands and user-options that add control to the editing behaviour. See: • **pel-modes-activating-superword-mode** : PEL automatically activates super-word-mode for make files. Use **<f11> t <f2>** to access the customization group. See language specific info: **GNU Make** 2025-12-18		
Open this PDF file. See also: **Help/Info**	**<f11> SPC M <f1>** **<f12> <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **GNU Make** 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 make support	**<f11> SPC M <f2>** **<f12> <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL make support: **pel-use-makefile** • **pel-make-mode-alist** to identify more file regexp and a make file major mode that must be used for those files. • **pel-makefile-activates-minor-modes** lists minor modes to automatically activate in makefile major modes. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
GNU Make	**<f11> SPC M <f3>** **<f12> <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs makefile support: makefile. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Select Make dialect mode See also: • **Customize** • **File/Directory Variables**	Emacs supports several dialects of **make**. It automatically selects the dialect when a file is visited using the mode and file specification association identified in the **auto-mode-alist** variable. The support associates the name and extensions of most make files with the corresponding dialect mode. The following make file dialect modes are supported: • makefile-mode (the based mode upon which all following modes are derived): • makefile-automake-mode : .am • makefile-bsdmake-mode : [Mm]akefile, .mk, .make • makefile-gmake-mode : GNUmakefile • makefile-imake-mode : Imakefile • makefile-makepp-mode : .makepp • makefile-nmake-mode : .mak PEL implements the makefile-nmake-mode to support Microsoft NMAKE syntax. • Some projects use the .mak extension for their makefile (the **dmd project** for example). • With PEL, set up the association using the **pel-auto-mode-alist** user-option. • You can access the relevant customization buffer for this user-option by using PEL **<f11> <f2> p** key sequence. See **Customize** • Its also possible to use file variables to explicitly identify the make dialect mode: write something like this on the first line: `-*- mode: makefile-gmake; -*-` • You can also use the following commands to manually activate one of these modes when on of them is already active.		
Activate automake mode	• **C-c RET C-a** • **C-c C-m C-a**	(**makefile-automake-mode**)	Activates the **automake** mode • The mode-line lighter is : Makefile.am
Activate BSD make mode	• **C-c RET C-b** • **C-c C-m C-b**	(**makefile-bsdmake-mode**)	Activates the **BSD make** mode. • BSD Make is the default make on macOS and BSD OS systems. • The mode-line lighter is : BSDmakefile
Activate GNU make mode	• **C-c RET C-g** • **C-c C-m C-g**	(**makefile-gmake-mode**)	Activates the **GNU make** mode. • The mode-line lighter is : GNUmakefile ⚠ Because this key sequence ends with **C-g**, type the **Esc** key 3 times to escape from the C-c C-m prefix. You can also use a key not in the list.
Activate imake mode	• **C-c RET <tab>** • **C-c C-m C-i**	(**makefile-imake-mode**)	Activate the **imake** mode • The mode-line lighter is : Imakefile
Activate standard make mode	• **C-c RET RET** • **C-c C-m C-m**	(**makefile-mode**)	Activates the major mode for editing standard Makefiles. • The mode-line lighter is : Makefile
Activate makepp mode	• **C-c RET C-p** • **C-c C-m C-p**	(**makefile-makepp-mode**)	Activates the **makepp** mode. Also called **make++** • makepp is written in Perl. It is mostly useful for writing C++ specific make files, as it expands GNU Make and removes the requirement of using recursive make. • The mode-line lighter is : Makeppfile
Activate NMAKE mode	• **C-c RET C-n** • **C-c C-m C-n**	(**makefile-nmake-mode**)	Activates the nmake mode, supporting Microsoft's NMAKE makefile syntax. • The mode-line lighter is: Nmake
Navigate	The standard Emacs make-mode.el package provides the 2 commands to navigate across make target/dependency statements. PEL complements this with commands to navigate across the macro definition statements.		
beginning of next token See also: **Navigation**	**C-<right>**	(**pel-forward-token-start** &optional N)	Move to the beginning of next word/symbol. • It handles characters that may be part of symbol in the current major mode, and jumps over them but stops at whitespace and operators. • Supports numerical argument for repetition. Negative argument reverses the movement direction. The command support shift-marking. • ☝ Useful when the superword-mode is not activated: allows jumping to next symbol while the word commands stop at each word separator character.
beginning of previous token See also: **Navigation**	**C-<left>**	(**pel-backward-token-start** &optional N)	Move to the beginning of previous word/symbol. • It handles characters that may be part of symbol in the current major mode (like '_' in C), and jumps over them but stops at whitespace and operators. • Supports numerical argument for repetition. Negative argument reverses the movement direction. The command support shift-marking. ☝ Useful when the superword-mode is not activated: allows jumping to previous symbol while the word commands stop at each word separator character.
Move point forward to next target/ dependency	• **M-n** • **<f12> <down>** • **M-<f12> <down>** **<f11> SPC M <down>**	(**makefile-next-dependency**)	Move point to the beginning of the next dependency line. • Skips comments and macro definitions.
Move point backward to previous target/ dependency	• **M-p** • **<f12> <up>** • **M-<f12> <up>** **<f11> SPC M <up>**	(**makefile-previous-dependency**)	Move point to the beginning of the previous dependency line. • Skips comments and macro definitions.
Move point forward to next macro definition statement	• **<f12> M-<down>** • **M-<f12> M-<down>** **<f11> SPC M M-<down>**	(**pel-make-next-macro** &optional N SILENT DONT-PUSH-MARK	Move to the beginning of next N make file macro definition statement. • The function skips over comments. • If no valid form is found, don't move point, issue an error describing the failure unless SILENT is non-nil, in which case the function returns nil on error and non-nil on success. • The error message states the number of instanced searched, the regexp used and the number of instances found. • On success, the function push original position on the mark ring unless DONT-PUSH-MARK is non-nil. The command support shift-marking.
Move point backward to previous macro definition statement	• **<f12> M-<up>** • **M-<f12> M-<up>** **<f11> SPC M M-<up>**	(**pel-make-previous-macro** &optional N SILENT DONT-PUSH-MARK)	Move to the beginning of previous N make file macro definition statement. • The function skips over comments. • If no valid form is found, don't move point, issue an error describing the failure unless SILENT is non-nil, in which case the function returns nil on error and non-nil on success. • The error message states the number of instanced searched, the regexp used and the number of instances found. • On success, the function push original position on the mark ring unless DONT-PUSH-MARK is non-nil. The command support shift-marking.
• **If statements**	Use the <f6> key prefix followed by **<right>**, **<left>**, **<up>** and **<down>** to navigate across GNU Make if statements. The first 2 also accept prefix to move to else.		
Move point forward to matching endif • **or matching else**	**<f6> <right>**	(**pel-make-forward-conditional** &optional TO-ELSE)	Move point forward to matching end of make conditional: if point is before a make conditional if statement it moves to the matching endif, or else when prefix arg is used. • With **C-u** or numerical arg: move backward to matching else. • On success, push the original position on the mark ring and return the new position. On error, issue user error on mismatch. Shift marking is available with **C-M-<right>**
Move point backward to matching if • **or matching else**	**<f6> <left>**	(**pel-make-backward-conditional** &optional TO-ELSE)	Move point backward to matching beginning of make conditional. • With **C-u** or numerical arg: move backward to matching else. • On success, push the original position on the mark ring and return the new position. On error, issue user error on mismatch. Shift marking is available with **C-M-<left>**

Description	Keystroke	Function	Note
Move outward forward to matching endif	**<f6> <down>**	(**pel-make-outward-forward-conditional** &optional NEST-COUNT)	Move point forward, outward to end of current if statement. • By default move 1 nest level outward. A larger count can be specified with optional NEST-COUNT numeric argument. • On success, push the original position on the mark ring and return the new position. On error, issue user error on mismatch.
Move outward backward to matching if	**<f6> <up>**	(**pel-make-outward-backward-conditional** &optional NEST-COUNT)	Move point backward, outward to beginning of current if statement. • By default move 1 nest level outward. A larger count can be specified with optional NEST-COUNT numeric argument. • On success, push the original position on the mark ring and return the new position. On error, issue user error on mismatch.
Show all Make conditional statements inside an occur buffer	**<f6> o**	(**pel-make-conditionals-occur** &optional NLINES)	Show make conditional statements inside an occur buffer. • Each line is shown with NLINES before and after, or -NLINES before if NLINES is negative. • NLINES defaults to **list-matching-lines-default-context-lines** user-option value. • If a region is defined the search is restricted to the region. See **occur search**
• by blocks	Move to the matching pair of character in the following sets: (),[],{},<>,"", ''.		
block backward	• **C-M-b** • **C-M-<left>** • **C-[C-b** • **Esc C-b** • **Esc C-<left>**	(**backward-sexp** &optional ARG)	Move backward across one balanced expression (sexp). • With ARG, do it that many times. Negative arg -N means move forward across N balanced expressions. This command assumes point is not in a string or comment. • **C-M-b** : Shift marking is available in graphics mode, not in terminal mode. • **C-M-<left>** : Shift marking works with this command.
	• ⚠ With PEL: if you want to use **Esc C-<left>** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. ❖ **C-M-<left>** does not work on Windows, but **H-<left>** works. Several Linux distros map **C-M-<left>** to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems->settings->keyboard->shortcuts to prevent it from using that key sequence.		
block forward	• **C-M-f** • **C-M-<right>** • **C-[C-f** • **Esc C-f** • **Esc C-<right>**	(**forward-sexp** &optional ARG)	Move forward across one balanced expression (sexp). • With ARG, do it that many times. Negative arg -N means move backward across N balanced expressions. This command assumes point is not in a string or comment. • **C-M-f** : Shift marking is available in graphics mode, not in terminal mode. • **C-M-<right>** : Shift marking works with this command.
	• ⚠ With PEL: if you want to use **Esc C-<right>** binding you must ensure that **pel-windmove-on-esc-cursor** user option is set to nil. ❖ **C-M-<right>** does not work on Windows, but **H-<right>** does. Several Linux distros map **C-M-<right>** to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems->settings->keyboard->shortcuts to prevent it from using that key sequence.		
iMenu/Speedbar See also: • Σ **Completion/Input** • Σ **Menus** • Σ **Speedbar**	You can navigate through makefile macros and targets (identified as *dependencies*) using Emacs iMenu and Speedbar capabilities. • Several commands are available to get a list of the various elements and move point to it. • These commands include the following. More are listed in the Σ **Completion/Input** . • Several packages extend the completion and how entry is done. PEL allows dynamic selection of several methods and can display the current status with **M-g ?** • You can also use the Σ **Speedbar** to list all items on a vertical side-bar and navigate through them.		
Find definitions using IMenu See also: • Σ **Completion/ Input** • Σ **Menus**	• **<f11> <f10> i** • **M-g i** • **M-g M-i**	(**imenu** INDEX-ITEM)	Lists imenu-detected items from the current buffer (according to its major mode).
		• For example, in a elisp file, the entry points are the function definitions and may include the variables and other items depending what function does the parsing (it can be semantic which provides more information). Provides one of the following interfaces to let user select entry to jump to: • The default: input completion, using the minibuffer window and tab completion. • a pop-up window : available in Graphics mode selected by mouse or in both graphics and terminal (TTY) modes when the **imenu-use-popup-menu** user-option is turned on. • with PEL you can use **pel-imenu-toggle-popup** (bound to **M-g <f4> p**) to toggle the user interface used by **imenu**.
Move to imenu detected symbol definition in current buffer ★★	• **M-g h** • **M-g M-h**	(**pel-goto-symbol**)	Prompt using for imenu symbol of the current buffer and move point to it. • Refresh imenu and jump to a place in the buffer using the completion method selected. • Modify user interface currently used with **M-g <f4> h**. • The command sets a ref-marker before moving. Return to previous location with **M-,**
Display current setting of commands: • **pel-goto-symbol** • **pel-goto-symbol-any-buffer** See also: • Σ **Completion/ Input**	**M-g ?**	(**pel-show-goto-symbol-settings**)	Display current settings used by the goto symbol commands in the echo area. For example: 
Insert & Edit	The following commands help the editing of the makefile contents.		
Insert GNU make function statement	• **C-c Tab** • **C-c C-i**	(**makefile-insert-gmake-function**)	Insert a **GNU make function call**. • Asks for the name of the function to use (with completion). • Then prompts for all required parameters.
Insert target at point	**C-c :**	(**makefile-insert-target-ref** TARGET-NAME)	Complete on a list of known targets, then insert TARGET-NAME at point.
Add/remove line continuation trailing backslashes	**C-c C-**	(**makefile-backslash-region** FROM TO DELETE-FLAG)	Insert, align, or delete end-of-line backslashes on the lines in the region. • With no argument, inserts backslashes and aligns existing backslashes. • With an argument, deletes the backslashes.
	This function does not modify the last line of the region if the region ends right at the start of the following line; it does not modify blank lines at the start of the region. So you can put the region around an entire macro definition and conveniently use this command.		
Perform completion at point	• **C-M-i** • **<f12> C-M-i**	(**completion-at-point**)	Perform completion on the text around point. The completion method is determined by 'completion-at-point-functions'. ⚠ The **C-M-i** is also often bound to flyspell command. Use **<f12> C-M-i** instead.
Electric Insert	When the makefile-mode makefile-electric-keys user-option is turned on (it is off by default), the characters **$: =** and **.** have special behaviour, described below.		
Insert macro reference	**$**	(**makefile-insert-macro-ref** MACRO-NAME)	Complete on a list of known macros, then insert complete ref at point.
Insert new target	**:**	(**makefile-electric-colon** ARG)	Prompt for name of new target. • Only prompts if point is at beginning of line. Anywhere else just self-inserts.
Insert macro defintion	**=**	(**makefile-electric-equal** ARG)	Prompt for name of a macro to insert. • Only prompts if point is at beginning of line. Anywhere else just self-inserts.
Insert special target	**.**	(**makefile-electric-dot** ARG)	Prompt for the name of a special target to insert. Supports tab completion. • Only does electric insertion at beginning of line. Anywhere else just self-inserts.
Indenting	In make file editing, the tab character is important. The make program distinguish the tab character from multiple space characters. ⚠ The **C-M-q** key sequence is bound to prog-indent-sexp but it does not work well in makefile. Use the other 3 commands.		
Insert a tab character	**<tab>**	(**indent-for-tab-command** &optional ARG)	Inserts a tab character in a makefile.
Indent line(s) rigidly	• **<f6> <tab>** • **<f11> <tab> c**	(**pel-indent-lines** &optional N)	Indent current or marked lines by N indentation levels. Each level uses a tab character. • Works with point anywhere on the line.
	• A special argument N can specify more than one indentation level. It defaults to 1. If a negative number is specified, 'pel-unindent-lines' is used. • If a region is marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all characters in all affected lines. Use **C-g** to de-activate the region.		

Description	Keystroke	Function	Note
Un-indent line(s) rigidly	• `<backtab>` • `<f6> <backtab>` • `<f11> <tab> C`	(**pel-unindent-lines** &optional N)	• Un-indent current line or marked lines by N indentation levels. • Works with point is anywhere on the line. • All lines touched by the region are un-indented. • If region was marked, the function does not deactivate it to allow repeated execution of the command. • If a region was marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all characters in all affected lines • Use **C-g** to de-activate the region.
Indent expression	`C-M-q`	(**prog-indent-sexp** &optional DEFUN)	Indent the expression after point. • When interactively called with prefix, indent the enclosing defun instead. ⚠️ This command does not work well in makefiles.
Comment control	Although the make file modes provide the comment-region command, it's best to use comment-dwim as it works much better: • Insert a comment, comment or un-comment a region with `M-;`		
Comment/un-comment See also: **Σ Comments** **With PEL**: Comment the current line with `M-0 M-;`	`M-;`	(**comment-dwim** ARG)	Comment or un-comment line or region.
		(**pel-comment-dwim** ARG)	Same as **comment-dwim** but comments the current line with a numeric ARG or 0.

	• Comment or un-comment line or region.
	  • When no marked region and no comment:
	  • On empty line: insert comment starter at the proper indentation level. Typed again: move it toward end of line.
	  • On line with code: insert comment starter after the code for an end-of-line comment
	  • With marked un-commented region: Comment region (each line is commented)
	  • With marked commented region: Removes the comment.
	• Call the comment command you want (Do What I Mean).
	  • If the region is active and 'transient-mark-mode' is on, call 'comment-region' (unless it only consists of comments, in which case it calls 'uncomment-region'). Else, if the current line is empty, call 'comment-insert-comment-function' if it is defined, otherwise insert a comment and indent it. Else if a prefix ARG is specified, call 'comment-kill'. Else, call 'comment-indent'.		

	`C-c C-c`	(**comment-region** BEG END &optional ARG)	Comment or uncomment each line in the region. ⚠️ Prefer **comment-dwim**: it works better.

	Comment or uncomment each line in the region.
	  • With just C-u prefix arg, uncomment each line in region BEG .. END.
	  • Numeric prefix ARG means use ARG comment characters. If ARG is negative, delete that many comment characters instead.
	• The strings used as comment starts are built from 'comment-start' and 'comment-padding'; the strings used as comment ends are built from 'comment-end' and 'comment-padding'.
	• By default, the 'comment-start' markers are inserted at the current indentation of the region, and comments are terminated on each line (even for syntaxes in which newline does not end the comment and blank lines do not get comments). This can be changed with 'comment-style'.		

Toggle display of comments in buffer or active region See also: **Σ Comments**	`<f11> ; ;`	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see **Σ Comments**). 🗹 PEL activates it when the **pel-use-hide-comnt** user option is **t**.
Analyze	The following commands analyze the content of the make file or the file system.		
Scan current directory files, checking for targets	`C-c C-f`	(**makefile-pickup-filenames-as-targets**)	Scan the current directory for filenames to use as targets. • Checks each filename against 'makefile-ignored-files-in-pickup-regex' and adds all qualifying names to the list of known targets.
Scan current buffer for makefile content	`C-c C-p`	(**makefile-pickup-everything** ARG)	Notice names of all macros and targets in Makefile. • Prefix arg means force pickups to be redone. Use this to refresh the list of macros and targets located in the makefile before executing another action on those.
Update scan with latest makefile buffer content	`C-c C-u`	(**makefile-create-up-to-date-overview**)	Create a buffer containing an overview of the state of all known targets. • Known targets are targets that are explicitly defined in that makefile; in other words, all targets that appear on the left hand side of a dependency in the makefile.
List macros and targets in dedicated buffer	`C-c C-b`	(**makefile-switch-to-browser**)	Open a *Macros and Target* buffer that only lists them. • It operates in Fundamental mode and aside listing the macros and targets provides nothing more.

Emacs & Makefile— References

Document	Notes
Make tools	See also: **GNU Autotools @ Wikipedia**, **GNU Coding Standard**, **section 7**, **Filesystem Hierarchy Standard** (**FHS** 3.0)
GNU Make Manuals	• **GNU Make Top page** • **How to run make** • **GNU Make - Appendix A - Quick Reference** • **Makefile Conventions** • **Autoconf Portable Make Programming**
Makepp home page	Makepp, also called make++ is a GNU Make replacement, written in Perl. It addresses the recursive make problem.
Make generic information	
Recursive Make Considered Harmful - Steve Miller	PDF paper (from the wayback machine archive) written by Steve Miller in 1997 describing the concept of recursive make technique showing why it causes several problems and what can be done to avoid them.
Non-Recursive Make Considered Harmful	A march 2016 PDF paper from Andrey Mokhov, Neil Mitchell, Simon Peyton Jones and Simon Marlow describe how even a non-recursive make based build system can be difficult to maintain and they propose something based on the Shake Haskell library.
Rules of Makefiles	Simple and clear rules to use as a guide for writing good make files.
How Not To Use VPATH	Describe problems to avoid when using VPATH
Multi-Architecture Builds	Describe a proper way to create make files.