Search and Replace

Description	Keystroke	Function	Note
Control/Query how Search Operates	Emacs searches are by default "case folding" and does "lax space matching". searches where any case is matched unless the specified pattern contains at least one upper case letter. It also has different modes for words and symbols. The behaviour can be modified using some of the commands below.		
Show how search behaves in mini buffer	`<f11> s m ?`	(**pel-show-search-case-state**)	Describe the search case handling behaviour. • The information is shown in the mini-buffer.

Toggle case impact on search	`<f11> s m u`	(**pel-toggle-search-upper-case**)	Toggle case sensitivity behaviour of yank in search prompt.
			• Rotates the value of search-upper-case to:
			  • nil: upper case don't force case sensitivity
			  • t: upper case force case sensitivity
			  • not-yanks : upper case force case sensitivity, and lower case text when yank in search minibuffer.

Toggle 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.
Search Tools Selection (See also: ∑ Customize)	PEL supports several search tools that impact the way the **`C-s`** command operates. PEL supports the following search tools: • Emacs' default ISearch • 📦 Anzu, ISearch with match count : set **pel-use-anzu** to **t**. • 📦 Swiper search with overview match list : set **pel-use-swiper** to **t** 🎛 • Use **`<f11><f1> s`**, to customize the PEL completion group user options above. • Set the **pel-initial-search-tool** user option to select which search tool is used when Emacs starts. As soon as one of the extra search tool is activated via the corresponding pel-use- user option, PEL makes the following commands available to change the currently used search tool and to see which one is currently active.		
Select search tool to use	`<f11> s s`	(**pel-select-search-tool**)	Prompt user for search tool to use with **`C-s`**. Show new active one. • Emacs normally maps the search-forward command to **`C-s`**. • PEL provides the ability to activate the following tools that can be activated for searching: • 📦 The Anzu external package activated by **pel-use-anzu** user option. Anzu provides a match count in the modeline when search command is used. • 📦 The Swiper external package activated by **pel-use-swiper** user option. Swiper is not using isarch-forward; it shows a list of matching lines in the mini-buffer. • 🎛 Use the **`<f11> <f1> s`** command to open the PEL search customize group and set the **pel-initial-search-tool** user option to identify which tool is used when Emacs starts. See the ∑Customize table for more information. ☝ Being able to search using either Emacs default ISearch (see below) and Swiper helps as they are both very useful in different scenarios.
Show which search tool is currently used	`<f1> ? s`	(**pel-show-active-search-tool**)	Display the currently used search tool.
newlines in search and replace	⚠**New line in search and replace**: • Several editors use the C string syntax "\n" to identify the newline character. Emacs does **not** use it in search and replace queries. • In Emacs search and replace queries use **`C-q C-j`** to identify newline characters.		
Non-Incremental Search	The *normal* (non-incremental) search can be performed using the commands and keystrokes listed below. • They can also be invoked by typing <RET> right after the invocation of the incremental search commands (see below).		

Search for word taken at point from the top of current or specified window	`<f11> s .`	(**pel-search-word-from-top** &optional N)	Search word at point from top/bottom of buffer in window identified by N.
			• **Search direction**:
			  • If N is nil, 0 or larger, perform a search-forward from the top of the buffer in window identified by N.
			  • If N is negative: perform a isearch-backward from the bottom of the buffer in the window selected by the absolute value of N.
			• **Window selection**:
			  • If N is not specified, nil or 1: 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
			  • If N is 9 or larger: search in window below.
			
			• Explicitly selecting the minibuffer window, or a non-existing window is not allowed, and search is done in current window.
			• Searched word is remembered and can be used again to repeat an interactive search with **`C-s`** or **`C-r`**.
			• Position before searched word is pushed on the mark ring.
			☝ Using superword-mode allows you to search for function names in buffer for programming languages.

Search forward	`<f11> s f`	(**search-forward** STRING &optional BOUND NOERROR COUNT)	Search forward from point for STRING. • Set point to the beginning of the occurrence found. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'. • ⚠ Lax Search is not supported.
Search backward	`<f11> s b`	(**search-backward** STRING &optional BOUND NOERROR COUNT)	Search backward from point for STRING. • Set point to the beginning of the occurrence found. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'. • ⚠ Lax Search is not supported.
Search regexp forward	`<f11> s x f`	(**re-search-forward** REGEXP &optional BOUND NOERROR COUNT)	Search forward from point for regular expression REGEXP. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'.
Search regexp backward	`<f11> s x b`	(**re-search-backward** REGEXP &optional BOUND NOERROR COUNT)	Search backward from point for regular expression REGEXP. • Search case-sensitivity is determined by the value of the variable 'case-fold-search'.

Description	Keystroke	Function	Note
Word Search	A word search finds a sequence of words without regard for the type of punctuation between them. • The word search commands do not perform character folding and toggling lax whitespace matching have no effect on them. • However there are "lax" word searches that succeed on incomplete words, they are listed below.		
Incremental Search Word	• `M-s w` • `<f11> s w i`	(**isearch-forward-word** &optional NOT-WORD NO-RECURSIVE-EDIT)	Do incremental search forward for a **sequence of words**. • With a prefix argument, do a regular string search instead. • Like ordinary incremental search except that your input is treated as a sequence of words without regard to how the words are separated. • See the command '**isearch-forward**' for more information.
Search word forward	• `M-s w <RET>` • `<f11> s w f`	(word-search-forward STRING &optional BOUND NOERROR COUNT)	Searches for exact words that may be separated by punctuations and/or lines. Search string must be a complete set of words.
Search word forward lax	`<f11> s w F`	(word-search-forward-lax STRING &optional BOUND NOERROR COUNT	Same as search word forward except that the search string may end in an incomplete word (unless it ends with whitespaces)
Search word backward	• `M-s w C-r <RET>` • `<f11> s w b`	(word-search-backward STRING &optional BOUND NOERROR COUNT	Searches for exact words that may be separated by punctuations and/or lines. Search string must be a complete set of words.
Search word backward lax	`<f11> s w B`	(word-search-backward-lax STRING &optional BOUND NOERROR COUNT)	Same as search word forward except that the search string may end in an incomplete word (unless it ends with whitespaces)
Incremental Search (ISearch)	Start an incremental search with one of the following commands. Type text to search, `<DEL>` to remove chars. Other key-chords can be used during the search. Re-type same key-chord after reaching end of buffer, wrap to other end and continue searching. Or repeat key-chord to repeat last search for same text. To reverse search direction, use the other key-chord (for example: if searching with `C-s`, use `C-r` to go backward) • Type `<RET>` to stop search and leave cursor at found position if next command is to insert a character. Other editing key-chords also stop the search but also perform the requested operation (like `C-a` which ends the search and moves point to the beginning of the line). • Abandon search (and return to where you started, type `<ESC><ESC><ESC>` or `C-g C-g`. On search exit, original point is added to mark ring, thus you can use `C-u C-SPC` or `C-x C-x` to return to the position before the search. ☝📦 `C-s` is normally mapped to isearch-forward. With PEL you can set the **pel-use-swiper** user option which activates the Swiper external package and the `<f11> s s` key. That key allows you to change what command is mapped to `C-s`: search-forward or swiper. You can specify which one is used by default via the **pel-search-with-swiper** user option. Use `<f11> <f1> s` to customize PEL controlled search.		

ISearch - forward
• **Incremental**
  • **literal search**
  • **regexp search**
	• `C-s`
	• `⌘-f`
		(**isearch-forward** &optional REGEXP-P NO-RECURSIVE-EDIT)	Do incremental search forward: start or continue a search.
			• With a prefix argument, do an **incremental regular expression search** instead, something like:
			  • `C-u 1 C-s`
			  • `M-- C-s`
			  • With PEL, `C-- C-s` works.
			  • `C-u C-s` does **not** work to perform a regexp ISearch.
			  ☛ Instead you can also use `C-M-s` to perform the regexp incremental search forward.
			• To continue to next match during search: type **C-s** again (with prefix argument if that was used for regexp Isearch).
			• To change direction: type **C-r**
			• To repeat last completed incremental search forward: **C-s C-s**
			• `⌘-f` is always mapped to isearch-forward.
			📦 On PEL:
			  • This key mapping is used when either **pel-use-swiper** or **pel-search-with-swiper** is nil.
			  • If **pel-use-swiper** is **t**, you can use `<f11> s s` to toggle the map to swiper instead.

Perform Swiper search: interactive search with an overview list	`C-s`	(**swiper** &optional INITIAL-INPUT)	Perform a Swiper text search. Opens up the mini buffer and show several matches as they are being typed. • Narrow the search by typing a pattern. • Multiple patterns are allowed by separating with a space. • Select with C-n, C-p, <up> and <down>. • Chose (and stop the search) with RET. • ☝ To search for a space with Swiper, type 2 spaces in the search expression. So: type "foo␣␣bar" to search for "foo␣bar". 📦 On PEL: • This key mapping is used when **pel-use-swiper** and **pel-search-with-swiper** are both set to **t**. • You can use `<f11> s s` to toggle the map to isearch-forward instead.

ISearch - backward
• **Incremental**
  • **literal search**
  • **regexp search**
	`C-r`	(**isearch-backward** &optional REGEXP-P NO-RECURSIVE-EDIT)	Do incremental search backward: start or continue a search.
			• With a prefix argument, do an **incremental regular expression search** instead; something like:
			  • `C-u 1 C-r`
			  • `M-- C-s`
			  • With PEL, `C-- C-r` works.
			  • `C-u C-r` does **not** work to perform a regexp ISearch.
			  ☛ Instead you can also use `C-M-r` to perform the regexp incremental search forward.
			• To continue to next match during search: type **C-r** again (with prefix argument if that was used for regexp Isearch.
			• To change direction: type **C-s**
			• To repeat last previously completed incremental search backward: **C-r C-r**

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`.

Incremental Symbol Search	Incremental **symbol** search is like incremental search except that the boundaries of the search must match the boundaries of a symbol (for the buffers' major mode). Only complete match will be found. For example searching for *forward-word* in a Lisp file will not match *isearch-forward-word*. Note: ☝ also see the command described above: **pel-search-word-from-top**, bound to `<f11> s .`		
ISearch symbol at point	`M-s .`	(**isearch-forward-symbol-at-point**)	Perform a symbol search starting with current symbol at point. • After capturing the word at point you can extend it by typing **M-w**. • ☝ Useful for searching inside source code while superiors mode is disabled. • Use `C-s` and/or `C-r` to perform extra searches on the same symbol.
ISearch for symbol	`M-s _`	(**isearch-forward-symbol** &optional NOT-SYMBOL NO-RECURSIVE-EDIT)	Prompt for symbol, perform symbol search. • Subsequent searches for the same symbol is done with `C-s` and/or `C-r`. • ☝ Useful for searching code. For example: "data size" matches "data.size" as well as "data->size", "data + size" and "data size".
ISearch for sequence of words	`M-s w`	(**isearch-forward-word** &optional NOT-WORD NO-RECURSIVE-EDIT)	Do incremental search forward for a **sequence of words**. • With a prefix argument, do a regular string search instead. • Like ordinary incremental search except that your input is treated as a sequence of words without regard to how the words are separated.

	Description	Keystroke	Function	Note
During ISearch		The incremental search can be modified to perform other searches. Right after typing the incremental search command you can type the following characters to modify or repeat the search.		
DURING ISEARCH COMMANDS	**Change the search type to:** **simple search**	`<RET>`	• (**search-forward** STRING &optional BOUND NOERROR COUNT) • (**search-backward** STRING &optional BOUND NOERROR COUNT)	Typing <**RET> right after typing** the command (**`C-s`**, **`C-r`**, **`C-M-s`** or **`C-M-r`**) and before typing the text to search for: • **`C-s <RET>`** or **`C-r <RET>`** perform a regular search instead of an iISearch. • **`C-M-s <RET>`** or **`C-M-r <RET>`** perform a regular regex search.
	Add word at point to search string	`C-w`	(**isearch-yank-word-or-char**)	Appends the next character or word at point to the search string. Repeat it to append more to the search string.
	repeat search forward	• `C-s` • `⌘-g`	(**isearch-repeat-forward**)	Repeat the current search, start searching again going forward
	repeat search backward	• `C-r` • `⌘-d`	(**isearch-repeat-backward**)	Repeat the current search, start searching again going backward
	Select searched string	While performing a search you can issue the following commands to modify the searched string text.		
	History previous	`M-p`	(**isearch-ring-retreat**)	Retrieve searched text from search history: get previous entry from history
	History next	`M-n`	(**isearch-ring-advance**)	Retrieve searched text from search history: get next entry from history
	"tab" complete history in buffer	• `C-M-i` • `M-<tab>`	(**isearch-complete**)	Perform "tab" completion for search item in the minibuffer against the search history. Opens a buffer with the complete search history. Any one of the past search string can be selected to perform the new search.
	Edit search string	`M-e`	(**isearch-edit-string**)	Use this while performing a search and wanting to change the string being searched. • When **`M-e`** is typed during the search, the prompt goes back to the minibuffer allowing the editing of the searched string. • Edit then search string in minibuffer. • End editing with **`<RET>`**, **`C-j`**, **`C-s`** or **`C-r`**
	Add rest of line at point to search string	`M-s C-e`	(**isearch-yank-line** &optional ARG)	While searching select the text from cursor to end of line as the search text. If point is already at end of line, appends next line. With numeric argument appends that many next lines.
	Add character at point to search string	`C-M-y`	(**isearch-yank-char** &optional ARG)	Appends character at point to the search string. If numeric argument appends that many characters.
	Yank from kill ring to search string	• `C-y` • `⌘-e`	(**isearch-yank-kill**)	Pull string from kill ring into search string.
	Replace just-yanked search string with previously killed string	`M-y`	(**isearch-yank-pop**)	Replace just-yanked search string (via (search-yank-kill) with previously killed string.
	Modify search method	While performing a search the following commands modify the search method.		
	Start query replace	`M-%`	(**isearch-query-replace** &optional ARG REGEXP-FLAG)	Transforms the Search into a query replace, using the current string as the string to be replaced.
	Start query replace regexp	`C-M-%`	(**isearch-query-replace-regexp** &optional ARG)	Transforms the Search into a regex query replace, using the current string as the regex string to be replaced.
	Enter occur search: list all occurrences	`M-s o`	(**isearch-occur** REGEXP &optional NLINES)	Start an "occur" search with current search string. • See "**`M-s o`**" row above for more information.
	Modify search mode	While performing a search the following commands modify the search modes.		
	Toggle lax whitespace matching	`M-s SPC`	(**isearch-toggle-lax-whitespace**)	Toggle lax matching during this search. Lax matching is on by default. • Any number of whitespace is accepted in the default lax matching. This can also be customized. When off: search exact string.
	Toggle case sensitivity	• `M-c` • `M-s-c`	(**isearch-toggle-case-fold)**	Toggle search case sensitivity.
	Toggle searching in invisible text	`M-s i`	(**isearch-toggle-invible**)	Toggle whether invisible text is searched. • Useful when editing outlined text.
	Toggle regular-expression searching	• `M-r` • `M-s-r`	(**isearch-toggle-regexp)**	Toggle regexp searching on or off.
	Toggles word mode	`M-s w`	(**isearch-toggle-word**)	Toggle word searching on or off. • Turning on word search turns off regexp mode. • For example: in C file : the expression it->second.first is not matched by "is second first" but when the word mode (or the symbol mode) is activated it matches.
	Toggles symbol mode	`M-s _`	(**isearch-toggle-symbol**)	Toggle symbol search mode. • Useful for searching code. For example: "data size" matches "data.size" as well as "data->size", "data + size" and "data size".
	Toggle character folding	`M-s '`	(**isearch-toggle-char-fold**)	Toggle char-fold searching on or off. • Turning on character-folding turns off regexp mode. • When character folding is activated all accentuated letters for a given letter match the letter., otherwise it does not match (ie: 'à' matches 'a' when character folding is activated and does not otherwise).
	Stop the incremental search	`<RET>` : 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.		
Occur Search				

List all matching occurrences of regexp in current buffer		**`M-s o`**	(**occur** REGEXP &optional NLINES)	• Prompts for a regexp
				• Can use **`M-n`** at prompt to recuse previous search strings
				• Use **`M-n`** prefix to specify n lines of context in result. Default=*list-matching-lines-default-context-lines*.
				• "**`M-s o`**" can be used during an incremental search.
				• **In *Occur* buffer**:
				  • <RET> visit corresponding position in the searched buffer
				  • "C-o" display the match in other window (but does not select it)
				  • < , > : go to the beginning and end of the buffer
				  • g : revert the buffer, refreshing the search results
				  • e : buffer enters the **Occur Edit Mode** which allows edits in both buffers simultaneously via edits in the *Occur* buffer.
				  • Exit Occur Edit Mode with:
				  • "**C-c C-c**" (which is: (**occur-cease-edit**))
				  • Navigate though occurrences (in original buffer):
				  • (next-error) : "**`C-x ``**" or "**`M-g n`**" or "**`M-g M-n`**"
				  • (previous-error): "**`M-g p`**" or "**`M-g M-p`**"

Description	Keystroke	Function	Note
Occur search in selected buffers	`<f11> s O`	(**multi-occur-in-matching-buffers** BUFREGEXP REGEXP &optional ALLBUFS)	For example to occur search in all .py files, select the buffers with "**\.py$**" (without the quotes).
Occur search in selected files	`<f11> s o`	(**multi-occur** BUFS REGEXP &optional NLINES)	
During Occur Search			
occur - next occurence	• `C-x ` `   • `M-g n`   • `M-g M-n`	(**next-error** &optional ARG RESET)	A prefix ARG specifies how many error messages to move; • negative means move back to previous error messages. • Just `C-u` as a prefix means reparse the error message buffer and start at the first error.
occur - previous occurence	• `M-g p` • `M-g M-p`	(**previous-error** &optional N)	Prefix arg N says how many error messages to move backwards (or forwards, if negative).
Exit occur mode	`C-c C-c`	(**occur-cease-edit**)	Exit the occur-edit mode. See "`M-s o`" note above.
Replace Commands	Emacs provides the following commands to perform string replacement in buffers. • Some external packages also provides several useful extensions: • visual-regexp • visual-regexp-steroids		
Unconditional Replace	Simple text replacement command.		
Unconditional replace	`<f11> s r`	(**replace-string** FROM-STRING TO-STRING &optional DELIMITED START END BACKWARD)	Replace all instances of from-string by to-string from point to end of buffer. Emacs displays the number of string replaced after the operation
Unconditional regex replace	`<f11> s x r`	(**replace-regexp** REGEXP TO-STRING &optional DELIMITED START END BACKWARD)	Replace every match for regex with new string.
Query Replace	Query replacement prompts. The following 2 commands are query replace. The answers to prompts are listed after the 2 commands.		
Query Replace	`M-%`	(**query-replace** FROM-STRING TO-STRING &optional DELIMITED START END BACKWARD REGION-NONCONTIGUOUS-P)	Replace *some* occurrences of a string with another, both specified by user. A negative argument replaces backwards.
Query Replace Regexp	• `C-M-%` • `<f11> s x q`	(**query-replace-regexp** REGEXP TO-STRING &optional DELIMITED START END BACKWARD REGION-NONCONTIGUOUS-P)	Replace *some* occurrences of a regex match with a specified string. • A negative argument replaces backwards. • C-% is not an ASCII control character, so C-M-% does not work in Terminal mode.
QR Response : keys to use during a query replacement to identify actions	• `y or SPC` : replace • `n or <DEL>` : don't replace, move to next • `.` : replace current and quit • `,` : replace & let me see result before moving on — Press SPC to move on. • `!` : replace all the rest and don't ask • `^` : back up to the previous instance • `u` : undo last replacement • `U` : undo ALL replacements • `q or <RET>` : abort/exit query-replace • `E` : modify the replacement string • `C-r` : enter recursive edit - Exit the recursive edit with one of: C-M-c or C-] • `C-w` : delete this instance and enter recursive edit —to make a custom replacement • `C-M-c` : exit recursive edit and resume query-replace • `C-]` : Exit recursive edit and exit query-replace • `?` : get help • `Y` : replace all strings in all buffer, no questions. — Multi-buffer QR Response • `N` : skip to next buffer without replacing remaining matches in current buffer — Multi buffer QR Response.		
Regular Expression Builder	Use the Regular Expression Builder 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*. Similar to ***string*** but requires double escaping of backslashes - similar to how it must be done in Elisp source code. For example: "\\(red\\\|green\\)" • ***string***: Similar to ***read*** but no double backslashes are needed. Example: "\(red\\|green\)" • ***rx***: A more advanced, s-expression regexp engine, used if you want lisp-style regexp engine.
Select the regular expression syntax used by the re-builder	`<f11> s x <f1>`	(**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.
Change target buffer	`C-c C-b`	(**reb-change-target-buffer** BUF)	Change the target buffer and display it in the target window.
Enter/leave sub-expression highlight mode	`C-c C-e`	(**reb-enter-subexp-mode**)	Enter the subexpression mode in the RE Builder. • Use this to only highlight the capturing groups. • Type **0** to **9** to identify the group to highlight. • Type **q** to exit that mode.
Select regular expression syntax used	• `C-c C-i` • `C-c <tab>`	(**reb-change-syntax** &optional SYNTAX)	Change the syntax used by the RE Builder.
Quit re-builder	`C-c C-q`	(**reb-quit**)	Quit the RE Builder mode.

Description	Keystroke	Function	Note
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.
Regular Expressions Syntax	The following rows describe **Emacs** regular expressions (which differ from other styles of regex) and tools to try them out.		
Regular expression syntax	Boundary anchors: • `^` : beginning of {line, string, buffer} • `$` : end of {line, string, buffer} • `\`` : beginning of {string, buffer} • `\'` : End of {string, buffer} • `\b` : word boundary marker • `\w` : any word character. Alternative: **[[:word:]]** • `\W` : any non-word character. Alternative: **[^[:word:]]** • `.` : any single character except newline • `\.` : one period • `?` : 0 or or 1 of the previous expression • `+?` : match previous pattern 1 or more times, but with minimal match (non-greedy) • `*` : group of 0 or more of the previous expression • `+` : group of 1 or more of the previous expression • `\<` : beginning of word • `\>` : end of word • `_<` : beginning of a symbol • `_>` : end of a symbol • GNU extensions to regular expressions supported by Emacs include \w, \W, \b, \B, \<, \>, \` , \' (start and end of buffer) • `[ ]` : any character in range. Example: [a-z] means all lowercase characters (when case sensitive). Inside range the following characters or expressions can be used: • `^` : complements the set (ie: means that we want to match anything but what is in the set. • `[:C:]` : **character class** ***C***, where ***C*** can be any of: • `alnum` : any letter or digit • `alpha` : any letter • `ascii` : any of the 127 ASCII characters • `blank` : horizontal whitespace • `cntrl` : any ASCII control character • `digit` : any digit character • `graph` : any graphic character; everything except whitespace, ASCII and non-ASCII control characters, surrogates and code points unassigned by Unicode. • `lower` : lower-case letters. If case-fold-search is non-nil it also matches upper-case letters. Use **<f11> s m f** to toggle the value of this variable. • `multibyte` • `nonascii` • `print` • `punct` • `space` • `unibyte` • `upper` • `word` • `xdigit` • `\sC` : Match any character whose syntax table code is ***C.*** • `\SC` : Match any character whose syntax table code is not ***C.*** The syntax table code ***C*** cab be one of: • `SPC or -` : any whitespace : space, newline, tab, carriage return, formfeed, backspace • `w` : word constituents: normally all upper- and lower-case letters, and digits. • `_` : symbol constituents: extra characters used in variable, function, command names. • `.` : punctuation characters. There is none in Lisp. C has some. • `(` : open parenthesis. Support '(', '{', '[' • `)` : close parenthesis. Support '), '}', ']' • `"` : string quotes • `\` : escape-syntax characters • `/` : character quotes • `<` : comment starters • `>` : comment enders • `!` : generic comment delimiters • `|` : generic string delimiters  • `\\|` : Alternative • `\(, \)` : Capturing group • `\{ , \}` : Repetition • `\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) • `\?` : prompt for user input • `\#` : inserts a number incremented from 0 • `\&` : insert whole match string • `\,(form …)` : uses an elisp form with arguments. Use elisp form that take and return strings, such as the following examples: • `\,(upcase \2)` : uppercase capturing group 2 • `\,(format "%.2f" \#3)` : Cast group 3 as number and format it as decimal with 2 decimal points.  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:]]**		

Variables controlling search aspects

Variable	Description	Note
case-fold-search	t : ignore case unless the user types in mixed or uppercase. nil: case sensitive: exact match.	Applies to all searches. To change: use **pel-toggle-case-fold-search**
case-replace	t: preserve case in replacements. nil: don't just case, replace with exact string identified.	Applies to all searches
NOTE =>		To set the variables, use: M-x set-variable
NOTE =>		To set defaults inside init.el, use: (setq-default VARIABLE VALUE)

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.
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 @ 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.