





















Grep

| Description | Keystroke | Function | Note |
|--|--|---|--|
| <p>Grep under Emacs</p> <p>See also:</p> <ul style="list-style-type: none"> 🔗 Dire 🔗 Projectile 🔗 Search/Replace | <p>Emacs support several find and grep commands that can be executed from within Emacs. Doing so has several advantages:</p> <ul style="list-style-type: none"> The command output is collected in the "grep" Emacs buffer while the command runs asynchronously. The buffer is read-only, you can search within it right away. If you type RET on a found line, Emacs visit the file of the match at the appropriate line. Without even going inside the search result buffer you can type one of the 'goto match' commands to move to the next or previous match: <ul style="list-style-type: none"> (next-error) : C-x ` ,M-g n or M-g M-n (previous_error): M-g p or M-g M-p Stop the grep asynchronous operation with C-c C-k or <f11> g k The search commands keep a history for the searches. You can browse the history at the prompt. <p>📦 PEL supports installation and use of several grep-enhancing external packages. 📖 All PEL user-options for grep are in the pel-pkg-for-grep group.</p> <p>🔗 Customize grep support to get new features and enhance its speed using the following commands.</p> | | |
| <p>Open this PDF file.</p> <p>See also: 🔗 Help/Info</p> | <f11> g <f1> | (pel-help-pdf &optional OPEN-WEB-PAGE) | Open the 🔗 Grep 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. |
| <p>Customize PEL Grep Support</p> <p>See also: 🔗 Customize</p> | <f11> g <f2> | (pel-customize-pel &optional OTHER-WINDOW) | Customize PEL grep support. Use it to activate/de-activate various grep tools. <ul style="list-style-type: none"> If OTHER-WINDOW is non-nil (use C-u), display in other window. |
| <p>Customize Emacs Grep Support</p> <p>See also: 🔗 Customize</p> | <f11> g <f3> | (pel-customize-library &optional OTHER-WINDOW) | Customize Emacs grep support. Groups: grep, ag, deadgrep, fzf, rg, ripgrep, wgrep. |
| <p>Grep Search in Emacs</p> | <p>Use one of the following commands to perform grep operation on files. All of these commands share the following:</p> <ul style="list-style-type: none"> They collect output in the "grep" buffer. Grep runs asynchronously, you can use any of the grep finding navigation commands while it is still running. See the list below. To kill the grep job before it finishes, type C-c C-k. This commands use a special history list for its arguments, so you can easily repeat a grep and find command. | | |
| <p>Run grep via find</p> <p>See also: 🔗 File mngr</p> | <ul style="list-style-type: none"> <f11> f g <f11> g f | (grep-find COMMAND-ARGS) | <p>Grep in files found by the find utility: run grep via find with user-specified args COMMAND-ARGS.</p> <ul style="list-style-type: none"> Collect output in the "grep" buffer. While find runs asynchronously, you can use the C-x ` to find the text that grep hits refer to. This command uses a special history list for its arguments, so you can easily repeat a find command. <p>👉 Faster alternatives are available if you use ripgrep or ag. See below.</p> |
| <p>Run grep</p> | <f11> g g | (grep COMMAND-ARGS) | <p>Run Grep with user-specified COMMAND-ARGS, collect output in a buffer.</p> <ul style="list-style-type: none"> Without prefix, prompts to complete a grep command. <ul style="list-style-type: none"> The default grep command is: grep -nH --null -e For example to search for main in all .c files complete it like this: grep -nH --null -e main *.c If you prefix the command with C-u, Emacs creates a command to grep for the string at point in the same types of files as the current file. |
| <p>Local grep</p> | <f11> g l | (lgrep REGEXP &optional FILES DIR CONFIRM) | <p>Run grep, searching for REGEXP in FILES in directory DIR.</p> <ul style="list-style-type: none"> The search is limited to file names matching shell pattern FILES. FILES may use abbreviations defined in 'grep-files-aliases', e.g. entering 'ch' is equivalent to '*. [ch]'. As whitespace triggers completion when entering a pattern, including it requires quoting, e.g. 'C-q<space>'. With C-u prefix, you can edit the constructed shell command line before it is executed. With C-u C-u prefix, directly edit and run 'grep-command'. This command shares argument histories with <f11> g r and <f11> g g. |
| <p>Recursive grep over files in directory tree using grep</p> | <f11> g r | (rgrep REGEXP &optional FILES DIR CONFIRM) | <p>Recursively grep for REGEXP in FILES in directory tree rooted at DIR.</p> <ul style="list-style-type: none"> The search is limited to file names matching shell pattern FILES. |
| | <ul style="list-style-type: none"> FILES may use abbreviations defined in 'grep-files-aliases', e.g. entering 'ch' is equivalent to '*. [ch]'. As whitespace triggers completion when entering a pattern, including it requires quoting, e.g. 'C-q<space>'. With C-u prefix, you can edit the constructed shell command line before it is executed. With C-u C-u prefix, directly edit and run 'grep-find-command'. This command shares argument histories with M-x lgrep and M-x grep-find. When called programmatically and FILES is nil, REGEXP is expected to specify a command to run. <p>As seen above, typing the C-u prefix before the command keystroke, we can modify the command line found, e.g. for adding a sort -V so that output list the files in sorted order with lines number also sorted.</p> | | |
| <p>Recursive Gzip grep</p> | <f11> g R | (zrgrep REGEXP &optional FILES DIR CONFIRM TEMPLATE) | <p>Recursively grep for REGEXP in gzipped FILES in tree rooted at DIR.</p> <ul style="list-style-type: none"> Like 'rgrep' but uses 'zgrep' for 'grep-program', sets the default file name to '*.gz', and sets 'grep-highlight-matches' to 'always'. It can search into compressed files as well as regular files. |
| | | | |
| <p>Kill grep process</p> | <ul style="list-style-type: none"> C-c C-k <f11> g k | (kill-grep) | <p>Kill the grep process that currently runs asynchronously.</p> <ul style="list-style-type: none"> Useful if you want to interrupt a long-running grep command. |
| <p>Grep findings navigation commands</p> | <p>Once a grep operation has been launched, the following commands can be used to navigate through the result, moving to the location of a match, inside the file at the matched line number. Fo async grep operations the commands can be used while the grep is still processing. The commands can be issued while point is inside the "grep" buffer but also outside.</p> | | |
| <p>Move to first match</p> | <f11> g l | (first-error &optional N) | <p>Restart at the first grep found.</p> <ul style="list-style-type: none"> Visit corresponding source code. With prefix arg N, visit the source code of the Nth error. |
| <p>Move to next match</p> | <ul style="list-style-type: none"> C-` 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. |
| <p>Move to previous match</p> | <ul style="list-style-type: none"> 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). |
| <p>wgrep: edit grep buffer & write changes to files</p> | <p>The grep, lgrep, rgrep and zgrep search result are show in a "grep" buffer that is read-only and can be used to jump to the location of the found text.</p> <p>📦 With the wgrep external package, you can activate a mode that allows you to edit the buffer and save the modification back to the files.</p> <p>📖 PEL activates wgrep when the pel-use-wgrep user option is non-nil. It also activates it when pel-use-ripgrep and pel-use-ivy are non-nil.</p> <ul style="list-style-type: none"> PEL activates support for wgrep in the grep-mode, rg-mode and ag-mode buffers. You can perform a grep search that places result in grep-mode buffer with the following commands: <ul style="list-style-type: none"> grep (<f11> g g), grep-find (<f11> g f), lgrep (<f11> g l), rgrep (<f11> g r), zgrep (<f11> g z). The RipGreg based commands: rg (<f11> g i), rg-menu (<f11> g m) The ag based commands: ag-regex (<f11> g a x), ag-files (<f11> g a f), ag-project-files (<f11> g a p f) and similar, ag-dired (<f11> g a d d) and similar. | | |
| <p>Activate wgrep in grep-mode buffer</p> | <ul style="list-style-type: none"> C-c C-p C-x C-q | (wgrep-change-to-wgrep-mode) | <p>Change to wgrep mode, allowing modifying the text to save back into file with a specific command (see below). PEL activates the C-x C-q key binding is available for grep-mode, ag-mode and rg-mode buffers.</p> <p>⚠️ When the "grep" buffer is huge, this might freeze your Emacs for several minutes.</p> |
| <p>Save all changes</p> | C-c C-s | (wgrep-save-all-buffers) | Save the buffers that wgrep changed. |
| <p>Save changes to files and exit wgrep mode</p> | <ul style="list-style-type: none"> C-C C-e C-C C-c C-x C-s | (wgrep-finish-edit) | <p>Apply changes to file buffers.</p> <p>These changes are not immediately saved to disk unless 'wgrep-auto-save-buffer' is non-nil.</p> |
| <p>Delete line</p> | C-C C-d | (wgrep-mark-deletion) | Mark current line as deleted. This change will be applied when C-x C-s is typed. |

| Description | Keystroke | Function | Note |
|--|--|--|---|
| Discard changes & quit | C-C C-k | (wgrep-abort-changes) | Discard all changes and return to original mode. |
| Quit | C-x C-q | (wgrep-exit) | Return to original mode. |
| Toggle read-only area | C-C C-p | (wgrep-toggle-readonly-area) | Toggle read-only area to remove a whole line. <ul style="list-style-type: none"> See the following example: you obviously don't want to edit the first line. If grep matches a lot of lines, it's hard to edit the grep buffer. After toggling to editable, you can call 'delete-matching-lines', 'delete-non-matching-lines'. <div> Example: <pre>----- ./svn/text-base/some.el.svn-base:87:(hoge) ./some.el:87:(hoge) -----</pre> </div> |
| Remove changes in region | C-C C-r | (wgrep-remove-change BEG END) | Remove changes in the region between BEG and END. |
| Remove all changes | C-C C-u C-c Esc | (wgrep-remove-all-change) | Remove changes in the whole buffer. |
| Search in project with projectile See 🔗 Projectile | If projectile is available, you can also search in project-related files with Projectile. With PEL, activate the projectile mode with <f11> <f8> <f8> Note that both ripgrep and ag also provide project-based searching but projectile provides more control over the concept of project.  These commands require projectile external package  PEL activates projectile when the pel-use-projectile user option is non-nil. More on ripgrep and ag below. | | |
| Search in project files with recursive grep | <f8> s g | (projectile-grep &optional REGEXP ARG) | Perform rgrep in the project. <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> Rust (ripgrep) regex syntax | <f8> s r | (projectile-ripgrep SEARCH-TERM &optional ARG) | Run a Ripgrep search with 'SEARCH-TERM' at current project root. <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> PCRE regex syntax | <f8> s s | (projectile-ag SEARCH-TERM &optional ARG) | Run an ag search with SEARCH-TERM in the project. <ul style="list-style-type: none"> 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. |
| Speeding up Emacs grep searching | By default Emacs uses the standard find, grep, rgrep and zgrep command line utilities to perform the grep-based search operations. <ul style="list-style-type: none"> The actual utility used depends on what is available on your Operating System.  As fast as grep might be, tools like ripgrep and ag are much faster. See the section below on ripgrep and ag. <ul style="list-style-type: none"> Also note that grep-find-command user option can be modified to use ripgrep for the normal grep-find command. See notes in the reference table below. | | |
| ripgrep front end tools <ul style="list-style-type: none"> Rust (ripgrep) regex syntax | PEL supports rg.el and deadgrep . They are both very useful. Each have their strong points. <ul style="list-style-type: none"> rg.el gives more control over the ripgrep command line options: the rg-menu command allows you to select —multiline search, for example. deadgrep outputs in buffers that can be refreshed as the content of files change. The search criteria can also be modified in the buffer. <ul style="list-style-type: none"> Ability to refresh the buffer is very useful when changing names of software elements: use 2 deadgrep searches: one for the old name and one for the new name and see the first buffer deplete and the second increase as you modify your code. | | |
| Searching with RipGrep explicitly using rg.el <ul style="list-style-type: none"> Rust (ripgrep) regex syntax | The following commands use ripgrep explicitly to perform various types of searches. These search are much faster than grep on large file set . <ul style="list-style-type: none"> Using ripgrep through the rg.el package inside Emacs makes it very flexible with lots of features available on the command line that can be easily activated via rg-menu <ul style="list-style-type: none"> To get the list of all rg command lines you can use man (or woman) inside Emacs. With PEL you can type: <f11> ? m rg RET <ul style="list-style-type: none"> See 🔗 Help/Info for info on man and woman. The following commands use ripgrep explicitly to perform fast grep-type searches. Using ripgrep is highly recommended.  Remember that ripgrep uses the Rust regexp syntax , not Emacs regexp syntax.  The commands require the rg.el package.  PEL activates the use of ripgrep with rg.el inside Emacs when pel-use-ripgrep user option is set to t. You must install ripgrep command line utility separately. When pel-use-projectile is set to t, PEL also installs ripgrep.el that projectile uses. | | |
| Recursive regexp grep over files in directory tree using RipGrep <ul style="list-style-type: none"> Rust (ripgrep) regex syntax | <ul style="list-style-type: none"> <f11> g i C-c s r | (rg QUERY FILES DIR) | Run ripgrep, searching for REGEXP in FILES in directory DIR. <ul style="list-style-type: none"> The search is limited to file names matching shell pattern FILES. |
| | <ul style="list-style-type: none"> FILES may use abbreviations defined in 'rg-custom-type-aliases' or ripgrep builtin type aliases, e.g. entering 'elisp' is equivalent to '*.el'. REGEXP is a regexp as defined by the ripgrep executable. With C-u prefix (CONFIRM), you can edit the constructed shell command line before it is executed. <ul style="list-style-type: none"> Useful to add the -z option to search into compressed files (.zip, .el.gz, etc...). Collect output in a buffer. While ripgrep runs asynchronously, you can use C-x ` (M-x 'next-error'), or RET in the rg output buffer, to go to the lines where rg found matches.  ⚠ With PEL, only <f11> g i forces loading of the rg package. At first the C-s s r key binding is not set. It will be as soon as rg is loaded (forced by the other bindings). | | |
| Recursive literal grep over files in directory tree using RipGrep <ul style="list-style-type: none"> Rust (ripgrep) regex syntax | <ul style="list-style-type: none"> <f11> g t C-c s t | (rg-literal QUERY FILES DIR) | Run ripgrep, searching for literal PATTERN in FILES in directory DIR. <ul style="list-style-type: none"> With C-u prefix (CONFIRM), you can edit the constructed shell command line before it is executed. <ul style="list-style-type: none"> Useful to add the -z option to search into compressed files (.zip, .el.gz, etc...).  ⚠ With PEL, only <f11> g I forces loading of the rg package. At first the C-s s t key binding is not set. It will be as soon as rg is loaded (forced by the other bindings). |
| Open RipGrep transient menu | <ul style="list-style-type: none"> <f11> g m C-c s | (rg-menu) | Shows the transient menu for rg. <ul style="list-style-type: none"> While the menu is active all keys and mouse actions are controlled by the rg menu, even though point remains where it was. <ul style="list-style-type: none"> The menu list options and commands shown below. <ul style="list-style-type: none"> Type the options first and include the dash in what you type. Then select the command. The current directory affects what is proposed in the directory and file type prompts of several commands. The results are show inside a *rg* buffer. There are several commands available inside the *rg* buffer which provide other functionality. Type ? in the *rg* buffer for help. ⚠ The default binding C-c s is also used for several cross-reference tools such as CScope which hijacks the binding when enabled if the cscope-keymap-prefix user option is not changed. <ul style="list-style-type: none"> With PEL, the PEL binding is always available in any case. <div> <div> Search d Dwim r Regex t Literal p Project c Current directory f Current file </div> <div> Manage l List s Save S Save as name </div> <div> Switches -h Search hidden files (--hidden) -z Search zipped files (--search-zip) -v Invert match (--invert-match) -U Multi line (--multiline-dotall --multiline) -w Search words (--word-regexp) -I Don't cross file system (--one-file-system) -n Override ignore files (--no-ignore) </div> <div> Options -C Show context (--context=) -M Omit long lines (--max-columns=) -m Max matches per file (--max-count=) -g Filter files glob (--glob=) -T Exclude files types (--type-not=) </div> </div> |

| Description | Keystroke | Function | Note |
|---|---|---------------------------------------|---|
| Grep with Fuzzy File Finders See 🔗 File-mngt | 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. To use it inside Emacs, you must: <ul style="list-style-type: none"> install and configure the fzf command line utility, and  Use the fzf.el external package  activated by pel-use-fzf. The fzf commands below are available when this is active. <ul style="list-style-type: none"> By default all fzf grep operations use grep. It can also use the much faster ripgrep command line tool. Customize fzf/grep-command with <code><f11> g <f3> 4</code> and select ripgrep for fzf/grep-command to take advantage of that. | | |
| Grep search files with fzf for specified regex | <code><f11> g z</code> | (fzf-grep) | Prompt for string to search and file grep selection expression, show grep results in a fzf session, select appropriate line to open the specific file at appropriate line. |
| Grep search files with fzf for specified regex in specified directory | <code><f11> g Z</code> | (fzf-grep-in-dir) | Prompt for directory, then for string to search and file grep selection expression, show grep results in a fzf session, select appropriate line to open the specific file at appropriate line. |
| Grep search Git repo member files with fzf for specified regex | <code><f11> g G</code> | (fzf-git-grep) | Prompt for string to search inside files committed in Git repository. Show grep results over current Git repo searched in a fzf session, use fzf extra filtering text to select select appropriate line to open the specific file at appropriate line.  This does *not* search into previous revisions of the Git committed files, it only searches the files currently in the file system that are committed. |
| Grep search HG repo member files and revisions with fzf for specified regex | <code><f11> g H</code> | (fzf-hg-grep &optional ALL-REVS) | Grep specified versions of files committed in Mercurial repo, fzf result. Prompts for string to search, using hg grep regular expressions (Python/Perl regexp). <ul style="list-style-type: none"> Without prefix argument: grep files committed in Mercurial repo and perform a fzf search on the output.  With C-u prefix it greps inside all revisions of the files. Perform fzf search on the grep result and open selection at specific line. |
| Searching with ripgrep using deadgrep |  The deadgrep external package uses the ripgrep command line tool to provide a buffer-central control of ripgrep searches.  PEL installs and activates it when the pel-use-deadgrep user-option is turned on (set to t). Use <code><f11> g <f2></code> to open the customization buffer for it. When activated the following commands are available. | | |
| Start a deadgrep search • Rust (ripgrep) regex syntax | <code><f11> g d</code> | (deadgrep SEARCH-TERM) | Start a ripgrep search for SEARCH-TERM. Prompt for the search string. <ul style="list-style-type: none"> If called with a prefix argument, create the results buffer but don't actually start the search. Opens a "deadgrep" search result and control buffer where you can see the findings, navigate through them and perform other ripgrep searches with the ability to modify various search parameters. See the deadgrep-mode commands available in this buffer in the following rows below. |
| Visit result at point | <code>RET</code> | (deadgrep-visit-result) | Goto the search result at point. |
| Visit result at point, in other window | <code>o</code> | (deadgrep-visit-result-other-window) | Goto the search result at point, opening in another window. |
| Move point to Next result/button | <code>n</code> | (deadgrep-forward) | Move forward to the next item. <ul style="list-style-type: none"> This will either be a button, a filename, or a search result. See also 'deadgrep-forward-match'. |
| Move point to Previous result/button | <code>p</code> | (deadgrep-backward) | Move backward to the previous item. <ul style="list-style-type: none"> This will either be a button, a filename, or a search result. See also 'deadgrep-backward-match'. |
| Move point to Next search hit | <code>N</code> | (deadgrep-forward-match) | Move point forward to the beginning of next match. <ul style="list-style-type: none"> Note that a result line may contain more than one match, or zero matches (if the result line has been truncated). |
| Move point to Previous search hit | <code>P</code> | (deadgrep-backward-match) | Move point backward to the beginning of previous match. |
| Move point to next file header | <code>M-n</code> | (deadgrep-forward-filename) | Move forward to the next filename. |
| Move point to previous file header | <code>M-p</code> | (deadgrep-backward-filename) | Move backward to the previous filename. |
| Re-run the search | <code>g</code> | (deadgrep-restart) | Re-run ripgrep with the current search settings. |
| Collapse/Expand result for a file | <code><tab></code> | (deadgrep-toggle-file-results) | Show/hide the results of the file at point. <ul style="list-style-type: none"> Point must be located on a file header line. |
| Stop a running search | <code>C-c C-k</code> | (deadgrep-kill-process) | Kill the deadgrep process associated with the current buffer. |
| Searching with Ag explicitly Ag uses PCRE See also: <ul style="list-style-type: none"> PCRE regex syntax PCRE cheatsheet Learn PCRE in Y | The following commands use ag explicitly to perform various types of searches. <ul style="list-style-type: none"> Some of the commands restrict the search to files of specified types. They search in field with extensions corresponding to the specified type. <ul style="list-style-type: none"> For a list of the file types , execute the command ag --list-file-types. It is also possible to provide more command lines to the ag command executed by a command. <ul style="list-style-type: none"> For that, type a command prefix (like C-u) before typing the command key sequence. For instance, you can use the -z command line switch to <i>search into zip files</i>. To get the list of all ag command lines you can use man (or woman) inside Emacs. With PEL you can type: <code><f11> ? m ag RET</code> See 🔗 Help/Info for info on man and woman.  These command require the ag.el package and the ag command line utility.  PEL activates the use of ag inside Emacs via ag.el when pel-use-ag user option is set to t. You must install ag command line utility separately. | | |
| Literal search in files of specified directory tree | <code><f11> g a a</code> | (ag STRING DIRECTORY) | Search using ag in a given DIRECTORY for a given literal search STRING, with STRING defaulting to the symbol under point. <ul style="list-style-type: none"> If called with a prefix, prompts for flags to pass to ag. |
| Regexp search in files of specified directory tree | <code><f11> g a x</code> | (ag-regexp STRING DIRECTORY) | Search using ag in a given directory for a given regexp. <ul style="list-style-type: none"> The regexp should be in PCRE syntax, not Emacs regexp syntax. If called with a prefix, prompts for flags to pass to ag. |
| Literal search in files of specific types in specified directory tree | <code><f11> g a f</code> | (ag-files STRING FILE-TYPE DIRECTORY) | Search using ag in a given DIRECTORY for a given literal search STRING, limited to files that match FILE-TYPE. <ul style="list-style-type: none"> Prompt for file type. Support tab completion. Type tab to get the list of file types. STRING defaults to the symbol under point. If called with a prefix, prompts for flags to pass to ag. |
| Ag search in project files | The following commands perform an ag search in the field of the current project directory tree. <ul style="list-style-type: none"> The project directory tree is identified by the presence of a (D)VCS depot file (.git, .hg, etc..) in a directory. | | |
| Literal search in project files of specific type | <code><f11> g a p f</code> | (ag-project-files STRING FILE-TYPE) | Search using ag for a given literal search STRING, limited to files that match FILE-TYPE. <ul style="list-style-type: none"> Prompt for file type. Support tab completion. Type tab to get the list of file types. STRING defaults to the symbol under point. If called with a prefix, prompts for flags to pass to ag. |
| Literal search in all project files | <code><f11> g a p p</code> | (ag-project STRING) | Guess the root of the current project and search it with ag for the given literal search STRING. <ul style="list-style-type: none"> If called with a prefix, prompts for flags to pass to ag. |
| Regexp Grep in all project files | <code><f11> g a p x</code> | (ag-project-regexp REGEXP) | Guess the root of the current project and search it with ag for the given regexp. <ul style="list-style-type: none"> The regexp should be in PCRE syntax, not Emacs regexp syntax. If called with a prefix, prompts for flags to pass to ag. |
| Find files with matching name in directory tree | <code><f11> g a d d</code> | (ag-dired DIR STRING) | Recursively find files in DIR matching literal search STRING. <ul style="list-style-type: none"> The PATTERN is matched against the full path to the file, not only against the file name. The results are presented as a 'dired-mode' buffer with 'default-directory' being DIR. |

| Description | Keystroke | Function | Note |
|---|---------------|----------------------------------|---|
| Find files with name matching regex in directory tree | <f11> g a d x | (ag-dired-regexp DIR REGEXP) | Recursively find files in DIR matching REGEXP. <ul style="list-style-type: none"> The regexp should be in PCRE syntax, not Emacs regexp syntax. The REGEXP is matched against the full path to the file, not only against the file name. Results are presented as a ‘dired-mode’ buffer with ‘default-directory’ being DIR. |
| Find project files with a name matching regex | <f11> g a d f | (ag-project-dired-regexp REGEXP) | Recursively find files in current project matching REGEXP. |
| | | | |
| Kill all ag buffers | <f11> g a k a | (ag-kill-buffers) | Kill all ‘ag-mode’ buffers. |
| Kill other ag buffers | <f11> g a k o | (ag-kill-other-buffers) | Kill all ‘ag-mode’ buffers other than the current buffer. |
| Kill ag process | <f11> g a k p | (ag/kill-process) | Kill the ‘ag’ process running in the current buffer. |

Grep under Emacs - Reference

| Grep | <p>The standard grep command is available on all POSIX systems.</p> <ul style="list-style-type: none"> Emacs default uses grep for the grep, zgrep and zgrep Emacs commands. <ul style="list-style-type: none"> For rgrep and zrgrep, Emacs uses a command line that invokes grep (or zgrep) via find. The actual used command line is specified by the grep-find-template user option. |
|--|--|
| GNU EMacs Manual — Searching with Grep under Emacs | Describes Emacs basic grep support - with the use of standard GNU grep |
| GNU Grep 3.5 manual | GNU grep is available by default on Linux distributions. The actual version depends on the distribution. |
| freeBSD Grep | freeBSD grep is available on freeBSD and macOS |
| OpenBSD Grep | |
| ripgrep : rg <div>Rust (ripgrep) regex syntax</div> | You can use ripgrep instead of GNU BSD grep to significantly speed grep searches. It's possible to use ripgrep by itself, with its own command. It is also possible to replace grep in the standard Emacs commands using grep. |
| Ripgreg utility: rg @ Github | Github home for the ripgrep command line utility: rg |
| Emacs rg.el package @ Github | Github home for rg.el |
| rg.el manual | Latest version of the rg.el manual (previous versions available on the same site) |
| ripgrep.el @ Github | Github home of ripgrep.el |
| Blazing-fast jump-to-grep in Emacs using ripgrep | <p>Describes how the grep-find-command user option can be modified to use ripgrep for the normal grep-find command. Note that you do not need to write Emacs code as described in that article, all you have to do is customize gre-find-command user option and change the value.</p> <p>By default, grep-find-command is nil, but is set by Emacs to:</p> <pre>'("find . -type f -exec grep -nH --null -e \\{\\} +" . 42)</pre> <p>Also note that changing grep-find-command will not impact the behaviour of the zgrep command, so it will still be slow. A better solution this problem is to leave grep-find-command alone and use the rg.el package which provides fast ripgrep searches, including gripping into zip files.</p> |
| • Deadgrep <div>Rust (ripgrep) regex syntax</div> | deadgrep is another, very useful, ripgrep front end for Emacs. Search result show in a refreshable buffer with option control. |
| deadgrep @ GitHub | Home page |
| Ag <div>PCRE regex syntax</div> | The ag.el package is a front end for the ag, “ <i>The Silver Searcher</i> ” command line utility, which uses PCRE. |
| Ag @ GitHub | Github home of the ag command line utility. |
| ag.el @ Github | Github home for the Emacs package that provides access to the ag command line utility. |
| Online regexp testers | <p>There are several online regexp tools. Some are shown below.</p> <ul style="list-style-type: none"> Debuggerx : Javascript, Python, PCRE. With cheatsheets. With visualization of regexp graph. Match rendering. regex101 : PCRE, Javascript, Python, Golang, Java 8. With cheatsheet. Textual explanation of regexp. Match rendering. regexr : Javascript, PCRE. Text graph rendering. RexV.2 : PHP PCRE, PHP Posix, Perl, Python, Javascript, Node.JS. Cheatsheet. Flags control. |