grep, searching for a pattern

version 2.4, 13 November 1999

Alain Magloire et al.

Published by the Free Software Foundation,
59 Temple Place - Suite 330,
Boston, MA 02111-1307, USA

1 Introduction

`grep` searches the input files for lines containing a match to a given pattern list. When it finds a match in a line, it copies the line to standard output (by default), or does whatever other sort of output you have requested with options. `grep` expects to do the matching on text. Since newline is also a separator for the list of patterns, there is no way to match newline characters in a text.

2 Invoking grep

`grep` comes with a rich set of options from POSIX.2 and GNU extensions.

'`-c`'
'`--count`' Suppress normal output; instead print a count of matching lines for each input file. With the '`-v`', '`--invert-match`' option, count non-matching lines.

'`-e` *pattern*'
'`--regexp=`*pattern*'
Use *pattern* as the pattern; useful to protect patterns beginning with a '`-`'.

'`-f` *file*'
'`--file=`*file*'
Obtain patterns from *file*, one per line. The empty file contains zero patterns, and therefore matches nothing.

'`-i`'
'`--ignore-case`'
Ignore case distinctions in both the pattern and the input files.

'`-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 of every file will stop on the first match.

'`-n`'
'`--line-number`'
Prefix each line of output with the line number within its input file.

'`-q`'
'`--quiet`'
'`--silent`'
Quiet; suppress normal output. The scanning of every file will stop on the first match. Also see the '`-s`' or '`--no-messages`' option.

'`-s`'
'`--no-messages`'
Suppress error messages about nonexistent or unreadable files. Portability note: unlike GNU `grep`, traditional `grep` did not conform to POSIX.2, because traditional `grep` lacked a '`-q`' option and its '`-s`' option behaved like GNU `grep`'s '`-q`' option. Shell scripts intended to be portable to traditional `grep` should avoid both '`-q`' and '`-s`' and should redirect output to '`/dev/null`' instead.

‘`-v`’
‘`--invert-match`’
: Invert the sense of matching, to select non-matching lines.

‘`-x`’
‘`--line-regexp`’
: Select only those matches that exactly match the whole line.

2.1 GNU Extensions

‘`-A` *num*’
‘`--after-context=`*num*’
: Print *num* lines of trailing context after matching lines.

‘`-B` *num*’
‘`--before-context=`*num*’
: Print *num* lines of leading context before matching lines.

‘`-C` *num*’
‘`--context=[`*num*`]`’
: Print *num* lines (default 2) of output context.

‘`-`*num*’
: Same as ‘`--context=`*num*’ lines of leading and trailing context. However, grep will never print any given line more than once.

‘`-V`’
‘`--version`’
: Print the version number of `grep` to the standard output stream. This version number should be included in all bug reports.

‘`--help`’
: Print a usage message briefly summarizing these command-line options and the bug-reporting address, then exit.

‘`-b`’
‘`--byte-offset`’
: Print the byte offset within the input file before each line of output. When `grep` runs on MS-DOS or MS-Windows, the printed byte offsets depend on whether the ‘`-u`’ (‘`--unix-byte-offsets`’) option is used; see below.

‘`-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 (some operating systems and filesystems disallow this, and will cause `grep` to print error messages for every directory). 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.

‘`-H`’
‘`--with-filename`’
: Print the filename for each match.

'-h'
'--no-filename'
Suppress the prefixing of filenames on output when multiple files are searched.

'-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 of every file will stop on the first match.

'-a'
'--text' Do not suppress output lines that contain binary data. Normally, if the first few bytes of a file indicate that the file contains binary data, grep outputs only a message saying that the file matches the pattern. This option causes grep to act as if the file is a text file, even if it would otherwise be treated as binary. *Warning:* the result might be binary garbage printed to the terminal, which can have nasty side-effects if the terminal driver interprets some of it as commands.

'-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.

'-r'
'--recursive'
For each directory mentioned in the command line, read and process all files in that directory, recursively. This is the same as the '-d recurse' option.

'-y' Obsolete synonym for '-i'.

'-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.

'-u'
'--unix-byte-offsets'
Report Unix-style byte offsets. This switch causes `grep` to report byte offsets as if the file were Unix style text file, i.e., the byte offsets ignore the `CR` characters which were stripped. 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.

'--mmap' If possible, use the mmap system call to read input, instead of the default read 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.

'-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.

'-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.

Several additional options control which variant of the grep matching engine is used. See Chapter 4 [Grep Programs], page 5.

2.2 Environment Variables

Grep's behavior is affected by the following environment variables.

GREP_OPTIONS
This variable specifies default options to be placed in front of any explicit options. For example, if GREP_OPTIONS is '--text --directories=skip', grep behaves as if the two options '--text' 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.

LC_ALL
LC_MESSAGES
LANG These variables specify the LC_MESSAGES locale, which determines the language that grep uses for messages. The locale is determined by the first of these variables that is set. American English is used if none of these environment variables are set, or if the message catalog is not installed, or if grep was not compiled with national language support (NLS).

LC_ALL
LC_CTYPE
LANG These variables specify the LC_CTYPE locale, which determines the type of characters, e.g., which characters are whitespace. The locale is determined by the first of these variables that is set. The POSIX locale is used if none of these environment variables are set, or if the locale catalog is not installed, or if grep was not compiled with national language support (NLS).

`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 *i*th character of this environment variable's value is '`1`', do not consider the *i*th 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.

3 Diagnostics

Normally, exit status is 0 if matches were found, and 1 if no matches were found (the '`-v`' option inverts the sense of the exit status). Exit status is 2 if there were syntax errors in the pattern, inaccessible input files, or other system errors.

4 grep programs

`grep` searches the named input files (or standard input if no files are named, or the file name '`-`' is given) for lines containing a match to the given pattern. By default, `grep` prints the matching lines. There are three major variants of `grep`, controlled by the following options.

'`-G`'
'`--basic-regexp`'
: Interpret pattern as a basic regular expression. This is the default.

'`-E`'
'`--extended-regexp`'
: Interpret pattern as an extended regular expression.

'`-F`'
'`--fixed-strings`'
: Interpret pattern as a list of fixed strings, separated by newlines, any of which is to be matched.

In addition, two variant programs EGREP and FGREP are available. EGREP is the same as '`grep -E`'. FGREP is the same as '`grep -F`'.

5 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 two different versions of regular expression syntax: "basic" and "extended". In GNU `grep`, there is no difference in available functionality using either syntax. 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.

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 metacharacter with special meaning may be quoted by preceding it with a backslash. A list of characters enclosed by '`[`' and '`]`' 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. A range of ASCII characters may be specified by giving the first and last characters, separated by a hyphen.

Finally, certain named classes of characters are predefined, as follows. Their interpretation depends on the `LC_CTYPE` locale; the interpretation below is that of the POSIX locale, which is the default if no `LC_CTYPE` locale is specified.

'`[:alnum:]`'
: Any of '`[:digit:]`' or '`[:alpha:]`'

'`[:alpha:]`'
: Any letter:
`a b c d e f g h i j k l m n o p q r s t u v w x y z`,
`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`.

'`[:blank:]`'
: Space or tab.

'`[:cntrl:]`'
: Any character with octal codes 000 through 037, or `DEL` (octal code 177).

'`[:digit:]`'
: Any one of `0 1 2 3 4 5 6 7 8 9`.

'`[:graph:]`'
: Anything that is not a '`[:alnum:]`' or '`[:punct:]`'.

'`[:lower:]`'
: Any one of `a b c d e f g h i j k l m n o p q r s t u v w x y z`.

'`[:print:]`'
: Any character from the '`[:space:]`' class, and any character that is **not** in the '`[:graph:]`' class.

'`[:punct:]`'
: Any one of `! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~`.

'`[:space:]`'
: Any one of `CR FF HT NL VT SPACE`.

'[:upper:]'
: Any one of A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.

'[:xdigit:]'
: Any one of a b c d e f A B C D E F 0 1 2 3 4 5 6 7 8 9.

For example, '[[:alnum:]]' means '[0-9A-Za-z]', except the latter form is dependent upon the ASCII character encoding, whereas the former is portable. (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 list.) Most metacharacters lose their special meaning inside lists. 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.

The period '.' matches any single character. The symbol '\w' is a synonym for '[[:alnum:]]' and '\W' is a synonym for '[^[:alnum]]'.

The caret '^' and the dollar sign '$' are metacharacters that respectively match the empty string at the beginning and end of a line. 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.

A regular expression may be followed by one of several repetition operators:

'?' The preceding item is optional and will be 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.

'{*n*,*m*}' The preceding item is matched at least *n* times, but not more than *m* times.

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated subexpressions.

Two regular expressions may be joined by the infix operator '|'; the resulting regular expression matches any string matching either subexpression.

Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole subexpression may be enclosed in parentheses to override these precedence rules.

The backreference '*n*', where *n* is a single digit, matches the substring previously matched by the *n*th parenthesized subexpression of the regular expression.

In basic regular expressions the metacharacters '?', '+', '{', '|', '(', and ')' lose their special meaning; instead use the backslashed versions '\?', '\+', '\{', '\|', '\(', and '\)'.

Traditional `egrep` did not support the '{' metacharacter, and some `egrep` implementations support '\{' instead, so portable scripts should avoid '{' in 'egrep' patterns and should use '[{]' to match a literal '{'.

GNU `egrep` 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 shell command '`egrep '{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.

6 Usage

Here is an example shell command that invokes GNU `grep`:

```
grep -i 'hello.*world' menu.h main.c
```

This lists all lines in the files '`menu.h`' and '`main.c`' that contain the string '`hello`' followed by the string '`world`'; this is because '`.*`' matches zero or more characters within a line. See Chapter 5 [Regular Expressions], page 6. The '`-i`' option causes `grep` to ignore case, causing it to match the line '`Hello, world!`', which it would not otherwise match. See Chapter 2 [Invoking], page 1, for more details about how to invoke `grep`.

Here are some common questions and answers about `grep` usage.

1. How can I list just the names of matching files?

   ```
   grep -l 'main' *.c
   ```

   lists the names of all C files in the current directory whose contents mention '`main`'.

2. How do I search directories recursively?

   ```
   grep -r 'hello' /home/gigi
   ```

   searches for '`hello`' in all files under the directory '`/home/gigi`'. For more control of which files are searched, use `find`, `grep` and `xargs`. For example, the following command searches only C files:

   ```
   find /home/gigi -name '*.c' -print | xargs grep 'hello' /dev/null
   ```

3. What if a pattern has a leading '`-`'?

   ```
   grep -e '--cut here--' *
   ```

   searches for all lines matching '`--cut here--`'. Without '`-e`', `grep` would attempt to parse '`--cut here--`' as a list of options.

4. Suppose I want to search for a whole word, not a part of a word?

   ```
   grep -w 'hello' *
   ```

   searches only for instances of '`hello`' that are entire words; it does not match '`Othello`'. For more control, use '`\<`' and '`\>`' to match the start and end of words. For example:

   ```
   grep 'hello\>' *
   ```

   searches only for words ending in '`hello`', so it matches the word '`Othello`'.

5. How do I output context around the matching lines?

   ```
   grep -C 2 'hello' *
   ```

   prints two lines of context around each matching line.

6. How do I force grep to print the name of the file?

   Append '`/dev/null`':

   ```
   grep 'eli' /etc/passwd /dev/null
   ```

7. Why do people use strange regular expressions on `ps` output?

```
ps -ef | grep '[c]ron'
```

If the pattern had been written without the square brackets, it would have matched not only the `ps` output line for `cron`, but also the `ps` output line for `grep`.

8. Why does `grep` report "Binary file matches"?

   If `grep` listed all matching "lines" from a binary file, it would probably generate output that is not useful, and it might even muck up your display. So GNU `grep` suppresses output from files that appear to be binary files. To force GNU `grep` to output lines even from files that appear to be binary, use the '`-a`' or '`--text`' option.

9. Why doesn't '`grep -lv`' print nonmatching file names?

   '`grep -lv`' lists the names of all files containing one or more lines that do not match. To list the names of all files that contain no matching lines, use the '`-L`' or '`--files-without-match`' option.

10. I can do OR with '`|`', but what about AND?

    ```
    grep 'paul' /etc/motd | grep 'franc,ois'
    ```

    finds all lines that contain both '`paul`' and '`franc,ois`'.

11. How can I search in both standard input and in files?

    Use the special file name '`-`':

    ```
    cat /etc/passwd | grep 'alain' - /etc/motd
    ```

7 Reporting bugs

Email bug reports to `bug-gnu-utils@gnu.org`. Be sure to include the word "grep" somewhere in the "Subject:" field.

Large repetition counts in the '`{m,n}`' 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. Backreferences are very slow, and may require exponential time.

Concept Index

This is a general index of all issues discussed in this manual, with the exception of the `grep` commands and command-line options.

P

Q

R

S

T

U

V

W

X

Z

Index

This is an alphabetical list of all `grep` commands, command-line options, and environment variables.

*

-

?

_

{

+

A

B

C

D

G

L

P

S

U

X

Table of Contents