↖ Topic Index

Abbreviations/Expansions

Operation	Keystroke	Function	Note

Abbreviations
○ Help & Customization
• Show abbrev settings
• Dynamic expansion
 	
○ Manual Abbrev: abbrev-mode
• toggle manual abbrev on/off
• create abbrev from spelling error
• manual abbreviation expansion
• creating abbreviations
○ edit abbreviations
• save abbreviations
• delete abbreviations
	Emacs supports several text abbreviation expansion mechanisms:
	• A manual mode: the **abbrev** minor mode: you must define abbreviations with matching expansion either explicitly or via special spell checking command.
	• Dynamic abbreviation expansion modes, where expansions are determined automatically from:
	  • the content of the current buffer (for **DAbbrev**), and
	  • from various, configurable sources (for **Hippie Expand**). Both can be used; they use different commands (and key bindings).
	
	PEL provides the following user-options to control what is being used:		

	pel-use-hippie-expand		When turned on (**t**) PEL remaps **M-/** to use **hippie-expand** instead of **dabbrev-expand**.
	pel-hippie-expand-try-functions		List of functions tried by the hippie-expand command. PEL stores the value of this user-option inside '***hippie-expand-try-functions-list***'.
	pel-dabbrev-friend-buffer-function		Identifies a function used by dabbrev to determine which buffers to search for matches. • Defaults to **dabbrev--same-major-mode-p**. • PEL stores this into **dabbrev-friend-buffer-function.**
	pel-modes-activating-abbrev-mode		List of major modes automatically activating the **abbrev** minor mode. ☝ The abbrev minor mode can be used to expand mode-specific keywords, as abbreviations are stored in mode specific lists. There is also the global abbreviation list that can be used to fix spelling mistakes that were accumulated by the **pel-ispell-word-then-abbrev** command.
	☝ The template expansion packages are not covered here. See **Inserting Text** for mechanisms like **yasnippet** and **Templates** for tempo skeletons. Also, auto-completion (which may use the abbreviation data) is covered in **Auto-Completion**		
Last updated on:	2026-03-19	The Code Completion In Emacs - Everything You Need To Know, describes abbrev, dabbrev and hippie-expand.	
Open this PDF file. See also: **Help/Info**	**<f11> a <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **Abbreviations** 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.
Open PEL abbreviation customization group. See also: **Customize**	**<f11> a <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Open the PEL customize group(s) for the current context. Use this to open to change PEL user option variables the activate and control the various abbreviations features. • When a prefix argument (like **C-u**) opens the buffer inside another window.
Customize Emacs built-in abbreviation support See also: **Customize**	**<f11> a <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs **abbrev** group which includes: abbrev-mode, dynamic-abbreviations, expand, hippie-expand. • When a prefix argument (like **C-u**) opens the buffer inside another window.
	• Group belonging to files that have not yet been loaded are normally not accessible in Emacs and via the customize-group command. PEL, however, attempts to locate the file that defines a non-loaded customization group and will prompt you for loading the file if it finds it.		
Show active abbrev mode status ☝ Provides quick access buttons to change customizable user-options.	**<f11> a ?**	(**pel-abbrev-info** &optional APPEND)	Print abbreviation/expansion control info in a *pel-abbrev-info* help-mode buffer. • Clear previous buffer content by default. Use prefix arg to append to buffer instead.
	• Prints current state and values of relevant user-options as buttons you can use to get more info and change their customized values. • Shows which one is enabled via customization and their current activation state. ☝ Underlines links are buttons that open the customization buffer where you can change the customized value.		
Dynamic Text Expansion: • **dabbrev-expand** • **hippie-expand** ★★	The *dynamic* expansion mode refers to the dynamic identification of the fully expanded text for partial text. • Emacs supports two built-in dynamic expansion commands: **dabbrev-expand** and **hippie-expand**. • The **hippie-expand** provides the same functionality as **dabbrev-expand** but it also includes more selectable sources of abbreviation/expansions. The **M-/** key is bound to **dabbrev-expand** by default. PEL remaps it to **hippie-expand** when **pel-use-hippie-expand** user-option is set to **t**. The following commands are available to explicitly request expansion of an abbreviation.		
Expand Abbreviation	• **M-/** • **<f11> a /**	(**dabbrev-expand** ARG)	Expand previous word "dynamically". Used in DAbbrev mode. PEL activates this when the **pel-use-hippie-expand** user option is set to **nil**
	• Expands to the most recent, preceding word for which this is a prefix. • If no suitable preceding word is found, words following point are considered. If still no suitable word is found, then look in the buffers accepted by the function pointed out by variable 'dabbrev-friend-buffer-function', if 'dabbrev-check-other-buffers' says so. Then, if 'dabbrev-check-all-buffers' is non-nil, look in all the other buffers, subject to constraints specified by 'dabbrev-ignored-buffer-names' and 'dabbrev-ignored-regexps'. • A positive prefix argument, N, says to take the Nth backward *distinct* possibility. A negative argument says search forward. • If the cursor has not moved from the end of the previous expansion and no argument is given, replace the previously-made expansion with the next possible expansion not yet tried. • The variable 'dabbrev-backward-only' may be used to limit the direction of search to backward if set non-nil.		
Hippie Expand Abbreviation See: **Text Expansion with Hippie Expand @ Mastering Emacs**	**M-/**	(**hippie-expand** ARG)	Try to expand text before point, using multiple methods.
		The expansion functions in '***hippie-expand-try-functions-list***' are tried in order, until a possible expansion is found. Repeated application of 'hippie-expand' inserts successively possible expansions. With a positive numeric argument, jumps directly to the ARG next function in this list. With a negative argument or just **C-u**, undoes the expansion. PEL activates this when the **pel-use-hippie-expand** user option is set to **t**.	
See also: **Hide/Show**	**M-/ M-/**	(**hippie-expand** ARG)	When the origami-mode is active, hippie-expand is re-mapped to this key binding. The **origami-mode** is a code folder mode. See **Hide/Show**
Find all possible expansions	• **C-M-/** • **<f11> a c**	(**dabbrev-completion** &optional ARG)	Completion on current word. Like **dabbrev-expand** but finds all expansions in the current buffer and presents suggestions for completion.
	• With a prefix argument ARG, it searches all buffers accepted by the function pointed out by '**dabbrev-friend-buffer-function**' to find the completions. • If the prefix argument is 16 (which comes from **C-u C-u**), then it searches *all* buffers.		

Operation	Keystroke	Function	Note
Manual Abbreviation: Abbrev Mode • To expand abbreviation ➽ • To prevent expansion ➽ • To automatically activate abbrev-mode in modes ➽	Emacs abbreviation system performs abbreviation expansions when the abbrev minor mode is active in the current buffer. • Association of abbreviation to fully expanded text must be defined explicitly with commands identified in the section "Creating Abbreviations" below. 👀 The abbreviations are stored in a file identified by the ***abbrev-file-name*** user option, which is "**~/.emacs.d/abbrev_defs**" by default. • When Emacs terminates, it prompts for saving the new abbreviations. Change that by setting the ***save-abbrevs*** user option to **silently** instead of **t**. ⚠️ Emacs load the abbreviation file at init time. As its size grows, the load time increases and that slows down Emacs startup time. ☝💻 PEL delays the loading of the abbreviation file by temporary changing the value of abbrev-file-name, loading it silently after initialization. • PEL installation instructions describe the required code; see the PEL Manual section **Delay Loading of Abbreviation Definition File**. • When abbrev-mode is active, type the abbreviation then space or punctuation to expand the abbreviation. • Abbreviations are mode specific with a global abbreviation list that can be used to fix your common spelling mistakes. ☝ To **prevent** abbreviation expansion type **C-q** after the abbreviation before typing space or punctuation. ☝ To automatically activate abbrev-mode for some major modes, add the name of these major modes to the **pel-modes-activating-abbrev-mode** user-option. Access the appropriate customize buffer with the **<f11> a <f2>** key sequence.		
Examples The examples show the default key bindings to the commands, but the PEL key bindings can also be used.	To define the following abbrevs:	cnst → construction def → definition	Type *construction* . Type **C-x a g** or **C-x a l** . Enter the abbreviation for it: *cnst* . Hit **RET**. Type *definition* . Type **C-x a g** or **C-x a l** . Enter the abbreviation for it: *def* . Hit **RET**.
	Expand: pre-def → pre-definition		Type: *pre-* **M-'** def **SPACE**
	Expand: re-cnst → reconstruction		Type: *re* **M-'** *cnst* **SPACE**
	Note that if you typed *recnst* and attempt to expand it, Emacs would not recognize *cnst* from it and would not perform any expansion. • The use of the **abbrev-prefix-mark** command (bound to M-') allows this expansion.		
Toggle Abbrev mode on/off ☝ This controls manual expansion, does not impact the dynamic expansions.	**<f11> a a**	(**abbrev-mode** &optional ARG)	Toggle the Abbrev mode on/off in current buffer. Prints new state in minibuffer.
	• Once the Abbrev mode is activated in a buffer, you can type the abbreviation then a space or punctuation (comma, period, return, etc…) to automatically expand the abbreviation to its complete expansion. • To explicitly **prevent the expansion**, type **C-q** just after the abbreviation and before the next character.		
☝ **Create abbreviations from spelling mistakes** See: • **Correcting Typos and Misspellings with Abbrev** • **autocorrection abbrevs** • **Misspellings @ Wikipedia**	Take advantage of the spell check mechanism to identify *abbreviations* that really are spell checking fixes by using the command **pel-ispell-word-then-abbrev** when you detect a spelling error either manually, or through flyspell with **flyspell-auto-correct-word.** After fixing a typo once with the command below, similar typos will be automatically corrected by the abbreviation replacement mechanism. • To activate flyspell corrections inside the abbreviation table to automatically correct future typos, modify the following flyspell user-options: • **flyspell-abbrev-p** : set it to t to automatically store flyspell corrections in local abbrev table. • **flyspell-use-global-abbrev-table-p** : set it to t to have it store in the global abbrev table instead. • If you prefer to define the spell-check detected abbreviations manually, then use the **pel-ispell-word-then-abbrev** command on each one.		
Fix spelling mistake before point • **Add old->new in the abbreviation table** See also: **Spell Checking**	• **<f11> a $** • **<f11> M-$**	(**pel-ispell-word-then-abbrev** &optional LOCALLY)	Fix spelling mistake in text before point. ☝ A similar operation is possible with flyspell. See **flyspell-auto-correct-word.**
	• Create an 'abbrev' abbreviation that match the typo to its correction. • Store the abbreviation globally unless the LOCALLY argument is non-nil, in which case store it in the local abbreviation list. • If there's nothing wrong with the word at point, keep looking for a typo until the beginning of buffer. You can skip typos you don't want to fix with '**SPC**', and you can abort completely with '**C-g**'.		
Manual Expansion of Abbreviations • **With whitespace character or punctuation** • **With M-'**	The following commands expand the abbreviations created manually or pre-defined by the current major mode. • When **abbrev-mode** is active: • abbreviation are expanded automatically by typing a **whitespace character or punctuation** after the abbreviation. • To expand an abbreviation ***after*** a prefix: type the prefix, hit **M-'** , then type the abbreviation and then type a **whitespace character or punctuation**. If the prefix itself is an abbreviation, the M-' command expands it, and when you type space or punctuation after the second abbreviation it is expanded right after the expanded prefix. • To prevent expansion: type **C-q** after the abbreviation before typing space or punctuation.		
Define prefix-based abbreviations ☝ See: **EmacWiki Abbrev Prefixes** for very nice example on how to use this.	**M-'**	(**abbrev-prefix-mark** &optional ARG)	Mark current point as the beginning of an abbrev. • Abbrev to be expanded starts here rather than at beginning of word.
	• This way, you can expand an abbrev with a prefix: insert the prefix, use this command, then insert the abbreviation. • This command inserts a temporary hyphen after the prefix (until the intended abbrev expansion occurs). The prefix is part of the final text. • If the prefix is itself an abbreviation, this command expands it, unless ARG is non-nil. Interactively, ARG is the prefix argument. If you just want to expand an abbreviation, type the abbreviation and then type **M-'**. • The command will expand and append a '-' character that you can then delete if you don't need it; that works even when **abbrev-mode** is turned off. • When **abbrev-mode** is off, If you want expand a word with a prefix type the prefix, type **M-'**, then type the abbreviation and type **M-'** again.		
Expand abbreviation (when abbrev-mode is turned off)	• **C-x '** • **C-x a e** • **<f11> a e**	(**expand-abbrev**)	Expand the abbrev before point, if there is an abbrev there. • This can be used to expand skeleton abbreviations defined in several major modes. • When the abbrev-mode is active you do not need to use this command: the abbreviation will expand after you type a whitespace or punctuation character.
Expand abbreviations in region	**<f11> a E**	(**expand-region-abbrevs** START END &optional NOQUERY)	• For abbrev occurrence in the region, offer to expand it. • The user is asked to type 'y' or 'n' for each occurrence. • A prefix argument means don't query; expand all abbrevs.
Undo last expansion	**<f11> a u**	(**unexpand-abbrev**)	Undo the expansion of the last abbrev that expanded. • This differs from ordinary undo in that other editing done since then is not undone.
Creating Abbreviations Abbreviations can be global or specific to a major mode.	• To **define an abbreviation**, position point after the word(s) that define the target expansion and use: • **C-x a g** to define the abbreviation globally or • **C-x a l** to define the abbreviation for the current mode only. In both cases Emacs prompts for the corresponding abbreviation. • The **inverse method** is to position the cursor after the abbreviation text in the buffer and use: • **C-x a i g** to define the target expansion globally or • **C-x a i l** to define it for the current mode only (local). The *explicit* commands listed below prompt for the abbreviation and its expansion. Abbreviations are either global or local to a specific major mode: • Command **define-global-abbrev** defines abbreviations that are accessible from all buffers. • Command **define-mode-abbrev** defines abbreviations that are only active in the buffer using the current major mode.		
Define an abbreviation	• **C-x a g** • **<f11> a g**	(**add-global-abbrev** ARG)	Define global (all modes) abbrev for last word(s) before point. • Prompts for the abbreviation that will be used when the abbrev-mode is used. • The prefix argument specifies the number of words before point that form the expansion; or zero means the region is the expansion. • A negative argument means to undefine the specified abbrev. • This command uses the minibuffer to read the abbreviation.
Define an abbreviation for the current mode only	• **C-x a l** • **C-x a +** • **C-x a C-a** • **<f11> a l**	(**add-mode-abbrev** ARG)	Define mode-specific abbrev for last word(s) before point. • Argument is how many words before point form the expansion; or zero means the region is the expansion. • A negative argument means to undefine the specified abbrev. • Reads the abbreviation in the minibuffer.
Define expansion	• **C-x a i g** • **C-x a -**	(**inverse-add-global-abbrev** N)	Define last word before point as a global (mode-independent) abbrev. • With prefix argument N, defines the Nth word before point. • This command uses the minibuffer to read the expansion. • Expands the abbreviation after defining it.
Define expansion for the current mode only	**C-x a i l**	(**inverse-add-mode-abbrev** N)	Define last word before point as a mode-specific abbrev. • With prefix argument N, defines the Nth word before point. • This command uses the minibuffer to read the expansion. • Expands the abbreviation after defining it.
***Explicitly* define global abbreviation/expansion**	**<f11> a G**	(**define-global-abbrev** ABBREV EXPANSION)	Define ABBREV as a global abbreviation for EXPANSION. Prompts for both. • All characters in ABBREV must all be word constituents in the standard syntax table.
***Explicitly* define local abbreviation/expansion**	**<f11> a L**	(**define-mode-abbrev** ABBREV EXPANSION)	Define ABBREV as a mode-specific abbreviation for EXPANSION. Prompts for both. • All characters in ABBREV must all be word-constituents in the current mode.

Operation	Keystroke	Function	Note
Editing Abbreviations in the *Abbrev* buffer.	Use the following commands to list all existing abbreviations and modify them inside the *Abbrevs* buffer. • The abbrevs editing buffer contains a header line for each abbrev table, which is the abbrev table name in parentheses. • This is followed by one line per abbrev in that table: `NAME USECOUNT EXPANSION HOOK` where NAME and EXPANSION are strings with quotes, USECOUNT is an integer, and HOOK is any valid function or may be omitted • (it is usually omitted).		
List & Edit all abbreviation definitions	`<f11> a M-l`	(**list-abbrevs** &optional LOCAL)	Display a list of defined abbreviations in **edit-abbrevs-mode** *Abbrevs* buffer. • If LOCAL is non-nil, interactively when invoked with a prefix arg (**C-u**), display only local, i.e. mode-specific, abbrevs. Otherwise display **all** abbrevs. • Edit the text in the opened *Abbrev* buffer to modify the abbreviations. Complete by typing **C-c C-c**.
Edit abbreviations in *Abbrev* buffer • Edit all abbreviations, but • put point over abbrevs of **current** buffer.	• `<f11> a M-e` • `⌘-E`	(**edit-abbrevs**)	Alter abbrev definitions by editing the current mode list in the *Abbrevs* buffer that operates in edit-abbrevs-mode. • Open the *Abbrevs* buffer holding a list of abbrev definitions with point located in the abbrev table for the current buffer. • Turns on '**edit-abbrevs-mode**' in the *Abbrevs* buffer: You can edit them and type **C-c C-c** to redefine abbrevs according to your editing.
***Abbrevs* buffer commands:**	The *Abbrevs* buffer use the **edit-abrev-mode** and lists the abbreviations organized by modes. The following commands can be used in this buffer.		
Use specified abbreviations	`C-c C-c`	(**edit-abbrevs-redefine**)	Redefine abbrevs according to current buffer contents.
Save abbreviations	`C-x C-s`	(**abbrev-edit-save-buffer)**	Save all user-level abbrev definitions in current buffer. • The saved abbrevs are written to the file specified by 'abbrev-file-name'.
Save abbreviations to specified file	`C-x C-w`	(**abbrev-edit-save-to-file** FILE)	Save all user-level abbrev definitions in current buffer to FILE.
Saving Abbreviations	Use the following commands to store abbreviations in a file or buffer, restore abbreviations from a file or a buffer.		
Write abbreviations in a file	`<f11> a s`	(**write-abbrev-file** &optional FILE VERBOSE)	Write all user-level abbrev definitions to a file of Lisp code. • This does not include system abbrevs; it includes only the abbrev tables listed in listed in 'abbrev-table-name-list'.
Read abbreviations from a file	`<f11> a r`	(**read-abbrev-file** &optional FILE QUIETLY)	Read abbrev definitions from file written with 'write-abbrev-file'. • Optional argument FILE is the name of file to read; it defaults to value of 'abbrev-file-name'.
Write abbreviations inside current buffer after point	`<f11> a i`	(**insert-abbrevs**)	Insert after point a description of all defined abbrevs. • Mark is set after the inserted text.
Define abbreviations by reading them from the current buffer	`<f11> a X`	(**pel-extract-abbrev-definitions**) **pel-extract-abbrev-definitions** calls **define-abbrevs** after a positive response to the prompt.	Read abbreviation/expansion data from the current abbrev definition buffer. • Prompt user before proceeding. • See '**edit-abbrevs**' for info on the format of the text you must have in the buffer. • With argument, eliminate all abbrev definitions except the ones defined from the buffer now.
Deleting Abbreviations	To remove all global and local abbreviations, type **M-x kill-all-abbrevs**		
Delete all abbreviations		(**kill-all-abbrevs**)	Un-define all defined abbrevs. Deletes globals & locals (current mode specific) ones.