Search and Replace

Description	Keystroke	Function	Note
Control/Query how Search Operates	Emacs searches are by default, using: • "case folding" : case insensitive searches. As specified by **search-upper-case** user option variable. • "lax space matching" : where number of spaces between words are considered unimportant. Emacs can also search for words and symbols, and the concept of "words" can be modified to include or exclude underscores and hyphen superword mode, useful for programming languages using snake_case a lot and subword mode that treats sections of camelCase and PascalCase as distinct words The following commands control the various aspects of the search behaviour.		
Show how search behaves in mini buffer	`<f11> s m ?`	(**pel-show-search-case-state**)	Display the search behaviour relative to: case handling, case folding, lax-whitespace and subword and superiors modes in the minibuffer.
Toggle search case sensitivity	`<f11> s m f`	(**pel-toggle-case-fold-search**)	Toggle value of case-fold-search variable.
Toggle lax space searching	`<f11> s m l`	(**isearch-toggle-lax-whitespace**)	Toggle lax-whitespace searching on or off.
Toggle case impact on search	`<f11> s m u`	(**pel-toggle-search-upper-case**)	Toggle case sensitivity behaviour of yank in search prompt. • Rotates the value of search-upper-case to: • nil: upper case don't force case sensitivity • t: upper case force case sensitivity • not-yanks : upper case force case sensitivity, and lower case text when yank in search minibuffer.
Toggle subword-mode (See also: ∑ Text Modes)	• `<f11> t m b` • `<f12> M-b` • `<M-f12> M-b`	(**subword-mode** &optional ARG)	Toggle subword-mode: a minor mode that treats sections of camelCase and PascalCase as distinct words. • With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise. • PEL provides the **`<f12> M-b`** key for the programming language modes where camelCase and PascalCase are popular.
Toggle superword-mode (See also: ∑ Text Modes)	• `<f11> t m p` • `<f12> M-p` • `<M-f12> 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 prefix argument ARG, enable Superword mode if ARG is positive, and disable it otherwise. • PEL provides the **`<f12> M-p`** key for the programming language modes where snake_case is popular (Emacs Lisp, C, C++, Erlang, Python, etc…)
Search Tools Selection (See also: ∑ Customize)	PEL supports several search tools that impact the way the **`C-s`** command operates. PEL supports the following search tools: • Emacs' default ISearch • 📦 Anzu, ISearch with match count : ✎ set **pel-use-anzu** to **t**. • 📦 Swiper search with overview match list : ✎ set **pel-use-swiper** to **t** ⚙ Use **`<f11><f1> s`**, to customize the PEL completion group user options above. • Set the **pel-initial-search-tool** user option to select which search tool is used when Emacs starts. When any of the extra search tool is activated via the corresponding pel-use- user option, pel-init makes the following commands available to change the currently used search tool and to see which one is currently active.		
Select search tool to use	`<f11> s s`	(**pel-select-search-tool**)	Prompt user for search tool to use with **`C-s`**. Show new active one. • Emacs normally maps the search-forward command to **`C-s`**. • PEL provides the ability to activate the following tools that can be activated for searching: • 📦 The Anzu external package ✎ activated by **pel-use-anzu** user option. Anzu provides a match count in the modeline when search command is used. • 📦 The Swiper external package ✎ activated by **pel-use-swiper** user option. Swiper is not using isarch-forward; it shows a list of matching lines in the mini-buffer. • ⚙ Use the **`<f11> <f1> s`** command to open the PEL search customize group and set the **pel-initial-search-tool** user option to identify which tool is used when Emacs starts. See the ∑Customize table for more information. ☝ Being able to search using either Emacs default ISearch (see below) and Swiper helps as they are both very useful in different scenarios.
Show which search tool is currently used	`<f11> ? s`	(**pel-show-active-search-tool**)	Display the currently used search tool.
newlines in search and replace	⚠️**New line in search and replace**: • Several editors use the C string syntax "\n" to identify the newline character. Emacs does **not** use it in search and replace queries. • In Emacs search and replace queries use **`C-q C-j`** to identify newline characters.		
Non-Incremental Search	The *normal* (non-incremental) search can be performed using the commands and keystrokes listed below. • They can also be invoked by typing <RET> right after the invocation of the incremental search commands (see below).		
Search for word taken at point from the top of current or specified window (See also: ∑ Key-Chords)	• `<f11> s .` • `.;`	(**pel-search-word-from-top** &optional N)	Search word at point from top/bottom of buffer in window identified by N. • **Search direction**: • If N is nil, 0 or larger, perform a search-forward from the top of the buffer in window identified by N. • If N is negative: perform a isearch-backward from the bottom of the buffer in the window selected by the absolute value of N. • **Window selection**: • If N is not specified, nil, 1, 3, 7 or 9 and larger: search in current window. • If N is 0: : search in other window • If N in [2,8] range, search in window identified by the direction corresponding to the cursor in a numeric keypad: 8 := 'up 4 := 'left 5 := 'current 6 := 'right 2 := 'down • **Temporary word mode toggle**: detecting a '*word*' is affected by the subword-mode and superword-mode. When searching in current buffer, the following values of N temporary toggle the mode when grabbing the word: • If N is 7: temporary toggle **subword-mode** to grab the word. • If N is 9: temporary toggle **superword-mode** to grab the word. • Explicitly selecting the minibuffer window, or a non-existing window is not allowed, and search is done in current window. • Searched word is remembered and can be used again to repeat an interactive search with **`C-s`** or **`C-r`**. • Position before searched word is pushed on the mark ring. ☝ Using **superword-mode** allows you to search for function names in buffer for programming languages. If you do not want to change the mode but want to search for the word as interpreted by the other state of the mode type the command with N equal to 9: **`M-9 <f11> s .`** 📦✎ With PEL, the **`.;`** key-chord is also available when pel-use-key-chord is non-nil. See ∑ Key-Chords. ☝ Command numeric prefix **is available** with the key-chord binding.

Description	Keystroke	Function	Note
Search forward	`<f11> s f`	(**search-forward** STRING &optional BOUND NOERROR COUNT)	Search forward from point for STRING. • Set point to the beginning of the occurrence found. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'. • ⚠️ Lax Search is not supported.
Search backward	`<f11> s b`	(**search-backward** STRING &optional BOUND NOERROR COUNT)	Search backward from point for STRING. • Set point to the beginning of the occurrence found. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'. • ⚠️ Lax Search is not supported.
Search regexp forward	`<f11> s x f`	(**re-search-forward** REGEXP &optional BOUND NOERROR COUNT)	Search forward from point for regular expression REGEXP. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'.
Search regexp backward	`<f11> s x b`	(**re-search-backward** REGEXP &optional BOUND NOERROR COUNT)	Search backward from point for regular expression REGEXP. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'.
Word Search	A word search finds a sequence of words without regard for the type of punctuation between them. • The word search commands do not perform character folding and toggling lax whitespace matching have no effect on them. • However there are "lax" word searches that succeed on incomplete words, they are listed below.		
Incremental Search Word	• `M-s w` • `<f11> s w i`	(**isearch-forward-word** &optional NOT-WORD NO-RECURSIVE-EDIT)	Do incremental search forward for a **sequence of words**. • With a prefix argument, do a regular string search instead. • Like ordinary incremental search except that your input is treated as a sequence of words without regard to how the words are separated. • See the command '**isearch-forward**' for more information.
Search word forward	• `M-s w <RET>` • `<f11> s w f`	(word-search-forward STRING &optional BOUND NOERROR COUNT)	Searches for exact words that may be separated by punctuations and/or lines. Search string must be a complete set of words.
Search word forward lax	`<f11> s w F`	(word-search-forward-lax STRING &optional BOUND NOERROR COUNT	Same as search word forward except that the search string may end in an incomplete word (unless it ends with whitespaces)
Search word backward	• `M-s w C-r <RET>` • `<f11> s w b`	(word-search-backward STRING &optional BOUND NOERROR COUNT	Searches for exact words that may be separated by punctuations and/or lines. Search string must be a complete set of words.
Search word backward lax	`<f11> s w B`	(word-search-backward-lax STRING &optional BOUND NOERROR COUNT)	Same as search word forward except that the search string may end in an incomplete word (unless it ends with whitespaces)
Incremental Search (ISearch)	Start an incremental search with one of the following commands. Type text to search, **`<DEL>`** to remove chars. Other key-chords can be used during the search. Re-type same key-chord after reaching end of buffer, wrap to other end and continue searching. Or repeat key-chord to repeat last search for same text. To reverse search direction, use the other key-chord (for example: if searching with **`C-s`**, use **`C-r`** to go backward) • Type **`<RET>`** to stop search and leave cursor at found position if next command is to insert a character. Other editing key-chords also stop the search but also perform the requested operation (like **`C-a`** which ends the search and moves point to the beginning of the line). • Abandon search (and return to where you started, type **`<ESC><ESC><ESC>`** or **`C-g C-g`**. On search exit, original point is added to mark ring, thus you can use **`C-u C-SPC`** or **`C-x C-x`** to return to the position before the search. ☝📦 **`C-s`** is normally mapped to isearch-forward. With PEL you can set the **pel-use-swiper** user option which activates the Swiper external package and the **`<f11> s s`** key. That key allows you to change what command is mapped to **`C-s`**: search-forward or swiper. You can specify which one is used by default via the **pel-initial-search-tool** user option. Use **`<f11> <f1> s`** to customize PEL controlled search.		
ISearch - forward • **Incremental** • **literal search** • **regexp search**	• `C-s` • `⌘-f`	(**isearch-forward** &optional REGEXP-P NO-RECURSIVE-EDIT)	Do incremental search forward: start or continue a search. • With a prefix argument, do an **incremental regular expression search** instead, something like: • **`C-u 1 C-s`** • **`M-- C-s`** • With PEL, **`C-- C-s`** works. • **`C-u C-s`** does **not** work to perform a regexp ISearch. ☛ Instead you can also use **`C-M-s`** to perform the regexp incremental search forward. • To continue to next match during search: type **C-s** again (with prefix argument if that was used for regexp Isearch). • To change direction: type **C-r** • To repeat last completed incremental search forward: **C-s C-s** • `⌘-f` is always mapped to isearch-forward. • When Anzu is used (see below) the modelling shows the match count. 📦 On PEL: • This key mapping is used when either **pel-initial-search-tool** nil or 'anzu' when **pel-use-anzu** is **t**. • If **pel-use-swiper** is **t**, you can use **`<f11> s s`** to change the tool used for search operations.
Perform Swiper search: **interactive search with an overview list**	`C-s`	(**swiper** &optional INITIAL-INPUT)	Perform a Swiper text search. Opens up the mini buffer and show several matches as they are being typed. • Narrow the search by typing a pattern. • Multiple patterns are allowed by separating with a space. • Select with C-n, C-p, <up> and <down>. • Chose (and stop the search) with RET. • ☝To search for a space with Swiper, type 2 spaces in the search expression. So: type "foo␣␣bar" to search for "foo␣bar". 📦 On PEL: • This key mapping is used when **pel-use-swiper** is **t** and **pel-initial-search-tool** is set to swiper. • You can use **`<f11> s s`** to change the tool used for search operations.
ISearch - backward • **Incremental** • **literal search** • **regexp search**	`C-r`	(**isearch-backward** &optional REGEXP-P NO-RECURSIVE-EDIT)	Do incremental search backward: start or continue a search. • With a prefix argument, do an **incremental regular expression search** instead; something like: • **`C-u 1 C-r`** • **`M-- C-s`** • With PEL, **`C-- C-r`** works. • **`C-u C-r`** does **not** work to perform a regexp ISearch. ☛ Instead you can also use **`C-M-r`** to perform the regexp incremental search forward. • To continue to next match during search: type **C-r** again (with prefix argument if that was used for regexp Isearch. • To change direction: type **C-s** • To repeat last previously completed incremental search backward: **C-r C-r** • When Anzu is used (see below) the modelling shows the match count. 📦 On PEL: • This key mapping is used when either **pel-initial-search-tool** nil or 'anzu' when **pel-use-anzu** is **t**. • If **pel-use-swiper** is **t**, you can use **`<f11> s s`** to change the tool used for search operations.

	Description	Keystroke	Function	Note
	ISearch - Regexp— forward • **Incremental** • **regexp search**	**`C-M-s`**	(**isearch-forward-regexp** &optional NOT-REGEXP NO-RECURSIVE-EDIT)	Incremental forward regular expression search. ☛ Everything that can be done with **`C-s`** can also be done here. For example repeating the search can be done with **`C-s`**.
	ISearch - Regexp - backward • **Incremental** • **regexp search**	**`C-M-r`**	(**isearch-backward-regexp** &optional NOT-REGEXP NO-RECURSIVE-EDIT)	Incremental backward regular expression search. ☛ Everything that can be done with **`C-r`** can also be done here. For example repeating the search can be done with **`C-r`**.
	Visual Regexp ISearch with Python regexp engine	**`<f11> s x C-s`**	(**vr/isearch-forward**)	Like isearch-forward, but using Python (or custom) regular expressions. 📦 Requires visual-regexp-steroids : 🛂 available when pel-use-visual-regexp-steroids is **t**.
	Visual Regexp backward ISearch with Python regexp engine	**`<f11> s x C-r`**	(**vr/isearch-backward**)	Like isearch-backward, but using Python (or custom) regular expressions. 📦 Requires visual-regexp-steroids : 🛂 available when pel-use-visual-regexp-steroids is **t**.
	Incremental Symbol Search	Incremental **symbol** search is like incremental search except that the boundaries of the search must match the boundaries of a symbol (for the buffers' major mode). Only complete match will be found. For example searching for *forward-word* in a Lisp file will not match *isearch-forward-word*. Note: ☝ also see the command described above: **pel-search-word-from-top**, bound to **`<f11> s .`**		
	ISearch symbol at point	**`M-s .`**	(**isearch-forward-symbol-at-point**)	Perform a symbol search starting with current symbol at point. • After capturing the word at point you can extend it by typing **`M-w`**. • ☝ Useful for searching inside source code while superiors mode is disabled. • Use **`C-s`** and/or **`C-r`** to perform extra searches on the same symbol.
	ISearch for symbol	**`M-s _`**	(**isearch-forward-symbol** &optional NOT-SYMBOL NO-RECURSIVE-EDIT)	Prompt for symbol, perform symbol search. • Subsequent searches for the same symbol is done with **`C-s`** and/or **`C-r`**. • ☝ Useful for searching code. For example: "data size" matches "data.size" as well as "data->size", "data + size" and "data size".
	ISearch for sequence of words	**`M-s w`**	(**isearch-forward-word** &optional NOT-WORD NO-RECURSIVE-EDIT)	Do incremental search forward for a **sequence of words**. • With a prefix argument, do a regular string search instead. • Like ordinary incremental search except that your input is treated as a sequence of words without regard to how the words are separated.
	During ISearch	The incremental search can be modified to perform other searches. Right after typing the incremental search command you can type the following characters to modify or repeat the search.		
DURING ISEARCH COMMANDS	**Change the search type to:** **simple search**	**`<RET>`**	• (**search-forward** STRING &optional BOUND NOERROR COUNT) • (**search-backward** STRING &optional BOUND NOERROR COUNT)	Typing <RET> **right after typing** the command (**`C-s`**, **`C-r`**, **`C-M-s`** or **`C-M-r`**) and before typing the text to search for: • **`C-s <RET>`** or **`C-r <RET>`** perform a regular search instead of an ilSearch. • **`C-M-s <RET>`** or **`C-M-r <RET>`** perform a regular regex search.
	Add word at point to search string	**`C-w`**	(**isearch-yank-word-or-char**)	Appends the next character or word at point to the search string. Repeat it to append more to the search string.
	repeat search forward	• **`C-s`** • **`⌘-g`**	(**isearch-repeat-forward**)	Repeat the current search, start searching again going forward
	repeat search backward	• **`C-r`** • **`⌘-d`**	(**isearch-repeat-backward**)	Repeat the current search, start searching again going backward
	Select searched string	While performing a search you can issue the following commands to modify the searched string text.		
	History previous	**`M-p`**	(**isearch-ring-retreat**)	Retrieve searched text from search history: get previous entry from history
	History next	**`M-n`**	(**isearch-ring-advance**)	Retrieve searched text from search history: get next entry from history
	"tab" complete history in buffer	• **`C-M-i`** • **`M-<tab>`**	(**isearch-complete**)	Perform "tab" completion for search item in the minibuffer against the search history. Opens a buffer with the complete search history. Any one of the past search string can be selected to perform the new search.
	Edit search string	**`M-e`**	(**isearch-edit-string**)	Use this while performing a search and wanting to change the string being searched. • When **`M-e`** is typed during the search, the prompt goes back to the minibuffer allowing the editing of the searched string. • Edit then search string in minibuffer. • End editing with **`<RET>`**, **`C-j`**, **`C-s`** or **`C-r`**
	Add rest of line at point to search string	**`M-s C-e`**	(**isearch-yank-line** &optional ARG)	While searching select the text from cursor to end of line as the search text. If point is already at end of line, appends next line. With numeric argument appends that many next lines.
	Add character at point to search string	**`C-M-y`**	(**isearch-yank-char** &optional ARG)	Appends character at point to the search string. If numeric argument appends that many characters.
	Yank from kill ring to search string	• **`C-y`** • **`⌘-e`**	(**isearch-yank-kill**)	Pull string from kill ring into search string.
	Replace just-yanked search string with previously killed string	**`M-y`**	(**isearch-yank-pop**)	Replace just-yanked search string (via (search-yank-kill) with previously killed string.
	Modify search method	While performing a search the following commands modify the search method.		
	Start query replace	**`M-%`**	(**isearch-query-replace** &optional ARG REGEXP-FLAG)	Transforms the Search into a query replace, using the current string as the string to be replaced.
	Start query replace regexp	**`C-M-%`**	(**isearch-query-replace-regexp** &optional ARG)	Transforms the Search into a regex query replace, using the current string as the regex string to be replaced.
	Enter occur search: list all occurrences	**`M-s o`**	(**isearch-occur** REGEXP &optional NLINES)	Start an "occur" search with current search string. • See "**`M-s o`**" row above for more information.
	Modify search mode	While performing a search the following commands modify the search modes.		
	Toggle lax whitespace matching	**`M-s SPC`**	(**isearch-toggle-lax-whitespace**)	Toggle lax matching during this search. Lax matching is on by default. • Any number of whitespace is accepted in the default lax matching. This can also be customized. When off: search exact string.
	Toggle case sensitivity	• **`M-c`** • **`M-s-c`**	(**isearch-toggle-case-fold**)	Toggle search case sensitivity.
	Toggle searching in invisible text	**`M-s i`**	(**isearch-toggle-invible**)	Toggle whether invisible text is searched. • Useful when editing outlined text.
	Toggle regular-expression searching	• **`M-r`** • **`M-s-r`**	(**isearch-toggle-regexp**)	Toggle regexp searching on or off.

Description	Keystroke	Function	Note
Toggles word mode	`M-s w`	(**isearch-toggle-word**)	Toggle word searching on or off. • Turning on word search turns off regexp mode. • For example: in C file : the expression it->second.first is not matched by “is second first” but when the word mode (or the symbol mode) is activated it matches.
Toggles symbol mode	`M-s _`	(**isearch-toggle-symbol**)	Toggle symbol search mode. • Useful for searching code. For example: “data size” matches “data.size” as well as “data->size”, “data + size” and “data size”.
Toggle character folding	`M-s ‘`	(**isearch-toggle-char-fold**)	Toggle char-fold searching on or off. • Turning on character-folding turns off regexp mode. • When character folding is activated all accentuated letters for a given letter match the letter., otherwise it does not match (ie: ‘à’ matches ‘a’ when character folding is activated and does not otherwise).

Stop the incremental search	`<RET>`
	`C-g`
		: Pick found text. Stop current search and leave cursor right after the found text.
		: Aborts current search and return point to original location.	

Occur Search			
List all matching occurrences of regexp in current buffer	**`M-s o`**	(**occur** REGEXP &optional NLINES)	• Prompts for a regexp • Can use **`M-n`** at prompt to recuse previous search strings • Use **`M-n`** prefix to specify n lines of context in result. Default=*list-matching-lines-default-context-lines*. • “**`M-s o`**” can be used during an incremental search. • **In *Occur* buffer**: • <RET> visit corresponding position in the searched buffer • “C-o” display the match in other window (but does not select it) • < , > : go to the beginning and end of the buffer • g : revert the buffer, refreshing the search results • e : buffer enters the **Occur Edit Mode** which allows edits in both buffers simultaneously via edits in the *Occur* buffer. • Exit Occur Edit Mode with: • “**C-c C-c**” (which is: (**occur-cease-edit**)) • Navigate though occurrences (in original buffer): • (next-error) : “**`C-x ``**” or “**`M-g n`**” or “**`M-g M-n`**” • (previous-error): “**`M-g p`**” or “**`M-g M-p`**”
Occur search in selected buffers	`<f11> s O`	(**multi-occur-in-matching-buffers** BUFREGEXP REGEXP &optional ALLBUFS)	For example to occur search in all .py files, select the buffers with “**\.py$**” (without the quotes).
Occur search in selected files	`<f11> s o`	(**multi-occur** BUFS REGEXP &optional NLINES)	
During Occur Search			
occur - next occurence	• **`C-x ``** • **`M-g n`** • **`M-g M-n`**	(**next-error** &optional ARG RESET)	A prefix ARG specifies how many error messages to move; • negative means move back to previous error messages. • Just **`C-u`** as a prefix means reparse the error message buffer and start at the first error.
occur - previous occurence	• **`M-g p`** • **`M-g M-p`**	(**previous-error** &optional N)	Prefix arg N says how many error messages to move backwards (or forwards, if negative).
Exit occur mode	`C-c C-c`	(**occur-cease-edit**)	Exit the occur-edit mode. See “**`M-s o`**” note above.
PEL Search/Replace Command Selection	Several regexp engines are available to perform regexp searches. Emacs provides its own. But several external packages provide extensions. • The visual-regexp enhances the Emacs regexp search and replace engine by showing matches in the buffer while you are typing the regexp, providing useful feedback when learning Emacs regexp. For replacement it shows both the match in original text and its replacement. • The visual-regexp-steroids extends this further by allowing the use of other rexgexp engines like pcre2el and Python. PEL provides the ability to dynamically select the extension to use for your regexp search and replace operations. Use the **`<f11> ? S`** to see what is available and active and use **`<f11> s S`** to change the regexp engine.		
	Emacs provides the following commands to perform string replacement in buffers. • The following external packages also provides several useful extensions: • pcre2el : available when pel-use-pcre2el is **t** • visual-regexp : available when pel-use-visual-regexp is **t** • visual-regexp-steroids : available when pel-use-visual-regexp-steroids is **t** • xr : available when pel-use-xr is **t**		
Search/Replace Regexp Engine Selection (See also: ∑ Customize)	PEL supports several regular expression search/replace engines that control the way several Search and Replace commands operate: • Emacs’ default regex engine • 📦 visual-regexp : set **pel-use-visual-regexp** to **t** • It provides visual feedback during search and replace operation: it shows matches in buffer while typing the search text. • 📦 visual-regexp-steroids : set **pel-use-visual-regexp-steroids** is **t** • An extension to visual-regex which provides same visual feedback but supports several regexp engines: • emacs, emacs-plain, pcre2el, python and custom (no special custom regexp yet implemented by PEL). • Use **`<f11><f1> r`**, to customize the PEL completion group user options above. • Set the **pel-initial-regexp-engine** user option to select which regexp engine is used when Emacs starts. When any of the extra search tool is activated via the corresponding pel-use- user option, pel-init makes the following commands available to change the currently used search/replace regexp engine and to see which one is currently active.		
Select the search/replace regexp engine	`<f11> s S`	(**pel-select-search-regexp-engine**)	Select the search/replace and regexp engine to use. • Shows currently used engine at the prompt. Supports completion. 🚧 With PEL, Activating the engines provided by visual-regexp-steroids currently prevents restoring the original engine. More work is required on PEL code for full dynamic flexibility.
Show which search/replace regexp engine is used	`<f11> ? S`	(**pel-show-active-search-regexp-engine** &optional WTH-DETAILS)	Display the currently used search regexp engine. Display a detailed message describing what is available the first time it is run and when a prefix argument is used (**`C-u`** or any numeric argument will do).
Unconditional Replace	Simple text replacement command.		
Unconditional replace	`<f11> s r`	(**replace-string** FROM-STRING TO-STRING &optional DELIMITED START END BACKWARD)	Replace all instances of from-string by to-string from point to end of buffer. Emacs displays the number of string replaced after the operation.

Description	Keystroke	Function	Note
Unconditional regex replace	• **<f11> s x r** • **C-c r**	(**replace-regexp** REGEXP TO-STRING &optional DELIMITED START END BACKWARD) — (**pel-replace-regexp**) : • (**replace-regexp** REGEXP TO-STRING &optional DELIMITED START END BACKWARD) • (**vr/replace** REGEXP REPLACE START END) • (**vr/select-replace**)	Replace every match for regex with new string. ☑ PEL only activates the **C-c r** binding if the pel-bind-keys-for-regexp user option is set to **t**. 📦 ☑ With PEL, when any of pel-use-visual-regexp or pel-use-visual-regexp-steroids is set to **t** , you can select a regexp engine provided by these external package (using **<f11> s S** to select another) and it affects what command is used here (pel-replace-string uses the command corresponding to your selection). ☝ It's possible to use lisp expressions in the replacement string, making this super powerful. See **examples in the Emacs Wiki.**
Visual Regexp Replace	**<f11> s x R**	(**vr/replace** REGEXP REPLACE START END) 📦 Requires visual-regexp : ☑ available when pel-use-visual-regexp is **t**.	Replace every match for regex with new string. With visual feedback. The following sub-commands are available while composing the search text: • **M-p** : Previous search/replacement string • **C-c ?** : help • **C-c a** : toggle show all or up to the default limit. Default limit is specified by **vr/default-feedback-limit** • **C-c p** : toggle preview • The following are available only when using the Python regexp engine: • **C-c i** : toggle case sensitivity (ignore case) • **C-c m** : toggle multi-line match of ^ and $ • **C-c s** : toggle dot matches newline • **C-c u** : enable Unicode by default.
Visual Regexp Replace with engine selection	**<f11> s x M-r**	(**vr/select-replace**) 📦 Requires visual-regexp-steroids : ☑ available when pel-use-visual-regexp-steroids is **t**.	
Visual Regexp Search to multiple-cursors (See also ∑ Cursor)	• **<f11> s x M** • **C-c m**	(**vr/mc-mark** REGEXP START END) 📦 This requires both visual-regexp and multiple-cursors external packages.	Convert regexp selection to multiple cursors. • First performs a Visual regexp search. When the result of the search is accepted (by hitting **RET**) all matches are converted to multiple cursors, which allows performing the same operations on all matches until the user quits the multiple cursor operation with **C-g**. ☑ PEL activates this command when both **pel-use-multiple-cursors** is **t** and either pel-use-visual-regexp or visual-regexp-steroids is **t**. ☑ PEL only activates the **C-c m** binding if the pel-bind-keys-for-regexp user option is set to **t**.
Visual Regexp Search to multiple-cursors with engine selection (See also ∑ Cursor)	**<f11> s x M-m**	(**vr/select-mc-mark**) 📦 This requires both visual-regexp-steroids and multiple-cursors external packages.	
Query Replace	Query replacement prompts. The following 2 commands are query replace. The answers to prompts are listed after the 2 commands.		
Query Replace	**M-%**	(**query-replace** FROM-STRING TO-STRING &optional DELIMITED START END BACKWARD REGION-NONCONTIGUOUS-P)	Replace *some* occurrences of a string with another, both specified by user. A negative argument replaces backwards.
Query Replace Regexp	• **C-M-%** • **<f11> s x q** • **C-c q**	(**query-replace-regexp** REGEXP TO-STRING &optional DELIMITED START END BACKWARD REGION-NONCONTIGUOUS-P) — (**pel-query-replace-string**)	Replace *some* occurrences of a regex match with a specified string. • A negative argument replaces backwards. • C-M-% does not work in Terminal mode. ☑ PEL only activates the **C-c q** binding if the pel-bind-keys-for-regexp user option is set to **t**. 📦 ☑ With PEL, when any of pel-use-visual-regexp or pel-use-visual-regexp-steroids is set to t , you can select a regexp engine provided by these external package (using **<f11> s S** to select another) and it affects what command is used here (pel-query-replace-string uses the command corresponding to your selection).
Visual Regexp Query Replace	**<f11> s x Q**	(**vr/query-replace** REGEXP REPLACE START END) 📦 Requires visual-regexp : ☑ available when pel-use-visual-regexp is **t**.	Replace *some* occurrences of a regex match with a specified string with visual feedback inside the buffer. • A negative argument replaces backwards. The following sub-commands are available while composing the search text: • **M-p** : Previous search/replacement string • **C-c ?** : help • **C-c a** : toggle show all or up to the default limit. Default limit is specified by **vr/default-feedback-limit** • **C-c p** : toggle preview • The following are available only when using the Python regexp engine: • **C-c i** : toggle case sensitivity (ignore case) • **C-c m** : toggle multi-line match of ^ and $ • **C-c s** : toggle dot matches newline • **C-c u** : enable Unicode by default.
Visual Regexp Query Replace with engine selection	**<f11> s x M-q**	(**vr/select-query-replace)** 📦 Requires visual-regexp-steroids : ☑ available when pel-use-visual-regexp-steroids is **t**.	
QR Response : keys to use during a query replacement to identify actions	• y or SPC : replace • n or <DEL> : don't replace, move to next • . : replace current and quit • , : replace & let me see result before moving on — Press SPC to move on. • ! : replace all the rest and don't ask • ^ : back up to the previous instance • u : undo last replacement • U : undo ALL replacements • q or <RET> : abort/exit query-replace • E : modify the replacement string • C-r : enter recursive edit - Exit the recursive edit with one of: C-M-c or C-] • C-w : delete this instance and enter recursive edit —to make a custom replacement • C-M-c : exit recursive edit and resume query-replace • C-] : Exit recursive edit and exit query-replace • ? : get help • Y : replace all strings in all buffer, no questions. — Multi-buffer QR Response • N : skip to next buffer without replacing remaining matches in current buffer — Multi buffer QR Response.		
Interpret and Lint Emacs Lisp Regexp with xr. **Convert it to rx-style semantic form**	📦 The xr external package provides a function that interprets Emacs Lisp regexp and prints a descriptive **rx-style semantic form** to explain it. • All commands described below require the xr external package activated when the **pel-use-xr** user option is set to **t.** • ☝ The rx Emacs Lisp macro can be used in Emacs Lisp code to express a regexp in a more readable fashion. Type **<f1> o rx** for more info. ☑ PEL provides xr when the **pel-use-xr** user option is set to **t.** PEL provides the following commands which take a regexp at the prompt or from text at point to print a description inside the *regexp-eval* buffer.		
Interpret Emacs Lisp regexp at point.	**<f11> s x x**	(**pel-xr-at-point** &optional DIALECT)	Grab regexp at point and print its interpretation in *regexp-eval* buffer. • Uses `xr-pp' to expand regexp in rx notation. • If region is marked, grab content of region instead. • DIALECT is selected by numeric argument: • nil, 1 := medium verbose • < 0 := terse • 4 := brief : short keywords • 16 := verbose : verbose keywords. To pass 4 type the **C-u** prefix, and 16 type **C-u C-u** prefix keys. **LIMITATION**: it does not support double quote inside a regexp taken at point even if it is quoted. To grab it mark the region, excluding the delimiting quotes.

<table>
<tr><th>Description</th><th>Keystroke</th><th>Function</th><th>Note</th></tr>
<tr><td>**Interpret Emacs Lisp regexp provided at prompt.**</td><td>`<f11> s x X`</td><td>(**pel-xr-regxp**)</td><td>Prompt for regexp and print its interpretation in *regexp-eval* buffer.
• Uses `xr-pp' to expand regexp in rx notation.</td></tr>
<tr><td>**Lint Emacs Lisp regexp at point**</td><td>`<f11> s x l`</td><td>(**pel-xr-lint-at-point** &optional FOR-FILE-MATCH)</td><td>Lint the regexp at point or inside region if region is marked.
• If FOR-FILE-MATCH argument is non-nil (use any prefix keystroke such as `C-u` or `M--` or `C--`), perform additional checkings to see if the regexp is OK for matching file name.
LIMITATION:
Does not support double quote inside a regexp taken at point even if it is quoted. To grab it mark the region, excluding the delimiting quotes.</td></tr>
<tr><td>**Lint Emacs Lisp regexp provided at prompt**</td><td>`<f11> s x L`</td><td>(**pel-xr-lint** &optional FOR-FILE-MATCH)</td><td>Prompt for a regexp, lint it and display results.
If FOR-FILE-MATCH argument is non-nil (use any prefix keystroke such as `C-u` or `M--` or `C--`), perform additional checkings to see if the regexp is OK for matching file name.</td></tr>
<tr><td>**Regex-tool**</td><td colspan="3">📦 The external regex-tool library implements a simple regular expression tester tool. PEL activates it when **pel-use-regex-tool** is **t**.
• While regex-tool is running: type `C-c C-c` to force an update and `C-c C-k` to quit using it.
• The regex-tool uses Emacs Lisp regular expressions by default. It can also use full Perl regexp if you have Perl installed on your system.
The **regex-tool-backend** user option identifies the regexp engine used. It can be emacs or perl.</td></tr>
<tr><td>**Open the regex-tool**</td><td>`<f11> s x T`</td><td>(**regex-tool**)</td><td>Open a 3-window frame (replacing all previous windows). The 3 windows are:
• Regular expression: enter/edit the expression freely
• Test string: enter text to match against
• Groups: lists the matching groups</td></tr>
<tr><td>**Force an update of regex-tool windows**</td><td>`C-c C-c`</td><td>(**regex-tool-markup-text** &optional BEG END LEN)</td><td>Force an update of the regex-tool windows.</td></tr>
<tr><td>**Quit regex-tool**</td><td>`C-c C-k`</td><td>(**regex-tool-quit**)</td><td>Quit regex-tool and close its 3 windows, reverting to the window layout used before it was just turned on.</td></tr>
<tr><td>**Change the regex-tool backend engine - select between Emacs and Perl.**</td><td>`C-c <f1>`</td><td>(**pel-select-regex-tool-engine**)</td><td>Open the customize buffer to change **regex-tool-backend** user option.
• Select between Emacs and Perl backend.
• To close the customize buffer, type `q`.
• Force an update of the regex-tool to rescan using the new backend, with `C-c C-c`.</td></tr>
<tr><td>**re-builder:**
Emacs Regular Expression Builder</td><td colspan="3">Emacs provides another regexp tester: the built-in Regular Expression Builder, targeted to learn the Emacs regular expression syntax.
• To open (start) the regular expression, execute `M-x re-builder`. PEL provides the `<f11> s x B` key for that.
• While the re builder is running:
  • type the regular expression (regexp) and see the matches in the other window,
  • if needed, change the regular expression syntax (Emacs supports 3 syntaxes, see below):
    • Use `C-c C-i` to select the new syntax.
    • With PEL, you can also use `<f11> s x <f1>` to quickly open the customize page to change the default syntax user option.
  • use one of the specialized commands available in reb-mode. These are listed below.
• To close (stop) the re-builder, type `C-c C-q`</td></tr>
<tr><td>**Build regular expression interactively with re-builder**

☝This is a great way to learn Emacs regexp!</td><td>`<f11> s x B`</td><td>(**re-builder**)</td><td>Construct and test a regexp **interactively**.
• This command makes the current buffer the "target" buffer of the regexp builder. It displays a buffer named “*RE-Builder*" in another window, initially containing an empty regexp.
• As you edit the regexp in the "*RE-Builder*" buffer, the matching parts of the target buffer will be highlighted.
☝re-builder supports different styles of regular expressions, selected by the value of the **reb-re-syntax** user option. The possible values are:
  • ***read***: the *default*. Similar to ***string*** but requires double escaping of backslashes - similar to how it must be done in Elisp source code. For example: “\\(red\\|green\\)”
  • ***string***: Similar to ***read*** but no double backslashes are needed. Example: “\(red\|green\)”
  • ***rx***: A more advanced, s-expression regexp engine, used if you want lisp-style regexp engine.</td></tr>
<tr><td>**Select the regular expression syntax used by the re-builder**</td><td>`<f11> s x <f1>`</td><td>(**pel-reb-re-syntax**)</td><td>Select regular expression syntax used by the re-builder:
• customize **reb-re-syntax** user option.
☝This user option is part of the re-builder group which contains other related settings.
• This is a global binding: it can be used any time.</td></tr>
<tr><td>**Change target buffer**</td><td>`C-c C-b`</td><td>(**reb-change-target-buffer** BUF)</td><td>Change the target buffer and display it in the target window.</td></tr>
<tr><td>**Enter/leave sub-expression highlight mode**</td><td>`C-c C-e`</td><td>(**reb-enter-subexp-mode**)</td><td>Enter the subexpression mode in the RE Builder.
• Use this to only highlight the capturing groups.
• Type `0` to `9` to identify the group to highlight.
• Type `q` to exit that mode.</td></tr>
<tr><td>**Select regular expression syntax used**</td><td>• `C-c C-i`
• `C-c <tab>`</td><td>(**reb-change-syntax** &optional SYNTAX)</td><td>Change the syntax used by the RE Builder.</td></tr>
<tr><td>**Quit re-builder**</td><td>`C-c C-q`</td><td>(**reb-quit)**</td><td>Quit the RE Builder mode.</td></tr>
<tr><td>**Move point to previous match**</td><td>`C-c C-r`</td><td>(**reb-prev-match)**</td><td>Go to previous match in the RE Builder target window.</td></tr>
<tr><td>**Move point to next match**</td><td>`C-c C-s`</td><td>(**reb-next-match**)</td><td>Go to next match in the RE Builder target window.</td></tr>
<tr><td>**Force update**</td><td>`C-c C-u`</td><td>(**reb-force-update**)</td><td>Force an update in the RE Builder target window without a match limit.</td></tr>
<tr><td>**Copy Regular Expression to kill ring.**</td><td>`C-c C-w`</td><td>(**reb-copy**)</td><td>Copy current RE into the kill ring for later insertion.
☝It also converts (where applicable) the expression to a string format suitable for use in Emacs Lisp source code.</td></tr>
<tr><td>**Convert string-syntax regexp to read-syntax**</td><td colspan="3">Emacs Lisp unfortunately does not have raw strings. This means that when writing a regex in Emacs Lisp code, which accepts what Emacs calls the read-syntax, you need to escape the double quote character (to allow the “ character be part of a string). The regex syntaxes also require escaping the backslash character. So to identify a backslash in a Elisp string regex you need to use 4 consecutive backslash. The following tool help convert a literal regexp into an elisp string regexp which provides escaping for Emacs Lisp purposes (as **reb-copy** , described above does).</td></tr>
<tr><td>**Prompt for regexp, insert quoted & escaped regexp string at point.**</td><td>`<f11> s x <SPC>`</td><td>(**pel-insert-regexp** &optional INSERT_BOTH)</td><td>Prompt for a regexp literal, insert corresponding quoted regexp at point.
• Converts what Emacs calls the ’string syntax’ into the Emacs ’read syntax’.
• When INSERT-BOTH argument is non-nil, insert both strings.
  • If INSERT-BOTH is a string, it is inserted between both strings, otherwise --> is inserted.
• At the prompt enter the literal regexp string, ie. a string with double quote, the capturing group parentheses and the alternative bar all escaped with a single backslash.
  • Example 1: when typing: `\(foo\|bar\)`
    • this text is inserted: `"\\(foo\\|bar\\)"`
  • Example 2: when typing: `\(foo\|bar—\"--\)`
    • this text is inserted: `"\\(foo\\|bar—\\\"--\\)"`
  • Example 3, using a `C-u` prefix argument for: `abc\S"gh`
    • this is inserted: `abc\S"gh —> "abc\\S\"gh"`
☛ Notice that you must not type the surrounding double quotes.</td></tr>
</table>

Description	Keystroke	Function	Note
Emacs Regular Expressions Syntax	The following rows describe **Emacs** regular expressions (which differ from other styles of regex) and tools to try them out. ⚠️ Be aware the the character classes like [:alpha:] **must** be used within a range. So if you want to express a space or tab, use **2** square brackets, as in: **[[:blank:]]**		
Emacs Regular expression syntax			

Special characters: `. * + ? [ ^ $ \`

Boundary anchors:

- `^` : beginning of {line, string, buffer}. Can be used at the beginning of the regexp or after **\(** or **\|**
- `$` : end of {line, string, buffer}
- `` \` `` : beginning of {string, buffer}
- `\'` : End of {string, buffer}
- `\b` : word boundary marker
- `\w` : any word character. Alternative: **[[:word:]]**
- `\W` : any non-word character. Alternative: **[^[:word:]]**

Basic:

- `.` : (a period) any single character except newline
- `\.` : one period
- `\` : either quotes a special character (such as $) or introduces special construct (see below).
- `\|` : Alternative

Expression Quantifiers - postfix operators:

- `?` : 0 or or 1 of the previous expression - greedy (greedy:= the longest valid match)
- `*` : 0 or more of the previous expression - greedy
- `+` : 1 or more of the previous expression - greedy
- `??` : 0 or or 1 of the previous expression - non-greedy (non-greedy := the shortest valid match)
- `*?` : 0 or more of the previous expression - non-greedy
- `+?` : 1 or more of the previous expression - non-greedy

Expression Quantifiers **- repetition** postfix operators**:**

- `\{n\}` : *n* repetitions. For example, **x\{4\}** matches the string `xxxx` and nothing else.
- `\{n,m\}` : between *n* and *m* repetitions: must match at least *n* times but no more than *m* times. If m is omitted there is no upper limit.

Boundaries:

- `\<` : beginning of word
- `\>` : end of word
- `\_<` : beginning of a symbol
- `\_>` : end of a symbol
- GNU extensions to regular expressions supported by Emacs include **\w**, **\W**, **\b**, **\B**, **\<**, **\>**, **\`** , **\'** (start and end of buffer)

Character alternative sets and Character Classes:

- `[ ]` : an alternative of characters, may include the following:
  - **Character range**: [c^1-c^2] where c^1 is the first character in the range and c^2 is the last, inclusive one. Example: [a-z] matches all lowercase characters (on case sensitive search).
  - **Inside alternative sets** the following characters or expressions can be used:
    - `^` : complements the set (ie: means that we want to match anything but what is in the set.
    - `[:C:]` : **character class** *C*, where *C* can be any of the following (**eg. [[:alnum]]:**):
      - `alnum` : any letter or digit
      - `alpha` : any letter
      - `ascii` : any of the 127 ASCII characters
      - `blank` : horizontal whitespace: a space or tab character
      - `cntrl` : any ASCII control character
      - `digit` : any digit character, same as [0-9]. **[-+[:digit:]]** matches any digit as well as '+' and '-'.
      - `graph` : any graphic character; everything except whitespace,
        ASCII and non-ASCII control characters, surrogates and code points unassigned by Unicode.
      - `lower` : lower-case letters. If case-fold-search is non-nil it also matches upper-case letters.
        Use `<f11> s m f` to toggle the value of this variable.
      - `multibyte` : any **multi-byte character**.
      - `nonascii` : matches any non-ASCII character
      - `print` : matches any printing character, either whitespace, or graphic character matched by [:graph:]
      - `punct` : any punctuation character. For multibyte character matches anything that has non-word syntax.
      - `space` : any character that has **whitespace syntax**. Note that syntax depends on the major mode.
      - `unibyte` : any **unibyte character**
      - `upper` : any upper-case letter, as determined by the current **case table**.
        If case-fold-search is non-nil, it also matches any lower-case letter!
      - `word` : any character that has **word syntax**.
      - `xdigit` : the hexadecimal digits: '0' through '9', 'a' through 'f' and 'A' through 'F'.
- **Special Characters**:
  - `\sC` : Match any character whose syntax table code is ***C***.
  - `\SC` : Match any character whose syntax table code is not ***C***.

  The **syntax table** code *C* cab be one of:

  - `SPC or -` : any whitespace : space, newline, tab, carriage return, formfeed, backspace
  - `w` : word constituents: normally all upper- and lower-case letters, and digits.
  - `_` : symbol constituents: extra characters used in variable, function, command names.
  - `.` : punctuation characters. There is none in Lisp. C has some.
  - `(` : open parenthesis. Support '(', '{', '['
  - `)` : close parenthesis. Support '), '}', ']'
  - `"` : string quotes. ☝️This is useful and important. In 'string-syntax' the double quote does **not** require escaping!
    Inside the 'string-syntax' regexp, you can use **\s"** and **\S"** .
    In read-syntax those become **\\s\"** and **\\S\"**
    ⚠️ The fact that a double quote is not needed in the string-syntax means that the string-syntax cannot be used inside Emacs Lisp source code. Don't be misled by its name! Emacs Lisp code only accepts read-syntax.
  - `\` : escape-syntax characters
  - `/` : character quotes
  - `<` : comment starters
  - `>` : comment enders
  - `!` : generic comment delimiters
  - `|` : generic string delimiters

Grouping:

- `\( … \)` : Capturing group
- `\(?: … \)` : Shy, non-capturing group, which cannot be referred to with \1 to \9. Is not counted in the group numbers
- `\1 to \9` : Insert text from group N
- `\#1 to \#9` : Insert text from group \N but cast as an integer (only useful in lisp forms)

Other:

- `\?` : prompt for user input
- `\#` : inserts a number incremented from 0
- `\&` : insert whole match string
- `\,(form …)` : uses an **Emacs Lisp form** with arguments. Use elisp form that take and return strings, such as the following examples:
  - `\,(upcase \2)` : uppercase capturing group 2
  - `\,(format "%.2f" \#3)` : Cast group 3 as number and format it as decimal with 2 decimal points.

New Line:

☝️When typing a regexp in read-syntax inside Elisp code string, you can represent the newline character with the **\n** character sequence. But, when typing it interactively at the prompt of a command you must insert the new line character by typing `C-q C-j` key sequence. The re-builder will accept the **\n** sequence when it uses the read-syntax, in string-syntax you must insert the newline with `C-q C-j`.

Unavailable sequences:

⚠️ The following do NOT work in Emacs, but there are alternatives, see above.

- `\d` : any digit : alternative: **[[:digit:]]**
- `\D` : any non digit character. Alternative: **[^[:digit:]]**

Description	Keystroke	Function	Note
PCRE support: pcre2el	PCRE (Perl Compatible Regular Expressions) is a popular regex syntax. 📦 This requires the pcre2el external package. 🔄 It is available when **pel-use-pcre2el** is **t** . • The pcre2el package provides the rxt-mode (RegeXp Translator or RegeXp Tools). According to its documentation the pcre2el package provides the following features: • convert Emacs syntax to PCRE • convert either syntax to rx, an S-expression based regexp syntax • untangle complex regexps by showing the parse tree in rx form and highlighting the corresponding chunks of code • show the complete list of strings (productions) matching a regexp, provided the list is finite • provide live font-locking of regexp syntax (so far only for Elisp buffers – other modes on the TODO list) This provides the commands listed below. 🚧 More work needed here to better document and integrate inside PEL.		
Toggle pcre-mode on/off. **In pcre-mode regexp use the PCRE syntax.** 🚧 Experimental function and experimental binding until it gets integrated better in PEL.	`<f11> s x P`	(**pcre-mode** &optional ARG)	Use emulated PCRE syntax for regexps wherever possible. Advises the 'interactive' specs of 'read-regexp' and the following other functions so that they read PCRE syntax and translate to its Emacs equivalent: • 'align-regexp' • 'find-tag-regexp' • 'sort-regexp-fields' • 'isearch-message-prefix' • 'ibuffer-do-replace-regexp' Also alters the behavior of 'isearch-mode' when searching by regexp.
Toggle rtx-mode on/off	`<f11> s x p`	(**rxt-mode** &optional ARG)	Toggle pcre2el rxt-mode. • With a prefix argument ARG, enable rxt-mode if ARG is positive, and disable it otherwise.
Explains regexp at point	`C-c / /`	(**rxt-explain)**	Pop up a buffer with pretty-printed 'rx' syntax for the regex at point. • Chooses regex syntax to read based on current major mode, calling 'rxt-explain-elisp' if buffer is in 'emacs-lisp-mode' or 'lisp-interaction-mode', or 'rxt-explain-pcre' otherwise.
Convert regexp to other syntax	`C-c / c`	(**rxt-convert-syntax**)	Convert regex at point to other kind of syntax, depending on major mode. • For buffers in 'emacs-lisp-mode' or 'lisp-interaction-mode', calls 'rxt-elisp-to-pcre' to convert to PCRE syntax. Otherwise, calls 'rxt-pcre-to-elisp' to convert to Emacs syntax. • The converted syntax is displayed in the echo area and copied to the kill ring; see the two functions named above for details.
Convert regexp at point to RX syntax	`C-c / x`	(**rxt-convert-to-rx)**	Convert regex at point or in region to RX syntax. If other found, prompt/ Chooses Emacs or PCRE syntax by major mode.
Convert regexp at point to RX syntax	`C-c / '`	(**rxt-convert-to-strings**)	Convert regex at point to RX syntax. Chooses Emacs or PCRE syntax by major mode.
Translate PCRE regexp to Emacs Lisp regexp and string to kill ring	`C-c / p e`	(**rxt-pcre-to-elisp** PCRE &optional FLAGS)	Translate PCRE, a regexp in Perl-compatible syntax, to Emacs Lisp. • Interactively, uses the contents of the region if it is active, otherwise reads from the minibuffer. Prints the Emacs translation in the echo area and copies it to the kill ring. • PCRE regexp features that cannot be translated into Emacs syntax will cause an error.
Query replace using PCRE syntax.	`C-c / %`	(**pcre-query-replace-regexp**)	Perform 'query-replace-regexp' using PCRE syntax. • Consider using '**pcre-mode**' instead of this function.
Translate PCRE regexp to RX syntax	`C-c / p x`	(**rxt-pcre-to-rx** PCRE &optional FLAGS)	Translate PCRE, a regexp in Perl-compatible syntax, to 'rx' syntax. • See 'rxt-pcre-to-elisp' for a description of the interactive behavior.
Return a list of strings matched by PCRE regexp	`C-c / p '`	(**rxt-pcre-to-strings** PCRE &optional FLAGS)	Return a list of all strings matched by PCRE, a Perl-compatible regexp. • See 'rxt-elisp-to-pcre' for a description of the interactive behavior and 'rxt-elisp-to-strings' for why this might be useful. • Throws an error if PCRE contains any infinite quantifiers.
Insert RX syntax for PCRE regexp in new buffer	`C-c / p /`	(**rxt-explain-pcre** REGEXP &optional FLAGS)	Insert the pretty-printed 'rx' syntax for REGEXP in a new buffer. • REGEXP is a regular expression in PCRE syntax. See rxt-pcre-to-elisp' for a description of how REGEXP is read interactively.
Insert RX syntax for Emacs Lisp regexp in new buffer	`C-c / e /`	(**rxt-explain-elisp** REGEXP)	Insert the pretty-printed 'rx' syntax for REGEXP in a new buffer. • REGEXP is a regular expression in Emacs Lisp syntax. See 'rxt-elisp-to-pcre' for a description of how REGEXP is read interactively.
Translate an Emacs Lisp regexp to PCRE	`C-c / e p`	(**rxt-elisp-to-pcre** REGEXP)	Translate REGEXP, a regexp in Emacs Lisp syntax, to Perl-compatible syntax. • Interactively, reads the regexp in one of three ways. • With a prefix arg, reads from minibuffer without string escaping, like 'query-replace-regexp'. • Without a prefix arg, uses the text of the region if it is active. • Otherwise, uses the result of evaluating the sexp before point (which might be a string regexp literal or an Emacs Lisp expression that produces a string). • Displays the translated PCRE regexp in the echo area and copies it to the kill ring. • Emacs regexp features such as syntax classes which cannot be translated to PCRE will cause an error.
Translate an Emacs Lisp regexp to RX syntax	`C-c / e x`	(**rxt-elisp-to-rx** REGEXP)	Translate REGEXP, a regexp in Emacs Lisp syntax, to 'rx' syntax. • See 'rxt-elisp-to-pcre' for a description of the interactive behavior and 'rx' for documentation of the S-expression based regexp syntax.
Get all strings that match an Emacs Lisp regexp	`C-c / e '`	(**rxt-elisp-to-strings** REGEXP)	Return a list of all strings matched by REGEXP, an Emacs Lisp regexp. • See 'rxt-elisp-to-pcre' for a description of the interactive behavior. • This is useful primarily for getting back the original list of strings from a regexp generated by 'regexp-opt', but it will work with any regexp without unbounded quantifiers (*, +, {2, } and so on). • Throws an error if REGEXP contains any infinite quantifiers.
Toggle regexp between Emacs Lisp systax and RX syntax	• `C-c / e t` • `C-c / t`	(**rxt-toggle-elisp-rx**)	Toggle the regexp near point between Elisp string and rx syntax.

Search & Replace — References

Topic & URL	Description
GNU Emacs - Searching and Replacement	GNU Emacs manual section describing search & replace features.
Regular Expression Help @ EmacsWiki	Some quick info on Emacs regular expression syntax.
Search - Incremental Search - Emacs Wiki	Large list of commands and key bindings. Also contains links to several other pages describing search modes, Icicle, etc..
Replace - GNU Emacs Manual - Replacement Commands	
Replace - ErgoEmacs - Emacs: Find and Replace Commands	Quick view of what's available by default.
Replace - How do I "M-x replace-string" across all buffers in emacs?	Some info here using ICycle.
Emacs Regular Expression Syntax	
Emacs Regular Expression Syntax @ GNU Emacs Manual	Reference for the Emacs Lisp regexp syntax
Regular Expression @ Emacs Wiki	Also describe the Emacs regexp syntax. Less dry. More examples.
Replace Regexp with Lisp Expressions @ Emacs Wiki	Describes the power of Emacs regexp in **replace-regexp** with ability to use embedded lisp code. Several examples.
Emacs Crash Regexp @ Emacs Wiki	More examples using **query-replace-regexp** which query before any change.
Multiline Regexp @ Emacs Wiki	
Searching in directory tree	
Is there a way to use query-replace from grep/ack/ag output modes?	This page describes several packages and functions to perform directory tree searches.
Regular Expressions & re-builder	
re-builder.el	Emacs built-in regular expression builder mode code.
re-builder: the Interactive regexp builder, @ Mastering Emacs by Mickey Petersen	A great little article on the various regexp syntaxes supported by the re-builder and how to change them.
Re Builder @ Emacs Wiki	
Why do regular expressions created with the regex builder use syntax different from the interactive regular expressions?	
re-builder: the Interactive regexp builder	
Search at Point	
"super star" or find the word under the cursor equivalent in emacs	Search at point with "M-s ."
Thing at point @ Emacs Wiki	Describes functions to retrieve text elements at point
The built-in regex-opt.el library	The built-in regex-opt package helps creation of simple regular expression strings.
Regexp Opt @ EmacsWiki	Quick description of regex-opt capabilities.
The built-in rx.el library	The rx macro converts an easy-to-read s-expression description of a regex into a regular expression
rx @ EmacsWiki	A quick overview of the idea behind rx. Also shows a macro that extends it.
Exploring Emacs Rx Macro from Francis Murillo	A more extensive presentation of rx with several examples.
Other Regular Expression Emacs Lisp Libraries	
xr - converts regex to structured rx form	Converts a string regular expression into the rx notation S-Exp form. Usefull to understand complex regex in Emacs Lisp source code.
pcre2el	As described in its overview: " `pcre2el' or `rxt' (RegeXp Translator or RegeXp Tools) is a utility for working with regular expressions in Emacs, based on a recursive-descent parser for regexp syntax."
visual-regexp	Useful library that provides commands to show regex matches in search and replace operations.
visual-regexp-steroid	Extends visual-regexp to bring simpler regex to Emacs commands. It supports both Python and pcre2el. It requires Python installed.
regex-tool	Tool using frame to test Emacs regular expressions.