Abbreviations/Expansions

Operation	Keystroke	Function	Note
Abbreviations			There are several text abbreviation expansion mechanisms available in Emacs. There's manual and dynamic abbreviation expansion modes. The manual mode is the Abbrev mode. In this mode you have to define the abbreviations and the matching expansion. In the dynamic expansion modes, the abbreviations and expansions are determined automatically from the content of the current buffer (for DAbbrev) and from various, configurable sources (for Hippie). There's also template expansion packages, but this page does not cover those.
Dynamic Text Expansion **Dabbrev/Hippie**			The **M-/** key is bound to dabbrev-expand by default and can be re-mapped to hippie-expand. Both implement dynamic expansion. The **hippie-expand** provides the same functionality as **dabbrev-expand** but it also includes more sources of abbreviation/expansions and that is controllable. The following commands are available in those modes.
Expand Abbreviation	`M-/`	**(dabbrev-expand** ARG)	Expand previous word "dynamically". Used in DAbbrev mode. • 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		**(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** customize variable is set to **t**.
Find all possible expansions	`C-M-/`	**(dabbrev-completion** &optional ARG)	Completion on current word. • Like **M-/** 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.
Manual Abbreviation: Abbrev Mode			As described above in Abbrev mode, you must define the abbreviations corresponding to the target expansions manually. It's possible to use a dynamic expansion mode as well as a manual expansion. They use different commands (and key bindings). The following 3 sections of rows describe how to create, delete and expand abbreviations manually.
Creating Abbreviations			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).
Define an abbreviation	`C-x a g`	**(add-global-abbrev** ARG)	Define global (all modes) abbrev for last word(s) before point. • 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`	**(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.
Deleting Abbreviations			To remove all global and local abbreviations, type **M-x kill-all-abbrevs**
Delete all abbreviations		**(kill-all-abbrevs)**	Undefine all defined abbrevs. Deletes both globals and locals (current mode specific) ones.
Manual Expansion of Abbreviations			The following commands expand the abbreviations create manually.
Expand abbreviation with prefix	`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 abbrev. This command inserts a temporary hyphen after the prefix (until the intended abbrev expansion occurs). • If the prefix is itself an abbrev, this command expands it, unless ARG is non-nil. Interactively, ARG is the prefix argument. To use this: 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 delete if you don't need it. If you want expand a word with a prefix type the prefix, type **M-'**, then type the abbreviation and type **M-'** again.
Expand abbreviation	• `C-x a e` • `<f11> a e`	**(expand-abbrev)**	Expand the abbrev before point, if there is an abbrev there.
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.
Editing Abbreviations			Use the following commands to list all existing abbreviations and modify them.
Edit abbreviations in new buffer	`<f11> a l`	**(list-abbrevs** &optional LOCAL)	Display a list of defined abbrevs. 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**.

Operation	Keystroke	Function	Note

Edit abbreviations in *Abbrev* buffer	• `<f11> a m`
	• `⌘-E`
		(edit-abbrevs)	Alter abbrev definitions by editing a list of them.
			• Selects a buffer containing a list of abbrev definitions with point located in the abbrev table for the current buffer, and turns on 'edit-abbrevs-mode' in that buffer.
			• You can edit them and type C-c C-c to redefine abbrevs according to your editing.
			• 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).

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 the file to read; it defaults to the 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 d`	**(pel-define-abbrevs)**	Define abbrevs according to current visible buffer contents after prompting the user. • See documentation of '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. 💻 pel-define-abbrevs calls define-abbrevs after a positive response to the prompt.