↖ Topic Index

Search and Replace

Description	Keystroke	Function	Note
Emacs Search & Replace ○ Help & Customize • Search settings ○ Search pre-defined text • Non incremental search • Word search • combined balanced exp ○ Incremental search • symbol isearch • commands during isearch ○ occur search • fuzzy search • iedit mode ○ iedit navigation ○ unconditional replace • query replace • regexp replace in dir tree files ○ Using Projectile • Interpret regexp with xr • regexp lint ○ Regular expressions syntax ○ regexp-tool, • easy-escape, re-builder • PCRE support Last updated on:			Emacs provides several very powerful search and replace mechanisms described in this table. It supports: • Literal and **regular expression** , incremental and non-incremental search and replace in one or several buffers and files. • Search a set of files. See: **Grep** • Search a set of specified files in a directory. See **Dired** • Search & replace in the files of a specific project. See **Projectile** • With the **Xref** mechanisms, you can also jump to the definitions of code elements. Emacs provides its own regex search engine. Several external packages provide extensions: **visual-regexp** : activated when **pel-use-visual-regexp** is **t**. It enhances it by showing matches in the buffer while you type the regexp. It shows both the match in original text and its replacement. **visual-regexp-steroids** : activated when **pel-use-visual-regexp-steroids** is **t**. Allows other rexgexp engines: pcre2el and Python. There's also the following related external packages: **cexp** (balanced expression match) available when **pel-use-cexp** is **t**. See example of using cexp to match balanced parenthesis **easy-escape** (simplifies escaping) available when pel-use-easy-escape is t. **fzf.el** uses **fzf command line utility** available when **pel-use-fzf** is **t**. Perform interactive fuzzy search. See **fzf manual**. **pcre2el** (**PCRE2 regexp**) available when **pel-use-pcre2el** is **t** **xr** (regexp parser & analyzer) available when **pel-use-xr** is **t** And the following iSearch extension packages: **isearch-mb** (control isearch from minibuffer) available when **pel-use-isearch-mb** is **t** or **use-from-start** (to activate the Emacs starts) 2025-12-15
Open this PDF file. See also: **Help/Info**	`<f11> s <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **Search/Replace** 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.
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 (which uses the **C-s** binding). • 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> s <f2>` to open the PEL customize group that holds these user options. • Set the **pel-initial-search-tool** user option to select which search tool is used when Emacs starts.
Customize PEL basic search support	`<f11> s <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL basic search support. • When a prefix argument (like **C-u**) opens the buffer inside another window.
Customize PEL Regular Expression support	`<f11> s x <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL regular expression tool support. Select default regexp engine with **pel-initial-regexp-engine** user option.
Customize Search Tools			The following commands provide access to the customization groups that control the behaviour of the various search tools.
Customize Emacs basic search mechanism	`<f11> s <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Search support: isearch, anzu, iedit, easy-escape, fzf, swiper.
Customize Emacs Regular Expression support	`<f11> s x <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs regular expression support: rxt, re-builder, visual-regex.
Select Search Tool			Use the following commands to show the currently active search tool or to change it.
Show all search settings	• `<f11> s ?` • `<f11> ? s`	(**pel-show-search-status** &optional APPEND	Display search status and configuration inside special help info buffer: the name of the search tool used, the regular expression tool used, the search case settings used. Access to relevant user-options. With. optional APPEND argument append to buffer.
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 isearch-forward command to **C-s**. PEL provides the ability to activate the following search extension minor modes: • Anzu activated by **pel-use-anzu** user option, provides a match count in the mode line when searching with iSearch. • Swiper activated by **pel-use-swiper** user option, shows a list of matching lines in the mini-buffer. **Emacs Tutorial 17 - Searching with swiper!** • Use `<f11> s <f2>` to open the PEL search customize group and set **pel-initial-search-tool** user option to identify which tool is used when Emacs starts. Use or `<f11> s ?` to see state and access the user-option. Both default ISearch and Swiper are very useful in different scenarios.
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. Needs more work.
newlines in search and replace			**New line in search and replace**: In Emacs search and replace queries use **C-q C-j** to identify newline characters. • Several editors use the C string syntax "\n" to identify the newline character. Emacs does **not** use it in search and replace queries.
Control/Query how Search Operates See also: • **Text Modes**			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. The concept of "words" can be modified to include or exclude underscores and hyphens. It supports: • **superword-mode** that treats words separated by hyphen and underscores as a single entity, useful for programming languages using snake_case like C, C++, Erlang. • **subword-mode** that treats sections of camelCase and PascalCase as distinct words. Useful when editing portions of these longer symbols. PEL provides the ability to activate these modes automatically for various major modes by identifying the major modes in the following user options: • **pel-modes-activating-superword-mode** • **pel-modes-activating-subword-mode** 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. If too long look in the Message buffer.
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, t, no-yanks • **nil**: upper case don't force case sensitivity. **t**: upper case force case sensitivity • **not-yanks** : upper case force case sensitivity, 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 prefix argument ARG, enable Subword mode if ARG is positive, disable it otherwise. • Use `<f12> M-b` key for 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 prefix argument ARG, enable Superword mode if ARG is positive, disable it otherwise. • Use `<f12> M-p` key for modes for snake_case (Emacs Lisp, C, C++, Erlang, Python, etc…)

Description	Keystroke	Function	Note
Specialized Search/Move See also: **Navigation**	PEL provides a set of convenience/specialized search/navigation commands that move to pre-defined searched strings.		
Move point to next/previous two consecutive spaces	• **<f11> s SPC** • **M-g M-SPC**	(**pel-search-two-spaces** BACKWARDS)	Move point forward to next location of 2 consecutive space characters. • With any argument: move backward to previous location of 2 consecutive spaces.
Move point to next/previous empty line	• **<f11> s RET** • **M-g M-RET**	(**pel-search-empty-line** BACKWARDS)	Move point forward to the next empty line. • With any argument: move backward to previous empty line.
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:
• **text in marked region or**
• **word taken at point,**
• **forward from top *or***
• **backward from bottom of:**
• **current or windows specified by argument number**
 	
★★
 	
See also:
• **Key-Chords**
• **Keyboard Macros**
 	
Notes:
• Search word at point or marked area.
• Supports toggling the word mode when grabbing word at point.
• On search failure, does not move point even when searching inside another window.
• On search success:
  • Captures string searched:
  • allow repeating that search with **C-s** or **C-r**
	• **<f11> s w .**
	• **M-<f5>**
	• **.;**
		(**pel-search-word-from-top** &optional N)	Search for text in marked region or word at point from top/bottom of buffer of window identified by the number of non-dedicated windows and by the numeric argument N.
			• A numeric argument is composed with the Meta key prior to the command:
			• For example, to search a word in the buffer of window located at the right of the current one, position the point on the word to search and type one of the following key sequences:
			M-6 <f11> s w . or **M-6 M-<f5>**
			With PEL, the **.;** key-chord is available when **pel-use-key-chord** is non-nil.
			Command numeric prefix is available with the key-chord binding. See **Key-Chords**
			
			⚠ Inside keyboard macros the key chords do not work well. Use the **<f11> s w .** or the **M-<f5>** keystroke when recording keyboard macros.
			
			• **Search direction**:
			• If there is only one window: search from the top of current buffer.
			• If there is 2 non-dedicated windows, the behaviour depends on the value of the **pel-search-from-top-in-other** user option:
			  • if **pel-search-from-top-in-other** user option is nil (the default) : search from the top of current buffer unless a numeric argument is specifying another window (see below).
			  • if the **pel-search-from-top-in-other** user option is t, search from the top of the ***other*** window unless a numeric argument 3 or 5 is specified, in which case it searches from the top of the current buffer.
			• If there are 3 or more non-dedicated windows search into the buffer of the window identified by the numeric argument N (see below).
			  • 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 in **[10..18] range**: temporary toggle **subword-mode** to grab the current buffer word, use the N-10 value to identify the window to search.
			  • If N is in **[20..28] range**: temporary toggle **superword-mode** to grab the current buffer word, use the N-20 value to identify the window to search.
			  • In several major modes (but not all) using **superword-mode** allows you to grab complete identifiers (function names, variables that use embedded underscore or hyphen in their names).
			  • 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 and search in the current buffer, type the command with N in the 20 to 28 range. To toggle superword-mode and search in window above, use: **M-28 <f11> s .**
			  • If N is in **[30..38] range**: temporary activate **superword-mode** to grab the current buffer word, use the N-30 value to identify the Customize buffer window to search and transform the searched string to the Customize buffer title. For example, if you grab the text "pel-use-ido" the searched string is transformed to "Pel Use Ido: " . Use **M-31 <f11> s .** to search for the customization entry for a specific user-option inside the current customization buffer.
			  Useful when reading a description in a customization buffer that refers to a user-option and you want to move point to that user option.
			• 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.
			• On search failure: 1) does not move point nor change window, 2) Throw an error signal (flash or beep depending on setting): this allows using this command inside keyboard macros: a keyboard macro with it will be interrupted on failed search.		

Search forward • Basic forward search: • repeat using: **<F5><up>**	**<f11> s f**	(**search-forward** STRING &optional BOUND NOERROR COUNT)	Search forward from point for STRING. ⚠ Lax Search is not supported. • Set point to the beginning of the occurrence found. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'.
Search backward • Basic backward search: • repeat using: **<F5><up>**	**<f11> s b**	(**search-backward** STRING &optional BOUND NOERROR COUNT)	Search backward from point for STRING. ⚠ Lax Search is not supported. • Set point to the beginning of the occurrence found. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'.
Search regexp forward • Basic forward regexp search: • repeat using prompt history	**<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 • Basic backward regxp search • repeat using prompt history	• **<f11> s x b** • **M-R**	(**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 • Captures string searched, • search again with **C-s** or **C-r**	• **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 • Basic search: • repeat using prompt history	• **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 • repeat using prompt history	**<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 • repeat using prompt history	• **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)
Combined expression regex search	**<f11> s c**	(**cexp-search-forward** CEXP &optional BOUND NOERROR COUNT)	Search for combined regular and balanced expression CEXP. • The syntax of CEXP is almost that of a regular expression with the exception that the string **\!(** introduces a balanced expression and **\!)** closes a balanced expression. • The matched balanced expressions and the matches for the regular expressions before, in between, and after the sexps appear in the match data. • Regular expression braces **\(** and **\)** may not include balanced expressions. On the other hand balanced expressions may include regular expressions with groups. • The optional parameters BOUND, NOERROR, AND COUNT work like for 'search-forward'. Requires the **cexp external package**. PEL download & activates it when **pel-use-cexp** user-option is **t**. See example of using cexp to match balanced parenthesis on StackExchange

Description	Keystroke	Function	Note
Incremental Search (ISearch) See also: • You have no idea how powerful search is! • Customize Showing match count ➨			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> s <f2>** to customize PEL controlled search. • On Emacs >= 27 set the **isearch-lazy-count** user-option to t to show match count during isearch. The **Anzu** activated by **pel-use-anzu** does something similar but not only isearch. For Emacs >= 27 PEL automatically activates **isearch-lazy-count** when pel-use-anzu is nil.

ISearch - forward
 	
• **Incremental**
  • **literal search**
  • **regexp search**
• Captures string searched,
• search again with **C-s** or **C-r**
• When **anzu** is used, the modeline shows the match count.
	• **C-s**
	• ⌘-f
		(**isearch-forward** &optional REGEXP-P NO-RECURSIVE-EDIT)	Do incremental search forward: start or continue a search.
			📦 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. ⌘-f is always mapped to isearch-forward.

			• With a prefix argument, do an **incremental regular expression search** instead, something like:
			  • **C-u 1 C-s** or **M-- C-s** or, with PEL: **C-- C-s**
			  • **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**

Perform Swiper search: interactive search with an overview list	**C-s**	(**swiper** &optional INITIAL-INPUT)	Perform a Swiper text search. In a minibuffer: show several matches as they are being typed. 📦 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.
			• 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**".

ISearch - backward
 	
• **Incremental**
  • **literal search**
  • **regexp search**
• Captures string searched,
• search again with **C-s** or **C-r**
• When **anzu** is used, the modeline shows the match count.
	C-r	(**isearch-backward** &optional REGEXP-P NO-RECURSIVE-EDIT)	Do incremental search backward: start or continue a search.
			📦 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.

			• With a prefix argument, do an **incremental regular expression search** instead; something like:
			  • **C-u 1 C-r** or **M-- C-s** or, with PEL: **C-- C-r**
			  • **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**

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**.
Constrained isearch/iedit **Select isearching in code, comment, strings or everywhere.**			PEL provides the ability to apply constraints to searches performed by search and **iedit** (which uses isearch): search in code, comment , or string.
	<f11> s ,	(**pel-isearch-in**)	Apply or remove a constraint to the isearch (and **iedit**) done in the current buffer. Prompts to select from: search everywhere, in code only, in comments only, in strings only, in code and comments, in code and strings. Update the isearch prompt according to the selected constraint.
minibuffer control of iSearch			📦 **isearch-mb** available when **pel-use-isearch-mb** is **t** or **use-from-start**, allows editing search string directly in minibuffer, without special commands.
Toggle iseach-mb mode global minor mode	**<f11> s i**	(**isearch-mb-mode** &optional ARG)	Toggle isearch-mb global minor mode: allow edit of search string in minibuffer. • Use with iSearch, iSearch with Anzu. ARG positive: activate, negative: de-activate.
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 • Grab word at point with **C-w** • Captures string searched, • search again with **C-s** or **C-r**	• **M-s .** • **<f11> 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 **C-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 • Grab word at point with **C-w** • Captures string searched, • search again with **C-s** or **C-r**	• **M-s _** • **<f11> 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 • Grab word at point with **C-w** • Captures string searched, • search again with **C-s** or **C-r**	• **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. • After entering the prompt type **C-w** to capture the word(s) at point for the search
Before typing any text to search: **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.
During ISearch Type **C-j** to search for end of line. **D U R I N G**			During the execution of the incremental search, several aspects can be modified by the following commands: • the text searched can be modified, taken from various places, • the way the search is done, case sensitivity, word or symbol searches... • the search direction, or specifying the very first or last instance in the buffer. Right after typing the incremental search command you can type the following characters to modify or repeat the search.
Stop the incremental search	**RET**		Pick found text. Stop current search and leave cursor right after the found text.
	C-g		Aborts current search and return point to original location.
Show key bindings available during isearch	• **C-h b** • **<f1> b**	(**describe-bindings** &optional PREFIX BUFFER)	Show all key bindings available while performing interactive search.
Show isearch command information	• **C-h m** • **<f1> m**	(**describe-mode** &optional BUFFER)	Show information about the currently used interactive search command. That also lists some of the key bindings.

	Description	Keystroke	Function	Note
ISEARCH COMMANDS	**Repeat/reverse**	Repeat last search, reverse the direction.		
	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 iSearched 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**
	Yank text to iSearch	While performing a search you can issue the following commands to modify the searched string text, grabbing text from current location.		
	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 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.
	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.
	Add text up until next specified character (Emacs >= 27.1)	C-M-z	(**isearch-yank-until-char** CHAR &optional ARG)	Pull everything until next instance of CHAR from buffer into search string. Prompts for CHAR. • If optional ARG is non-nil, pull until next ARGth instance of CHAR. This is often useful for keyboard macros, for example in programming languages or markup languages in which CHAR marks a token boundary.
	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.
	iSearch first/last	While performing a isearch the following commands search for the first or last string in the buffer. • With Emacs >= 28.1, you might want to use '**isearch-allow-motion**' instead of these 2 commands. See below.		
	Go to first occurrence in buffer (Emacs >= 27.1)	M-s M-<	(**isearch-beginning-of-buffer** &optional ARG)	Go to the first occurrence of the current search string. • Move point to the beginning of the buffer and search forwards from the top. • With a numeric argument, go to the ARGth absolute occurrence counting from the beginning of the buffer. To find the next relative occurrence forwards, type **C-s** with a numeric argument.
	Go to last occurrence in buffer (Emacs >= 27.1)	M-s M->	(**isearch-end-of-buffer** &optional ARG)	Go to the last occurrence of the current search string. • Move point to the end of the buffer and search backwards from the bottom. • With a numeric argument, go to the ARGth absolute occurrence counting from the end of the buffer. To find the next relative occurrence backwards, type **C-r** with a numeric argument.
	iSearch Motion (Emacs >= 28.1)	With Emacs >= 28.1, with the '**isearch-allow-motion**' user-option set to 1 (on), you can use the following single keys to perform quick navigation searches. These include the above 2 commands plus 2 more. These commands, however, do not accept arguments like their counterparts above. ☝ PEL automatically sets '**isearch-allow-motion**' user-option set to t (on) for Emacs >= 28. Since this simplifies navigation.		
	Go to first occurrence in buffer (Emacs >= 28.1)	M-<	(beginning-of-buffer)	Go to the first occurrence of the current search string. • Move point to the beginning of the buffer and search forwards from the top.
	Go to last occurrence in buffer (Emacs >= 28.1)	M->	(end-of-buffer)	Go to the last occurrence of the current search string. • Move point to the end of the buffer and search backwards from the bottom.
	Search backward from top of window (Emacs >= 28.1)	• M-v • <PgUp>	(scroll-down-command)	Search backward from the top of the *window* (the portion of the buffer currently visible).
	Search forward from bottom of window (Emacs >= 28.1)	• C-v • <Pgdn>	(scroll-up-command)	Search forward from the bottom of the *window* (the portion of the buffer currently visible).
	Modify iSearch mode	While performing a isearch 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.
	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).
	Use occur search	While performing isearch you can start an occur search for it.		
	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.
	Start query replace	While performing a isearch the following commands start a query replace. • To **replace char at point**, do: **C-s**, **C-M-y** then **M-%.** To **replace word at point**, do: **C-s**, **C-w** then **M-%** • To **replace line at point**, do: **C-s**, **C-y** then **M-%**		
	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. You can repeat the middle command to include several chars, words or lines. ☝ When prompted for replacement, **M-p** retrieves the original text that you can then modify. • Type **C-r** to start **recursive editing** during the query replace operation.
	Start query replace regexp	• C-M-% • <f11> s x i • C-c Q	(**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. ☝ PEL provides direct access to the command via **<f11> s x i.** • If **pel-bind-keys-for-regexp** user-option is t, PEL adds the **C-c Q** key binding.

Description	Keystroke	Function	Note
Occur Search ★★ See: **Searching & Editing in Buffers with Occur Mode** ⚠ All occur searches are done in **buffers**, not files! ★★ Edit source buffer	The results are shown inside an ***Occur* buffer** which supports the following commands: • **<RET>** visit corresponding position in the searched buffer **C-c C-c** Visit corresponding position in searched buffer. • **C-o** display the match in other window (but does not select it) • **< , >** go to the beginning and end of the buffer • **n , p** Next and previous match. **Emacs >= 28** • **l** (Lower case L) Center buffer where found text is located. **Emacs >= 28** • **e** buffer enters the **Occur Edit Mode** which allows edits in both buffers simultaneously via edits in the *Occur* buffer. • **g** revert the buffer, refreshing the search results • **q** Quit occur search: close *Occur* buffer. • **list-matching-lines-default-context-lines** user-option controls the # of contextual lines around the match		
List all matching occurrences of regexp in current buffer	`M-s o`	(**occur** REGEXP &optional NLINES)	• Prompts for a regexp to search. • Use numeric prefix to specify n lines of context in result (defaults to **list-matching-lines-default-context-lines** see above) • Can use `M-n` and `M-n` at prompt to recurse previous search regexp strings. • `M-s o` can be used during an incremental search.
Occur search in selected buffers	`<f11> s O` `M-s /`	(**multi-occur-in-matching-buffers** BUFREGEXP REGEXP &optional ALLBUFS)	Show all lines matching REGEXP in buffers specified by BUFREGEXP. • Prompts for a regular expression that identifies files, then one for the text to search. • Normally BUFREGEXP matches against each buffer's visited file name, but if you specify a prefix argument, it matches against the buffer name. • For example to occur search in all .py files, select the buffers with **\.py$**
Occur search in selected files	`<f11> s o`	(**multi-occur** BUFS REGEXP &optional NLINES)	Show all lines in buffers BUFS containing a match for REGEXP. • This function acts on multiple buffers; otherwise, it is exactly like 'occur'. When you invoke this command interactively, you must specify the buffer names that you want, one by one.
Occur search in all buffers of same mode	• `<f11> s M-o` • `M-s m`	(**pel-multi-occur-in-this-mode**)	Perform an occur search in all buffers in the same major mode as the current buffer. • *Credits: Mickey Petersen*
Occur search in all buffers visiting files (or all buffers)	`M-s /`	(**pel-multi-occur-in-all** REGEXP &optional ALL)	Perform an occur search in all file-visiting buffers. • With a prefix argument (such as **C-u** or any numeric argument) search **all** buffers.
Search for occurrence of text in Σ Projectile project buffers	`<f8> o`	(**projectile-multi-occur** &optional NLINES)	Do a 'multi-occur' in the project's **buffers**. • With a prefix argument, show NLINES of context.
During Occur Search ★★	☝ Navigate through occurrences with commands issued from the original, possibly multiple, buffer(s). No need to be inside the *Occur* buffer.		
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-edit mode	`C-c C-c`	(**occur-cease-edit**)	Exit the **occur-edit** mode from within the *Occur* buffer or incremental search via `M-s o`
Fuzzy Finders Search See: Σ **File-mngt**, **fzf manual**, **fzf search syntax**	The **fzf command line utility** is a very fast fuzzy file finder that can be used within Emacs via the **fzf.el** emacs front-end. It can be used with grep, **ripgrep** or other search tools. To use it inside Emacs, you must: • 1) install and configure the **fzf command line utility**, and 2) 📦 Use the **fzf.el** external package 🔄 activated by **pel-use-fzf**		
Search current buffer with fzf	`<f11> s z`	(**fzf-find-in-buffer**)	Fuzzy search the current file-visiting buffer. Move point to the selected line.
iEdit mode ★★ ☝ To limit searches to code, comment, or strings, PEL provides **pel-isearch-in** , bound to: `<f11> s ,`	iEdit Mode - **Edit multiple symbols in a function or region in the same way simultaneously** 📦 Requires the **iedit** external package. 🔄 PEL downloads, installs and activates it when either **pel-use-iedit** or **pel-use-lispy** user options is set to **t**. 👀 PEL extends the iedit-mode slightly and uses different key bindings for some keys whose global mode meaning is sometimes useful when point is inside a iedit-mode highlighted area. Since most iedit-mode key bindings use key bindings that require holding the Meta and the Shift keys, PEL replaces the use of the `<tab>`, `<backtab>` and `M-;` keys by `M-S-<fx>` function key bindings. • You can force the use of the default iedit-mode keys by turning **pel-iedit-use-alternate-keys** user option off.		
Toggle iedit mode See also: • Σ **Cursor** • Σ **Highlight** • Σ **Key-Chords** • See also Σ **Rectangles** for rectangle editing supported by **iedit**. Notes: • Select only occurrences inside current function by using the numerical prefix 0 . • To search and select only inside code, comment, string or combinations, first restrict the search with **pel-search-in**, bound to `<f11> s ,` ☝	• `C-;` • `<f11> e` • `<f11> h e` • `<f11> m e`	(**iedit-mode** &optional ARG)	Toggle iEdit mode: **edit all symbols in scope or region simultaneously**. When turning it on: • With **C-u** prefix, highlight last highlighted text of current buffer • With **C-u C-u** prefix highlight text last highlighted in any buffer. • With numerical argument **0**, only highlight symbols in current function. • With numerical argument **1**, highlight current symbol only. • If region is active, highlight symbols inside region only. With **C-u** prefix: outside of region • This command behaves differently, depending on the mark, point, prefix argument and variable 'iedit-transient-mark-sensitive'. • With iEdit mode, all the occurrences of the current region in the buffer (possibly narrowed) or a region are highlighted. If one occurrence is modified, the change are propagated to all other occurrences simultaneously. • If region is not active, 'iedit-default-occurrence' is called to get an occurrence candidate, according to the thing at point. It might be url, email address, markup tag or current symbol(or word). • Can switch iEdit mode from **isearch** mode directly. The current search string is used as occurrence. Highlights all current search string occurrences. • **With a universal prefix argument, the occurrence when iEdit mode is turned off last time in current buffer is used as occurrence.** This is intended to recover last iEdit mode which is turned off. If region active, iEdit mode is limited within the current region. • With repeated universal prefix argument, the occurrence when iEdit mode is turned off last time (might be in other buffer) is used as occurrence. If region active, iEdit mode is limited within the current region. • With digital prefix argument 1, iEdit mode is limited on the current symbol or the active region, which means just one instance is highlighted. This behavior serves as a start point of incremental selection work flow. ⚠ iedit-mode may break key-chords commands from being detected: disable and re-enable the keychord mode with `<f11> M-K`. ⚠ Both iEdit and Flyspell use the `C-;` key as their default binding. PEL detects and reports that situation. ☆ PEL improves **iedit** in several major modes, temporarily changing the syntax table of those major mode to allow **iedit** to properly detect the symbols regardless of the characters surrounding them.
Customize iedit-mode	With point over text highlighted by iedit-mode, some extra key binding are activated, like the `<f11> <f2>` and the `<f11> <f3>` below, that open customization buffer for the PEL control of iedit-mode and for iedit-mode itself. For those, the background colour is a little darker.		
Customize PEL's iedit-mode	`<f11> <f2>`	(**pel-customize-pel-iedit**)	Open the customization buffer for the PEL control of the iedit-mode group.
Customize edit-mode	`<f11> <f3>`	(**pel-customize-iedit**)	Open the customization buffer for the iedit group.
iedit-mode help commands	Use the following commands to get extra help on the commands available when the edit-minor mode is active.		
Show edit-mode help	`<f1> <f1>`	(**iedit-help-for-help**)	Display iedit metahelp menu; select further option: • use **b** to show all edit-mode key binding or **m** for complete help.
Show keys used to modify occurrences	• `C-?` • `<f1> <f2>`	(**iedit-help-for-occurrences**)	Display a short message showing the iedit key bindings you can use to modify the occurrence text, toggle iedit buffering, move to first or last occurrence.
Show/hide occurence lines.	• `C-"` • `C-c C-o`	(**iedit-show/hide-occurrence-lines**)	Show or hide occurrence lines using invisible overlay.
Show/Hide context lines	• `C-'` • `C-c C-a`	(**iedit-show/hide-context-lines** &optional ARG)	Show or hide context lines. ☝ Show all instances selected by hiding all occurence lines. • A prefix ARG specifies how many lines before and after the occurrences are not hidden; negative is treated the same as zero. • If no prefix argument, the prefix argument last time or default value of 'iedit-occurrence-context-lines' is used for this time.

Description	Keystroke	Function	Note
iedit-mode navigation	Use the following commands to move point to other occurrence. PEL replaces several key bindings here. To use default key bindings (show in red) , set **pel-iedit-use-alternate-keys** user option off (nil).		
Move to previous occurence	• `S-<tab>` ✂ • `<backtab>` ✂ • `M-S-<f7>`	(**iedit-prev-occurrence**)	Move backward to the previous occurrence in the 'iedit'. • If the point is already in the first occurrences, you are asked to type another 'iedit-prev-occurrence', it starts again from the end of the buffer. ✂ With PEL, the backtab keys are not used for iedit, allowing their standard bindings. • To use them with iedit-mode, set **pel-iedit-use-alternate-keys** user option off (nil).
Move to next occurrence	• `<tab>` ✂ • `M-S-<f9>`	(**iedit-next-occurrence**)	Move forward to the next occurrence in the 'iedit'. • If the point is already in the last occurrences, you are asked to type another 'iedit-next-occurrence', it starts again from the beginning of the buffer. ✂ With PEL, the tab key is not used for iedit, allowing its standard bindings. • To use it with iedit-mode, set **pel-iedit-use-alternate-keys** user option off (nil).
Move to first occurence	`M-<`	(**iedit-goto-first-occurrence**)	Move to the first occurrence.
Move to last occurence	`M->`	(**iedit-goto-last-occurrence**)	Move to the last occurrence.
iedit-mode search area	Use the following commands to change the text area where occurrences are found. PEL replace some bindings for convenience (see ✂)		
Toggle selection of occurrence	• `M-;` ✂ • `M-S-<f8>`	(**iedit-toggle-selection**)	Select or deselect the occurrence under point. ☝When deselecting, if there was only 1 occurrence, iedit-mode is also turned off. ✂ With PEL, `M-;` is replaced by `M-<f8>` which does not clash with comment-dwim . • To use the default `M-;` key binding, set **pel-iedit-use-alternate-keys** user option off (nil).
Restrict searched area to current function	`M-H`	(**iedit-restrict-function** &optional ARG)	Restricting Iedit mode in current function.
Restrict searched area to current line	`M-I`	(**iedit-restrict-current-line**)	Restrict Iedit mode to current line.
Expand searched area backwards	`M-{`	(**iedit-expand-up-a-line** &optional N)	After start iedit-mode only on current symbol or the active region, this function expands the search region upwards by N line. N defaults to 1. If N is negative, collapses the top of the search region by '-N' lines.
Expand searched area forward	`M-}`	(**iedit-expand-down-a-line** &optional N)	After start iedit-mode only on current symbol or the active region, this function expands the search region downwards by N line. N defaults to 1. If N is negative, collapses the bottom of the search region by '-N' lines.
Expand searched area to previous match	`M-p`	(**iedit-expand-up-to-occurrence** &optional ARG)	Expand the search region upwards until reaching a new occurrence. If no such occurrence can be found, throw an error. With a prefix, bring the top of the region back down one occurrence.
Expand searched area to next match	`M-n`	(**iedit-expand-down-to-occurrence** &optional ARG)	Expand the search region downwards until reaching a new occurrence. If no such occurrence can be found, throw an error. With a prefix, bring the bottom of the region back up one occurrence.
Toggle case sensitivity of occurence matching	• `M-C` ✂ • `<f1> M-c`	(**iedit-toggle-case-sensitive**)	Toggle case-sensitive matching occurrences. ✂ With PEL, the default `M-C` key is replaced by `<f1> M-c`.
iedit-mode replacement	Use the following commands to replace all marked occurrences. These commands are available over a highlighted occurrence.		
Convert occurrences to lower case letters	• `M-L` • `M-c`	(**iedit-downcase-occurrences**)	Convert occurrences to lower case.
Convert occurrences to upper case letters	• `M-U` ✂ • `M-C`	(**iedit-upcase-occurrences**)	Convert occurrences to upper case. ✂ With PEL, default `M-U` key is replaced by `M-C`. This way `M-U` remains bound to pel-redo.
Replace text of occurrences	`M-R`	(**iedit-replace-occurrences** &optional TO-STRING)	Replace occurrences with STRING. Prompt for replacement string. • Note: instead of using this command the occurrences can also be edited using in place.
Blank occurences	`M-SPACE`	(**iedit-blank-occurrences**)	Replace occurrences with blank spaces.
Delete occurrences text	`M-D`	(**iedit-delete-occurrences**)	Delete occurrences.
Prefix occurrences with a number	`M-N`	(**iedit-number-occurrences** START-AT &optional FORMAT-STRING)	Insert numbers in front of the occurrences. • START-AT, if non-nil, should be a number from which to begin counting. • Format string specified by **iedit-increment-format-string** user option. • When called interactively with a prefix argument, prompt for START-AT and FORMAT.
Misc iedit commands	Other iedit-mode commands. These commands are available over a highlighted occurrence.		
Toggle edit buffering	`M-B`	(**iedit-toggle-buffering**)	Toggle buffering. • Buffering is intended to improve iedit's response time, when the number of occurrences is huge, which might slow down update of all occurrences on each key stoke. • When buffering is on, modification is only applied to the current occurrence and will be applied to other occurrences when buffering is turned off. • Therefore, if you experience slow update for a large number of occurrences, turn buffering on, make your modifications in the occurrence at point, then turn buffering back off to allow edit to update all other occurrences.
Apply last global modification	`M-G`	(**iedit-apply-global-modification**)	Apply last global modification.
Switch to multiple-cursors-mode	`M-M`	(**iedit-switch-to-mc-mode**)	Switch to 'multiple-cursors-mode'. So that you can navigate out of the occurrence and edit simultaneously with multiple cursors. 📦 Requires the multiple-cursors external package. 🛂 PEL activates it when **pel-use-multiple-cursors** is set to **t**.
Quit edit-mode	`C-g`	(**iedit-quit**)	Quit edit-mode. Must be typed while cursor is over a highlighted occurence character.

Description	Keystroke	Function	Note
Unconditional Replace	Non-interactive text replacement commands.		
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.
Unconditional regex replace ☝**replace-regexp** TO-STRING arg uses string syntax and accepts lisp expressions in parens prefixed with a single backslash. This is super powerful. See **examples in the Emacs Wiki.**	• `<f11> s x r` • `C-c r`	(**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 when **pel-bind-keys-for-regexp** is set to **t**. 📦 When **pel-use-visual-regexp** or **pel-use-visual-regexp-steroids** is set to t, PEL selects the **pel-replace-regexp** command instead of Emacs' **replace-regexp**. • **pel-replace-regexp** uses regexp engine previously selected by **pel-select-search-engine-regexp** (bound to `<f11> s S`): • **replace-regexp** • **vr/replace** • **vr/select-replace**
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: 𝕀 **Cursor**	• `<f11> s x M` • `C-c m`	(**vr/mc-mark** REGEXP START END) 📦 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 these commands 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 when **pel-bind-keys-for-regexp** is set to **t**.
Visual Regexp Search to multiple-cursors with engine selection See also: 𝕀 **Cursor**	`<f11> s x M-m`	(**vr/select-mc-mark**) 📦 Requires both visual-regexp-steroids & 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. ☝When prompted for replacement use M-p to retrieve the original text that you can then modify. • Type `C-r` to start **recursive editing** during the query replace operation.
Query Replace Regexp ☝**query-replace-regexp** TO-STRING arg uses string syntax and accepts lisp expressions in parens prefixed with a single backslash. This is super powerful. See **examples in the Emacs Wiki.**	• `C-M-%` • `<f11> s x q` • `C-c q`	(**pel-query-replace-regexp**) (**query-replace-regexp** REGEXP TO-STRING &optional DELIMITED START END BACKWARD REGION-NONCONTIGUOUS-P) (**vr/query-replace** REGEXP REPLACE START END) (**vr/select-query-replace**)	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 **pel-bind-keys-for-regexp** user option is set to **t**. 📦 When **pel-use-visual-regexp** or **pel-use-visual-regexp-steroids** is set to t, PEL selects the **pel-query-replace-regexp** command instead of Emacs' **replace-regexp**. • **pel-query-replace-regexp** uses regexp engine previously selected by **pel-select-search-engine-regexp** (bound to `<f11> s S`): • **query-replace-regexp** • **vr/query-replace** • **vr/select-query-replace**
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**.	
Replace text with regexp found in marked file(s) using Dired See also: 𝕀 **Dired**	Open a Dired buffer with `C-x d` ,mark the files and directories to search/replace into with `m`, then type `Q`	(**dired-do-find-regexp-and-replace** FROM TO)	Replace matches in all files and/or directories currently marked in Dired buffer. • For any marked directory, matches in all of its files are replaced, recursively. However, skip files matching **grep-find-ignored-files** & subdirectories matching **grep-find-ignored-directories** ☝REGEXP should use constructs supported by your local 'grep' command. • `.` : replace current and quit
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 all the rest and don't ask • `^` : back up to previous instance • `u` : undo last replacement • `U` : undo ALL replacements • `q` or `<RET>` : abort/exit query-replace • `E` : modify the replacement string		• `,` : replace & let me see result before moving on — Press **`SPC`** to move on. • `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 w/o replacing remaining matches in current buffer. Multi buffer QR Response.
Regexp replace text in all specified files under directory tree ★★ 🎛 **Customizable user-options** **Sub-commands:**	`<f11> s M-%`	(**pel-dirtree-find-replace** TEXT-RE NEW-TEXT ROOT-DIR FN-RE)	Search for regexp text in all files identified by regexp name under a directory tree root and replace that text by specified replacement. Backup original file (by default). • Prompt for text regexp, replacement text, root directory, file name regexp. Keeps entry history. • **pel-dirtree-replace-file-forbidden-dir-re**: list of regexp identifying directory base names that must be skipped. Defaults to **'("/\\.")** • **pel-dirtree-replace-files-is-verbose:** controls whether messages showing modified files names is shown. • **pel-dirtree-replace-file-backup-suffix**: controls whether a backup of modified files is made and the suffix appended to the file name. • **pel-dirtree-replace-file-newtext-is-fixedcase:** controls whether the text replacement is donees fixed case or follows text casing folding rules. • **pel-dirtree-replace-file-newtext-is-literal**: controls whether the replacement text is used literally or as an Emacs regexp (string format: **\1** ➡ first group).
Undo all replacements done by last pel-dirtree-find-replace	`<f11> s M-u`	(**pel-dirtree-replace-undo** &optional NO-PROMPT)	Undo all changes done by last 'pel-dirtree-find-replace' execution. Prompt to confirm before proceeding unless NO-PROMPT is non-nil. Interactively, use any prefix argument, like `C-u`.
List all files modified (and their backup) in a 𝕀 Dired buffer	`<f11> s %`	(**pel-dt-fr-changed-files-in-dired**)	Show all files changed by the last 'pel-dirtree-replaced-file' command (and their backup files if any) in a dired buffer. Use it to compare the original and new files & review the changes, or delete backups.
Change backup behaviour †	`<f11> s <f4> b`	(**pel-dt-fr-set-backup-suffix** NEW-SUFFIX)	Change backup suffix used by '**pel-dirtree-find-replace**': Modify the value of `pel-dirtree-replace-file-backup-suffix' in current session to change the behaviour of backup use by the command.
Toggle text replacement: fixed case or adjusted case †	`<f11> s <f4> c`	(**pel-dt-fr-toggle-fixedcase**)	Change behaviour of '**pel-dirtree-find-replace**' string replacement:whether the function performs a fixed case string replacement or adjust capitalization based on the replaced text
Toggle replacement text meaning: literal or regexp †	`<f11> s <f4> l`	(**pel-dt-fr-toggle-literal**)	Change behaviour of '**pel-dirtree-find-replace**' literal string replacement: whether the function performs a literal string replacement or interpret the new-text string as an Emacs regexp.

Description	Keystroke	Function	Note
Using Σ Projectile	📦 For the following commands Projectile mode must be activated first. PEL provides the following key binding to do it when **pel-use-projectile** user option is turned on: the key sequence: `<f11> <f8> <f8>`		

Search in Project Using Σ Projectile	• Searching in project buffers: projectile provides the multi-occur for project buffers, shown non the first row.
	• Searching in project files: projectile provides the following recursive-grep like search tools, they are listed starting on the second row.
	  • The first one searches inside buffers, not in files. That may be useful when looking for unsaved buffers or for special buffers.
	  • The last 2 require external packages and external command line utilities that must have been installed separately: ripgrep and ag.
	  • The ripgrep and ag searches are faster than the standard grep search.		

Search for occurrence of text in project buffers	`<f8> o`	(**projectile-multi-occur** &optional NLINES)	Do a 'multi-occur' in the project's **buffers**. • With a prefix argument, show NLINES of context. See Occur section above for navigation.
Search in project files with recursive grep	`<f8> s g`	(**projectile-grep** &optional REGEXP ARG)	Perform rgrep in the project. • With a prefix ARG asks for files (globbing-aware) which to grep in. • With prefix ARG of '-' (such as 'M--'), default the files (without prompt), to 'projectile-grep-default-files'. • With REGEXP given, don't query the user for a regexp.
Search in project files with ripgrep • **Rust (ripgrep) regex syntax**	`<f8> s r`	(**projectile-ripgrep** SEARCH-TERM &optional ARG)	Run a Ripgrep search with 'SEARCH-TERM' at current project root. • With an optional prefix argument ARG SEARCH-TERM is interpreted as a regular expression. 📦 Requires the projectile, ripgrep.el external packages as well as the ripgrep command line utility. PEL activates this command when **pel-use-projectile** is non-nil. But to make it work you must also set **pel-use-ripgrep** to **t**. Also note that the ripgrep command line utility must be installed manually.
Search in project files with ag • **PCRE regex syntax**	`<f8> s s`	(**projectile-ag** SEARCH-TERM &optional ARG)	Run an ag search with SEARCH-TERM in the project. • With an optional prefix argument ARG SEARCH-TERM is interpreted as a • regular expression. 📦 Requires the projectile, ag.el external packages as well as the ag command line utility. PEL activates this command when **pel-use-projectile** is non-nil. But to make it work you must also set **pel-use-ag** to **t**. Also note that the ag command line utility must be installed manually.
Replace in Project	Text replacement inside all project files.		
Replace test in project files	`<f8> r`	(**projectile-replace** &optional ARG)	Replace literal string in project using non-regexp 'tags-query-replace'. • With a prefix argument ARG prompts you for a directory on which to run the replacement.
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**, a regex parser and analyzer, 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.
Interpret Emacs Lisp regexp provided at prompt.	`<f11> s x X`	(**pel-xr-regxp**)	Prompt for regexp and print its interpretation in *regexp-eval* buffer. • Uses `xr-pp' to expand regexp in rx notation.
Lint Emacs Lisp regexp at point	`<f11> s x l`	(**pel-xr-lint-at-point** &optional FOR-FILE-MATCH)	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.
Lint Emacs Lisp regexp provided at prompt	`<f11> s x L`	(**pel-xr-lint** &optional FOR-FILE-MATCH)	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.

relint — Regular Expression Lint
See also: **𝔉𝔓𝔏 - Emacs Lisp**
	The following commands can be used to analyze the validity of the regular expressions inside Emacs Lisp code stored inside:
	  • the current Emacs Lisp buffer,
	  • an Emacs Lisp file or,
	  • all Emacs Lisp files inside a directory tree.
	• From the *relint* buffer press **g** to re-run the same checks.
	The package can also used in a script to analyze regular expressions using Emacs batch invocation.
	📦 Requires the relint external package. PEL installs and activates it when the pel-use-relint user-option is set to **t**.		

Lint regular expressions in current buffer	`<f11> s x M-l b`	(**relint-current-buffer**)	Scan the current buffer for regexp errors. ⚠ The buffer must be in emacs-lisp-mode.
Lint regular expressions in specified file	`<f11> s x M-l f`	(**relint-file** FILE)	Scan FILE, an elisp file, for regexp-related errors. • Prompts for Emacs Lisp file.
Lint regular expressions in specified directory	`<f11> s x M-l d`	(**relint-directory** DIR)	Scan all *.el files in DIR for regexp-related errors. • Prompts for the directory. • Scans directory tree: all Emacs Lisp files in the specified directory all all sub-directories , recursively.

Description	Keystroke	Function	Note

Emacs Regexp Syntax

Emacs Regular expression syntax

- **Backslash constructs** (elisp)
- **Backslash constructs** (emacs)

⚠ Of the above 2 manuals, the Elisp manual is more complete. The Emacs manual does not describe all available syntax.

The following rows describe **Emacs** regular expressions (which differ from other styles of regex) and tools to try them out.

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}
- `\=` : match empty string, but only at point. (This construct is not defined when matching against a string.)
- `\b` : word boundary marker
- `\B` : not word boundary marker
- `\w` : any word character. Alternative: **[[:word:]]**
- `\W` : any non-word character. Alternative: **[^[:word:]]**

Basic:

- `.` : (a period) any single character **except newline**. ☝To search for any character including newline use: `[^\0]`
- `\.` : 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: (GNU extensions to regular expressions supported by Emacs):

- `\<` : beginning of word
- `\_<` : beginning of a symbol
- `\w` : word-constituent character (as defined in syntax table)
- `\b` : empty string at beginning/end of word
- `\`` : start of string or buffer
- `\=` : match the empty string but only at point.
- `\>` : end of word
- `\_>` : end of a symbol
- `\W` : any character that is not word-constituent
- `\B` : empty string but not at beginning/end of word
- `\'` : end of string or buffer

Character alternative sets and Character Classes: ⚠ character classes like [:alpha:] must be used within a character alternative set. So if you want to express a space or tab, use **2** square brackets, as in: **[[:blank:]]**

- `[ ]` : set of alternative of characters, may include the following: *without escaping:* `'$'`, `'.'`. Include a `'-'` at end, `']'` at first, `'^'` anywhere but first.
  - **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. Can match a newline if not in 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 it.
      - **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.
    - `(` : matches the open "*parens*" characters identified by the syntax-table of the buffer major-mode; typically: '(', '{', '[', …
    - `)` : matches the close "*parens*" characters identified by the syntax-table of the buffer major-mode; typically: '), '}', ']',…
    - `"` : 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\"**

      ⚠ Because 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
  - `\cC` : Match any character whose category is ***C***. Some of the category chars: **a** : ASCII, **b** : arabic, **c** : chinese, **g** : greek, **l** : latin
  - `\CC` : Match any character whose category is not ***C***.

Grouping:

- `\( … \)` : Capturing group
- `\(?: … \)` : Shy, non-capturing group, which cannot be referred to with \1 to \9. Is not counted in the group numbers
- `\(?num: … \)` : Explictly numbered capturing group. No restriction on numbering. Several groups can have the same number, the last match wins.
- `\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 in 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. **See also this article.**

New Line, hard-tab and ASCII control character basic char syntax :

☝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**. This is the same for *most* ASCII control characters that have Emacs basic char syntax: **\a**, **\b**, **\t**, **\n**, **\v**, **\f**, **\r**, and **\e**

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:]]**

☝Use **describe-syntax** to see a description of the syntax-table of the current buffer.

- See: **Help/Info**
- **Syntax Table**
  - **Syntax Descriptors**
    - **Syntax Classes**
    - **Syntax Flags**

See Also (elisp)**:**

- **Backslash constructs**
- **Match Data**
  - **Replacing matched text**
  - **Simple Match Data Access**
  - **Access Entire Match Data**
  - **Saving/Restoring Match Data**

Description	Keystroke	Function	Note

Toggle easy-escape minor mode
 	
Simplify display of escape characters used in regexp strings
	<f11> "	(**easy-escape-minor-mode** &optional ARG)	Compose escape signs together to make regexps more readable.
			• When this mode is active, \\ in strings is displayed as a single \, fontified using 'easy-escape-face' and composed into 'easy-escape-character'.
			  • See **easy-escape Github page** for more information and an example, which includes:
			  • **"\\(\\|\\)" ↔ "(|)"**
			  • **"[\\t\\n]" ↔ "[\t\n]"**
			📦 Requires the easy-escape external package. PEL activates when the **pel-use-easy-escape** is set to **t**.
			• You can also identify major modes where it is activated automatically by setting the **pel-modes-activating-easy-escape** user-option.
			• Use **<f11> s <f2>** to open the relevant PEL customization group.
			🎛 If you find the distinction between the fontified double-slash and the single slash too subtle, try the following, customizing the following user-options in the easy-escape customization group (with PEL use **<f11> s <f3> 4** to open the easy-escape customization group):
			  • Adjust the foreground of '**easy-escape-face**'
			  • Set '**easy-escape-character**' to a different character.

Regex-tool	📦 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.		

Open the regex-tool	**<f11> s x T**	(**regex-tool**)	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

Force an update of regex-tool windows	**C-c C-c**	(**regex-tool-markup-text** &optional BEG END LEN)	Force an update of the regex-tool windows.
Quit regex-tool	**C-c C-k**	(**regex-tool-quit**)	Quit regex-tool and close its 3 windows, revert to the window layout used before it was used.
Change the regex-tool backend engine - select between Emacs and Perl.	**C-c <f2>**	(**pel-select-regex-tool-engine**)	Open the customize buffer to change **regex-tool-backend** user option. • Select between Emacs and Perl backend. • To close the customize buffer, type **q**. • **C-c C-c** forces an update of the regex-tool to rescan using the new backend.

re-builder:
 	
Emacs Regular Expression Builder
	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**		

Build regular expression interactively with re-builder ☝ This is a great way to learn Emacs regexp!	**<f11> s x B**	(**re-builder**)	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*. The syntax used by Emacs Lisp code: requires double escaping of backslashes. For example: `"\\(red\\|green\\)"`
	  • ***string***: Like ***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.
	☝ The syntax to express a single backslash is: string format: `"\\"` read format: `"\\\\"`		

Customize re-builder regular expression syntax	**<f11> s x M-B**	(**pel-reb-re-syntax**)	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.
Select re-builder regular expression syntax	• **C-c C-i** • **C-c <tab>**	(**reb-change-syntax** &optional SYNTAX)	Change the syntax used by the RE Builder. • Affects current session. Does not change customize default.
Change target buffer	**C-c C-b**	(**reb-change-target-buffer** BUF)	Change the target buffer and display it in the target window.
Toggle case sensitivity	**C-c C-c**	(**reb-toggle-case**)	Toggle case sensitivity of searches for RE Builder target buffer.
Enter/leave sub-expression highlight mode	**C-c C-e**	(**reb-enter-subexp-mode**)	Enter the subexpression mode in the RE Builder, which only highlights the specified capturing group of each match. • Type **0** to **9** to identify the group to highlight. Type **q** to exit that mode.
Move point to previous match	**C-c C-r**	(**reb-prev-match**)	Go to previous match in the RE Builder target window.
Move point to next match	**C-c C-s**	(**reb-next-match**)	Go to next match in the RE Builder target window.
Force update	**C-c C-u**	(**reb-force-update**)	Force an update in the RE Builder target window without a match limit.
Copy Regular Expression to kill ring.	**C-c C-w**	(**reb-copy**)	Copy current RE into the kill ring for later insertion. ☝ It also converts the expression to a read format suitable for use in Emacs Lisp source code.
Delete regexp	**C-c C-d**	(**pel-reb-erase-regexp**)	Replace currently used regular expression by an empty string.
Quit re-builder	**C-c C-q**	(**reb-quit**)	Quit the RE Builder mode.
Convert string-syntax regexp to read-syntax	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 does **reb-copy**, described above does).		

Prompt for regexp, insert quoted & escaped regexp string at point.	**<f11> s x <SPC>**	(**pel-insert-regexp** &optional INSERT_BOTH)	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"**
			☝ Note that you must not type the surrounding double quotes.

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.		

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.
pcre2el rxt-mode	The **pcre2el** minor mode must be activated for the local, buffer to activate the various commands that use the `C-c /` key prefix. ☝ You may want to automatically activate the **rxt-mode** for Perl buffers. (See **Perl**). • That can be done by customization. With PEL use the `<f12> <f2>` key sequence to open the PEL customization for the current major mode. • For example, add **rxt-mode** to the list of minor modes identified by the pel-cperl-activates-minor-modes to automatically activate **rxt-mode** in Perl buffers using the cperl-mode.		
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.
Do what I mean commands	The following commands try to detect what regexp syntax to use based on the current major mode.		
Explains regexp at point	`C-c / /`	(**rxt-explain**)	Pop up a buffer with pretty-printed 'rx' syntax for the regex in marked area. Prompts for one if nothing currently marked. • 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 'rxt-elisp-to-pcre' and 'rxt-pcre-to-elisp' 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.
Commands that work on PCRE regexp	The following commands take a PCRE regexp.		
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.
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.
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.
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.
Commands that work on Emacs regexp	The following commands take a Emacs regexp.		
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.