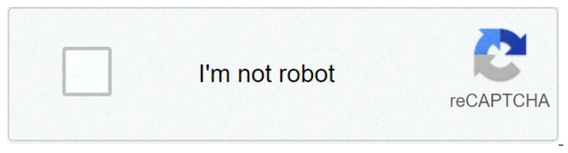

Continue

Multiple grep command in linux

The grep command is one of the oldest tools for Linux and other platforms. Actually, it is older than Linux itself. It was written by Ken Thompson more than 45 years ago! The name grep stands for “globally regular expression print”. This name comes from its predecessor ed and the specific mode in which you would globally search, using a regular expression, and print the output. The related command was “g/re/p”. For more history, have a look at the Wikipedia entry. Otherwise, let’s dive into the tool and get to know some practical grep examples for daily usage.Grep by exampleIntroductionOne of the reasons to create this blog post is that there are a lot of examples available for the grep command. But with all information scattered, most people don’t take the time to really learn the most basic commands. We want to leverage the full potential of the grep command, as it can be used in many work-related or personal related activities. It is common to use it for checking configuration files and searching through log files.Why learn the grep command and regular expressions?As with every tool, it is often easy to start using it, but hard to really master it. The man page is very extensive, so is the online help documentation. Although these sources are a great reference, we will be showing the grep command by example. And we will include specific use-cases which are common for system administrators and security professionals. Especially if you have to deal often with data, investing some time in doing things efficiently will pay off.Before you continueIf you are using grep on another platform than Linux, you may not have the GNU version of grep. Some things in this guide may not be working, or need specific tailoring. You can easily find out what version you have with grep --version.Need a particular job to be done with the grep command and can’t get it to work? Use the comments and share what you have tried. Let’s start with the basics and become a ‘grep master’.Basic usage examples of grepUse grep for simple actionsThe grep utility does not need much to starts doing its work. The syntax of grep consists of four parts.grep commandoptional: option(s)string to searchfile, files, or path to be searchedThe options that grep uses typically have a long and short format. The long format has two dashes, followed by a word or words. Use the long format when using them in scripts, so that it becomes obvious what the grep command is doing. Use the short notation in your daily tasks and on the command line, to save on typing and speed up your work.If you would like to find the root user in your /etc/passwd file, just tell it to search for ‘root’ and the file name itself. In this case, no option is needed.grep root /etc/passwdThe output may look something like this:Using colored grep outputIf the command above did not show colored output on your system, you might want to enable that. It can be done with --color auto. As this would mean you have to type it in each time, using an alias would save you from a lot of typing.alias grep='grep --color=auto'You can add this alias to your .bash_aliases or .bashrc file if you are using the bash shell. Otherwise, add it to the respective profile file. These files can be found in your home directory.Ignore case sensitivityNow that we have performed a basic grep command, we can start to change its behavior. Often we already know the word or words we are looking for. What we don’t always know is if one or more occurrences of the word are using capitals. By default, the grep command will be case-sensitive. So only the right match will be displayed. We can tell grep to ignore case-sensitive searches with the -ioption.grep -i root /etc/passwdShow line numbersDepending on your search, you may have many occurrences of the text you were searching for. Use the -n option to have grep show the related line numbers.grep -n root /etc/passwdExcluding wordsTo exclude particular words or lines, use the –invert-match option. Use grep -v as a shorter alternative. Exclude multiple words with grep by adding -E and use a pipe (|) to define the specific words. Optionally make it case insensitive with the -i as listed above.grep -i -v -E 'banana|monkey' zoo.txtMatch countingIt may be useful to know the number of occurrences of your specified word. This count is displayed when using grep -c or grep -c -v to show the number of non-matching lines.grep -c monkey zoo.txtRecursive search through directories and filesTo search in one directory, there are the -r and -R options to achieve this. Depending on the target and the existence of symlinks, you might want to use the first one if you do not want to follow them. Use the capitalized option, grep -R, if you want to include any possible symlinked file to be searched as well. This may take much longer and could result in other file systems to be searched as well.grep -r password /etcTip: if you don’t want the filenames in the output, add the -h option.Show matching files onlySometimes you just want to see the files that match a particular text string. There is the grep -l command to do achieve this.grep -l -R password /etcTo show all files that do not match your target, use the capitalized version: grep -L.Using regular expressionsThe grep utility is a powerful tool and can use regular expressions. Regular expressions can be considered ‘logic rules’ for matching text strings. Think of something like “I know the word should be starting with the letter ‘a’, but after that everything is fine”. By using a regular expression we can express this in short notation (e.g. "a.*").Match specific words onlyYou may be searching for a very short, yet specific word. In this case, grep will return way too many results. By using more specific statements we can limit the output.grep "\bbin\b" /etc/passwdThe \btells grep to use word boundaries.Although you could use spaces to search for a full word, that often won’t give you the right result. It will return some hits, while it might be missing a few as well. For example, any occurrences at the begin or end of the file. There will also be no match if any special characters are followed by it, or even a simple character like a comma.Tip: the -woption does the same as this regular expression and is easier to remember.Find lines starting with a specific stringWith the carrot symbol (^) we can activate a regular expression that defines that the line should start with a specific piece of text.grep "^systemd" /etc/passwdFind lines ending with a specific stringLike the carrot symbol, we can use the dollar sign ($) to mark the end. Only lines that match that, will be returned. A great way to find all accounts that have a particular shell configured.grep "bin/bash$" /etc/passwdSearch for multiple wordsSometimes you want to match multiple words. By using parentheses you can tell grep to search for one word, or the other. Each possible match is split by a pipe sign.grep -E "^(backup|root|syslog)" /etc/passwdMatching multiple wordsNote: use the -E option to enable extended regular expressions. Without it, the command won’t give any results.Combining grep with other toolsExit codeUsing grep in your shell scripts can be very useful. For example, you can use it to determine if a particular file has the right configuration setting and then perform an action based on that. Another one is to see if a particular user exists in your /etc/passwd file.grep -q michael /etc/passwdGrep will not display anything, but end with an exit code. This exit code will be stored in a special variable with the name $?. If you want to see it on the command line, use it with echo.echo $?Exit codes:0 = match found1 = no match found2 = errorExample syntax to use grep in your shell script:if $(grep -q michael /etc/passwd); then echo "Michael is in passwd file"; else echo "Michael is not in passwd file"; fiUsing pipesThe grep command is a great utility to use in combination and filter the output of other commands. This way the screen only shows that data you are interested in. To achieve this we use the pipe sign (|) to tell the shell to send any output to the next command in line.It is common to apply multiple grep commands by piping them together. When using big data files, try to limit the number of pipes to increase performance. You may also want to look for alternative solutions when you are repeating them often.Example: Search in dmesg outputThe dmesg command gives a lot of lines as output. If we are just interested in information regarding our storage, we can easily do by searching for “sd”.dmesg | grep sdIf we just would like to find AppArmor related events, it would make sense to ignore case due to the capitals in the name. By smart combining the right tools, we can form a powerful data filter.dmesg | grep -i apparmorAdvanced tipsImprove search speed: fixed stringsTypically you may be using already a specific word that you want to be matched. When searching through big files, grep may take a while to complete its task. By using the -F (fixed strings) option this can be dramatically improved. The only downside is that regular expressions can not be used.Searching inside compressed data (avoid using gunzip!)Need to search inside compressed files? Use the zgrep utility. It has the same syntax and it knows how to deal with compressed data.ConclusionThe grep command is a very powerful tool and easy to work with. To truly master it, one should be learning more about regular expressions. It makes searching and finding the right data much easier. Knowledge about regular expressions will also come in handy for other tools, like sed and awk. If you really want to learn how to use the grep command, use it daily and create your own list of commands you often use.Do you have a great one-liner that you often use with grep? Don’t keep it secret and share it in the comments!
grep, egrep, fgrep - print lines matching a pattern
Synopsis
grep [OPTIONS] PATTERN [FILE...]
grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]
Description
grep searches the named input FILEs (or standard input if no files are named, or if a single hyphen-minus (-) is given as file name) for lines containing a match to the given PATTERN. By default, grep prints the matching lines. In addition, two variant programs egrep and fgrep are available. egrep is the same as grep -E. fgrep is the same as grep -F. Direct invocation as either egrep or fgrep is deprecated, but is provided to allow historical applications that rely on them to run unmodified.
Options
Generic Program Information
--help Print a usage message briefly summarizing these command-line options and the bug-reporting address, then exit.
-V, --version Print the version number of grep to the standard output stream. This version number should be included in all bug reports (see below).
Matcher Selection
-E, --extended-regexp Interpret PATTERN as an extended regular expression (ERE, see below). (-E is specified by POSIX .)
-F, --fixed-strings Interpret PATTERN as a list of fixed strings, separated by newlines, any of which is to be matched. (-F is specified by POSIX .)
-G, --basic-regexp Interpret PATTERN as a basic regular expression (BRE, see below). This is the default.
-P, --perl-regexp Interpret PATTERN as a Perl regular expression. This is highly experimental and grep -P may warn of unimplemented features.
Matching Control
-e PATTERN, --regexp=PATTERN Use PATTERN as the pattern. This can be used to specify multiple search patterns, or to protect a pattern beginning with a hyphen (-).
(-e is specified by POSIX .)
-f FILE, --file=FILE Obtain patterns from FILE, one per line. The empty file contains zero patterns, and therefore matches nothing. (-f is specified by POSIX .)
-i, --ignore-case Ignore case distinctions in both the PATTERN and the input files. (-i is specified by POSIX .)
-v, --invert-match Invert the sense of matching, to select non-matching lines. (-v is specified by POSIX .)
-w, --word-regexp Select only those lines containing matches that form whole words. The test is that the matching substring must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.
-x, --line-regexp Select only those matches that exactly match the whole line. (-x is specified by POSIX .)
-y Obsolete synonym for -i.
General Output Control
-c, --count Suppress normal output; instead print a count of matching lines for each input file. With the -v, --invert-match option (see below), count non-matching lines. (-c is specified by POSIX .)
--color[=WHEN], --colour[=WHEN] Surround the matched (non-empty) strings, matching lines, context lines, file names, line numbers, byte offsets, and separators (for fields and groups of context lines) with escape sequences to display them in color on the terminal. The colors are defined by the environment variable GREP_COLORS. The deprecated environment variable GREP_COLOR is still supported, but its setting does not have priority. WHEN is never, always, or auto.
-L, --files-without-match Suppress normal output; instead print the name of each input file from which no output would normally have been printed. The scanning will stop on the first match.
-l, --files-with-matches Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match.
(-l is specified by POSIX .)
-m NUM, --max-count=NUM Stop reading a file after NUM matching lines. If the input is standard input from a regular file, and NUM matching lines are output, grep ensures that the standard input is positioned to just after the last matching line before exiting, regardless of the presence of trailing context lines. This enables a calling process to resume a search. When grep stops after NUM matching lines, it outputs any trailing context lines. When the -c or --count option is also used, grep does not output a count greater than NUM. When the -v or --invert-match option is also used, grep stops after outputting NUM non-matching lines.
-o, --only-matching Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.
-q, --quiet, --silent Quiet; do not write anything to standard output. Exit immediately with zero status if any match is found, even if an error was detected. Also see the -s or --no-messages option.
(-q is specified by POSIX .)
-s, --no-messages Suppress error messages about nonexistent or unreadable files.
Portability note: unlike GNU grep, 7th Edition Unix grep did not conform to POSIX , because it lacked -q and its -s option behaved like GNU grep's -q option. USG -style grep also lacked -q but its -s option behaved like GNU grep. Portable shell scripts should avoid both -q and -s and should redirect standard and error output to /dev/null instead.
(-s is specified by POSIX .)
Output Line Prefix Control
-b, --byte-offset Print the 0-based byte offset within the input file before each line of output. If -o (--only-matching) is specified, print the offset of the matching part itself.
-H, --with-filename Print the file name for each match. This is the default when there is more than one file to search.
-h, --no-filename Suppress the prefixing of file names on output. This is the default when there is only one file (or only standard input) to search.
--label=LABEL Display input actually coming from standard input as input coming from file LABEL. This is especially useful when implementing tools like zgrep, e.g., gzip -cd foo.gz | grep --label=foo -H something. See also the -H option.
-n, --line-number Prefix each line of output with the 1-based line number within its input file.
(-n is specified by POSIX .)
-T, --initial-tab Make sure that the first character of actual line content lies on a tab stop, so that the alignment of tabs looks normal. This is useful with options that prefix their output to the actual content: -H,-n, and -b.
In order to improve the probability that lines from a single file will all start at the same column, this also causes the line number and byte offset (if present) to be printed in a minimum size field width.
-u, --unix-byte-offsets Report Unix-style byte offsets. This switch causes grep to report byte offsets as if the file were a Unix-style text file, i.e., with CR characters stripped off. This will produce results identical to running grep on a Unix machine. This option has no effect unless -b option is also used; it has no effect on platforms other than MS-DOS and MS -Windows.
-Z, --null Output a zero byte (the ASCII NUL character) instead of the character that normally follows a file name. For example, grep -lZ outputs a zero byte after each file name instead of the usual newline. This option makes the output unambiguous, even in the presence of file names containing unusual characters like newlines. This option can be used with commands like find -print0, perl -0, sort -z, and xargs -0 to process arbitrary file names, even those that contain newline characters.
Context Line Control
-A NUM, --after-context=NUM Print NUM lines of trailing context after matching lines. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching option, this has no effect and a warning is given.
-B NUM, --before-context=NUM Print NUM lines of leading context before matching lines. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching option, this has no effect and a warning is given.
-C NUM, -NUM, --context=NUM Print NUM lines of output context. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching option, this has no effect and a warning is given.
File and Directory Selection
-a, --text Process a binary file as if it were text; this is equivalent to the --binary-files=text option.
--binary-files=TYPE If the first few bytes of a file indicate that the file contains binary data, assume that the file is of type TYPE. By default, TYPE is binary, and grep normally outputs either a one-line message saying that a binary file matches, or no message if there is no match. If TYPE is without-match, grep assumes that a binary file does not match; this is equivalent to the -I option. If TYPE is text, grep processes a binary file as if it were text; this is equivalent to the -a option. Warning: grep --binary-files=text might output binary garbage, which can have nasty side effects if the output is a terminal and if the terminal driver interprets some of it as commands.
-D ACTION, --devices=ACTION If an input file is a device, FIFO or socket, use ACTION to process it. By default, ACTION is read, which means that devices are read just as if they were ordinary files. If ACTION is skip, devices are silently skipped.
-d ACTION, --directories=ACTION If an input file is a directory, use ACTION to process it. By default, ACTION is read, which means that directories are read just as if they were ordinary files. If ACTION is skip, directories are silently skipped. If ACTION is recurse, grep reads all files under each directory, recursively; this is equivalent to the -r option.
--exclude=GLOB Skip files whose base name matches GLOB (using wildcard matching). A file-name glob can use *, ?, and [...] as wildcards, and \ to quote a wildcard or backslash character literally.
--exclude-from=FILE Skip files whose base name matches any of the file-name globs read from FILE (using wildcard matching as described under --exclude).
--exclude-dir=DIR Exclude directories matching the pattern DIR from recursive searches.
-I Process a binary file as if it did not contain matching data; this is equivalent to the --binary-files=without-match option.
--include=GLOB Search only files whose base name matches GLOB (using wildcard matching as described under --exclude).
-R, -r, --recursive Read all files under each directory, recursively; this is equivalent to the -d recurse option.
Other Options
--line-buffered Use line buffering on output. This can cause a performance penalty.
--mmap If possible, use the mmap(2) system call to read input, instead of the default read(2) system call. In some situations, --mmap yields better performance. However, --mmap can cause undefined behavior (including core dumps) if an input file shrinks while grep is operating, or if an I/O error occurs.
-U, --binary Treat the file(s) as binary. By default, under MS-DOS and MS -Windows, grep guesses the file type by looking at the contents of the first 32KB read from the file. If grep decides the file is a text file, it strips the CR characters from the original file contents (to make regular expressions with ^ and $ work correctly). Specifying -U overrules this guesswork, causing all files to be read and passed to the matching mechanism verbatim; if the file is a text file with CR/LF pairs at the end of each line, this will cause some regular expressions to fail. This option has no effect on platforms other than MS-DOS and MS -Windows.
-z, --null-data Treat the input as a set of lines, each terminated by a zero byte (the ASCII NUL character) instead of a newline. Like the -Z or --null option, this option can be used with commands like sort -z to process arbitrary file names.
Regular Expressions
A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.
grep understands three different versions of regular expression syntax: "basic," "extended" and "perl." In GNU grep, there is no difference in available functionality between basic and extended syntaxes. In other implementations, basic regular expressions are less powerful. The following description applies to extended regular expressions; differences for basic regular expressions are summarized afterwards. Perl regular expressions give additional functionality, and are documented in pcresyntax(3) and pcrepattern(3), but may not be available on every system.
The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any meta-character with special meaning may be quoted by preceding it with a backslash.
The period . matches any single character.
Character Classes and Bracket Expressions
A bracket expression is a list of characters enclosed by [and]. It matches any single character in that list; if the first character of the list is the caret ^ then it matches any character not in the list. For example, the regular expression [0123456789] matches any single digit.
Within a bracket expression, a range expression consists of two characters separated by a hyphen. It matches any single character that sorts between the two characters, inclusive, using the locale's collating sequence and character set. For example, in the default C locale, [a-d] is equivalent to [abcd]. Many locales sort characters in dictionary order, and in these locales [a-d] is typically not equivalent to [abcd]; it might be equivalent to [aBbCcDd], for example. To obtain the traditional interpretation of bracket expressions, you can use the C locale by setting the LC_ALL environment variable to the value C.
Finally, certain named classes of characters are predefined within bracket expressions, as follows. Their names are self explanatory, and they are [:alnum:], [:alpha:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], and [:xdigit:]. For example, [[:alnum:]] means [0-9A-Za-z], except the latter form depends upon the C locale and the ASCII character encoding, whereas the former is independent of locale and character set. (Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket expression.) Most meta-characters lose their special meaning inside bracket expressions. To include a literal] place it first in the list. Similarly, to include a literal ^ place it anywhere but first. Finally, to include a literal - place it last.
Anchoring
The caret ^ and the dollar sign $ are meta-characters that respectively match the empty string at the beginning and end of a line. The Backslash Character and Special Expressions
The symbols \< and \> respectively match the empty string at the beginning and end of a word. The symbol \b matches the empty string at the edge of a word, and \B matches the empty string provided it's not at the edge of a word. The symbol \w is a synonym for [[:alnum:]] and \W is a synonym for [^[:alnum:]].
Repetition
A regular expression may be followed by one of several repetition operators: ? The preceding item is optional and matched at most once. * The preceding item will be matched zero or more times. + The preceding item will be matched one or more times. {n} The preceding item is matched exactly n times. {n,} The preceding item is matched n or more times. {,m} The preceding item is matched at most m times. {n,m} The preceding item is matched at least n times, but not more than m times.
Concatenation
Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated expressions.
Alternation
Two regular expressions may be joined by the infix operator |; the resulting regular expression matches any string matching either alternate expression.
Precedence
Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole expression may be enclosed in parentheses to override these precedence rules and form a subexpression.
Back References and Subexpressions
The back-reference , where n is a single digit, matches the substring previously matched by the nth parenthesized subexpression of the regular expression.
Basic vs Extended Regular Expressions
In basic regular expressions the meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(, and \).
Traditional egrep did not support the { meta-character, and some egrep implementations support \{ instead, so portable scripts should avoid { in grep -E patterns and should use [{] to match a literal {. GNU grep -E attempts to support traditional usage by assuming that { is not special if it would be the start of an invalid interval specification. For example, the command grep -E '{1' searches for the two-character string {1 instead of reporting a syntax error in the regular expression. POSIX.2 allows this behavior as an extension, but portable scripts should avoid it.
Environment Variables
The behavior of grep is affected by the following environment variables.
The locale for category LC_foo is specified by examining the three environment variables LC_ALL, LC_foo, LANG, in that order. The first of these variables that is set specifies the locale. For example, if LC_ALL is not set, but LC_MESSAGES is set to pt_BR, then the Brazilian Portuguese locale is used for the LC_MESSAGES category. The C locale is used if none of these environment variables are set, if the locale catalog is not installed, or if grep was not compiled with national language support (NLS).
GREP_OPTIONS This variable specifies default options to be placed in front of any explicit options. For example, if GREP_OPTIONS is '--binary-files=without-match --directories=skip', grep behaves as if the two options --binary-files=without-match and --directories=skip had been specified before any explicit options. Option specifications are separated by whitespace. A backslash escapes the next character, so it can be used to specify an option containing whitespace or a backslash.
GREP_COLOR This variable specifies the color used to highlight matched (non-empty) text. It is deprecated in favor of GREP_COLORS, but still supported. The mt, ms, and mc capabilities of GREP_COLORS have priority over it. It can only specify the color used to highlight the matching non-empty text in any matching line (a selected line when the -v command-line option is omitted, or a context line when -v is specified). The default is 01;31, which means a bold red foreground text on the terminal's default background.
GREP_COLORS Specifies the colors and other attributes used to highlight various parts of the output. Its value is a colon-separated list of capabilities that defaults to ms=01;31:mc=01;31:sl=:cx=:fn=35:ln=32:bn=32:se=36 with the rv and ne boolean capabilities omitted (i.e., false). Supported capabilities are as follows.
sl= SGR substring for whole selected lines (i.e., matching lines when the -v command-line option is omitted, or non-matching lines when -v is specified). If however the boolean rv capability and the -v command-line option are both specified, it applies to context matching lines instead. The default is empty (i.e., the terminal's default color pair).
cx= SGR substring for whole context lines (i.e., non-matching lines when the -v command-line option is omitted, or matching lines when -v is specified). If however the boolean rv capability and the -v command-line option are both specified, it applies to selected non-matching lines instead. The default is empty (i.e., the terminal's default color pair).
rv Boolean value that reverses (swaps) the meanings of the sl= and cx= capabilities when the -v command-line option is specified. The default is false (i.e., the capability is omitted).
mt=01;31 SGR substring for matching non-empty text in any matching line (i.e., a selected line when the -v command-line option is omitted, or a context line when -v is specified). Setting this is equivalent to setting both ms= and mc= at once to the same value. The default is a bold red text foreground over the current line background.
ms=01;31 SGR substring for matching non-empty text in a selected line. (This is only used when the -v command-line option is omitted.) The effect of the sl= (or cx= if rv) capability remains active when this kicks in. The default is a bold red text foreground over the current line background.
mc=01;31 SGR substring for matching non-empty text in a context line. (This is only used when the -v command-line option is specified.) The effect of the cx= (or sl= if rv) capability remains active when this kicks in. The default is a bold red text foreground over the current line background.
fn=35 SGR substring for file names prefixing any content line. The default is a magenta text foreground over the terminal's default background.
ln=32 SGR substring for line numbers prefixing any content line. The default is a green text foreground over the terminal's default background.
bn=32 SGR substring for byte offsets prefixing any content line. The default is a green text foreground over the terminal's default background.
se=36 SGR substring for separators that are inserted between selected line fields (:), between context line fields, (-), and between groups of adjacent lines when nonzero context is specified (--). The default is a cyan text foreground over the terminal's default background.
ne Boolean value that prevents clearing to the end of line using Erase in Line (EL) to Right (\33[K) each time a colorized item ends. This is needed on terminals on which EL is not supported. It is otherwise useful on terminals for which the back_color_erase (bce) boolean terminfo capability does not apply, when the chosen highlight colors do not affect the background, or when EL is too slow or causes too much flicker. The default is false (i.e., the capability is omitted).
Note that boolean capabilities have no =... part. They are omitted (i.e., false) by default and become true when specified.
See the Select Graphic Rendition (SGR) section in the documentation of the text terminal that is used for permitted values and their meaning as character attributes. These substring values are integers in decimal representation and can be concatenated with semicolons. grep takes care of assembling the result into a complete SGR sequence (\33[...m). Common values to concatenate include 1 for bold, 4 for underline, 5 for blink, 7 for inverse, 39 for default foreground color, 30 to 37 for foreground colors, 90 to 97 for 16-color mode foreground colors, 38;5;0 to 38;5;255 for 88-color and 256-color modes foreground colors, 49 for default background color, 40 to 47 for background colors, 100 to 107 for 16-color mode background colors, and 48;5;0 to 48;5;255 for 88-color and 256-color modes background colors.
LC_ALL, LC_COLLATE, LANG These variables specify the locale for the LC_COLLATE category, which determines the collating sequence used to interpret range expressions like [a-z].
LC_ALL, LC_CTYPE, LANG These variables specify the locale for the LC_CTYPE category, which determines the type of characters, e.g., which characters are whitespace.
LC_ALL, LC_MESSAGES, LANG These variables specify the locale for the LC_MESSAGES category, which determines the language that grep uses for messages. The default C locale uses American English messages.
POSIXLY_CORRECT If set, grep behaves as POSIX.2 requires; otherwise, grep behaves more like other GNU programs. POSIX.2 requires that options that follow file names must be treated as file names; by default, such options are permuted to the front of the operand list and are treated as options. Also, POSIX.2 requires that unrecognized options be diagnosed as "illegal", but since they are not really against the law the default is to diagnose them as "invalid".
POSIXLY_CORRECT also disables _N_GNU_nonoption_argv_flags_, described below.
_N_GNU_nonoption_argv_flags_ (Here N is grep's numeric process ID.) If the ith character of this environment variable's value is 1, do not consider the ith operand of grep to be an option, even if it appears to be one. A shell can put this variable in the environment for each command it runs, specifying which operands are the results of file name wildcard expansion and therefore should not be treated as options. This behavior is available only with the GNU C library, and only when POSIXLY_CORRECT is not set.
Exit Status
Normally, the exit status is 0 if selected lines are found and 1 otherwise. But the exit status is 2 if an error occurred, unless the -q or --quiet or --silent option is used and a selected line is found. Note, however, that POSIX only mandates, for programs such as grep, cmp, and diff, that the exit status in case of error be greater than 1; it is therefore advisable, for the sake of portability, to use logic that tests for this general condition instead of strict equality with 2.
Copyright
Copyright 1998-2000, 2002, 2005-2010 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Bugs
Reporting Bugs
Email bug reports to , a mailing list whose web page is < .
grep's Savannah bug tracker is located at < .
Known Bugs
Large repetition counts in the {n,m} construct may cause grep to use lots of memory. In addition, certain other obscure regular expressions require exponential time and space, and may cause grep to run out of memory.
Back-references are very slow, and may require exponential time.
See Also
Regular Manual Pages
awk(1), cmp(1), diff(1), find(1), gzip(1), perl(1), sed(1), sort(1), xargs(1), zgrep(1), mmap(2), read(2), pcre(3), pcresyntax(3), pcrepattern(3), terminfo(5), glob(7), regex(7).
POSIX Programmer's Manual Page
grep(1p).
TeXinfo Documentation
The full documentation for grep is maintained as a TeXinfo manual. If the info and grep programs are properly installed at your site, the command info grep should give you access to the complete manual.
Notes
GNU 's not Unix, but Unix is a beast; its plural form is Unixen.
Referenced By
bzgrep(1), flowdumper(1), fortune(6), gnome-search-tool(1), grepmail(1), ip(8), ksh93(1), look(1), makeindex(1), mirrordir(1), mksh(1), nawk(1), nget(1), pdsh(1), perlfunc(1), perlglossary(1), procmail(1), procmailex(5), procmailrc(5), procmailsc(5), quilt(1), regex(3), sudo(8), sudoers(5), tcpstat(1), trace-cmd-record(1), uwildmat(3), wildmat(3), xzgrep(1)