GNU Grep Simulation

Print lines matching patterns

Avik Sinha Roy, Saradindu Sengupta, Somnath Paul

31/05/2016

This manual is for `grep`, a pattern matching engine.

ACHARIYA PRAFULLA CHANDRA COLLEGE

Certificate of Originality

This is to certify that the Project Report entitled “PAYROLL” submitted to ACHARIYA PRAFULLA CHANDRA COLLEGE in partial fulfillment of the requirements for the award of the degree of BSc, is an authentic and original work carried out under my guidance by the following students with University roll no.

Name	University Roll No
Avik Sinha Roy	**10111123214001422**
Saradindu Sengupta	**10111123212001435**
Somnath Paul	**10111123214001416**

The matter embodied in this project is genuine work done by the student and has not submitted whether to this Institute or to any other University /Institute for the fulfillment of the requirements of any course of study.
This project is here by approved as accreditable performance for a group of Final Year B.Sc. students. It is carried out and presented in a manner will satisfactory to warrant it’s acceptance as a prerequisite to the B.Sc. for which it has been submitted. It is understood that by approval of this project Report the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the Project Report for the purpose it has been submitted.

Name & Signature of the Student with Date:

Name : Avik Sinha Roy Signature:

Name : Saradindu Sengupta Signature:

Name: Somnath Paul Signature:

Signature of the Guide with Date:

Mr. Joydeb Das Biswas
Project Guide
Department of CMSA

CERTIFICATE OF APPROVAL

This is to certify that Avik Sinha Roy, Saradindu Sengupta, Somnath Paul are the students of Acharya Prafulla Chandra College, have successfully completed a project on “GNU Grep simulation” in part III final examination at Department of Computer Science.

This report has not been submitted to any other Organization & does not form part of any Course undergone by then, for the award of Bachelor of Computer Science.

Signature of the HOD with Date:

Prof. Joydeb Das Biswas
Head of the Department.
Department of CMSA

ACKNOWLEDGEMENT :

It is our privilege to express our sincerest regards to our project coordinator, Mr. Joydeb Das Biswas, for his valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project.

We deeply express our sincere thanks to our Head of Department Prof. Joydeb Das Biswas for encouraging and allowing us to present the project on the topic “GNU Grep Simulation” at our department premises for the partial fulfillment of the requirements leading to the award of B.Sc.

We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our career. Last but not the least we express our thanks to our friends for their cooperation and support.

Name & Signature of the Student with Date:

Name : Avik Sinha Roy **Signature:**

Name : Saradindu Sengupta **Signature:**

Name : Somnath Paul **Signature:**

Table of Contents

1 Introduction

`grep` searches 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 produces whatever other sort of output you have requested with options.

Though `grep` expects to do the matching on text, it has no limits on input line length other than available memory, and it can match arbitrary characters within a line. If the final byte of an input file is not a newline, `grep` silently supplies one. 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

The general synopsis of the `grep` command line is

```
grep options pattern input_file_names
```

There can be zero or more options. *`pattern`* will only be seen as such (and not as an *`input_file_name`*) if it wasn't already specified within options (by using the '`-e pattern`' or '`-f file`' options). There can be zero or more *`input_file_names`*.

2.1 Command-line Options

`grep` comes with a rich set of options: some from POSIX and some being GNU extensions. Long option names are always a GNU extension, even for options that are from POSIX specifications. Options that are specified by POSIX, under their short names, are explicitly marked as such to facilitate POSIX-portable programming. A few option names are provided for compatibility with older or more exotic implementations. Several additional options control which are variant of the `grep` matching engine is used.

2.1.1 Generic Program Information

`--help` Print a usage message briefly summarizing the 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.

2.1.2 Matching Control

`-e` *`pattern`*

`--regexp=`*`pattern`*

Use *`pattern`* as the pattern. If this option is used multiple times or is combined with the `-f` (`--file`) option, search for all patterns given. (`-e` is specified by POSIX.)

`-f` *`file`*

`--file=`*`file`*

Obtain patterns from *`file`*, one per line. If this option is used multiple times or is combined with the `-e` (`--regexp`) option, search for all patterns given. The

empty file contains zero patterns, and therefore matches nothing. (`-f` is specified by POSIX.)

`-i`

`-y`

`--ignore-case`

Ignore case distinctions, so that characters that differ only in case match each other. Although this is straightforward when letters differ in case only via lowercase-uppercase pairs, the behavior is unspecified in other situations. For example, uppercase “S” has an unusual lowercase counterpart “” (Unicode character U+017F, LATIN SMALL LETTER LONG S) in many locales, and it is unspecified whether this unusual character matches “S” or “s” even though uppercasing it yields “S”. Another example: the lowercase German letter “ß” (U+00DF, LATIN SMALL LETTER SHARP S) is normally capitalized as the two-character string “SS” but it does not match “SS”, and it might not match the uppercase letter “” (U+1E9E, LATIN CAPITAL LETTER SHARP S) even though lowercasing the latter yields the former. `-y` is an obsolete synonym that is provided for compatibility. (`-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. This option has no effect if `-x` is also specified.

`-x`

`--line-regexp`

Select only those matches that exactly match the whole line. For a regular expression pattern, this is like parenthesizing the pattern and then surrounding it with ‘`^`’ and ‘`$`’. (`-x` is specified by POSIX.)

2.1.3 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, 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 colours are defined by the environment variable `GREP_COLORS` and default to ‘`ms=01;31:mc=01;31:sl=:cx=:fn=35:ln=32:bn=32:se=36`’ for bold red matched text, magenta file names, green line numbers, green byte offsets, cyan separators, and default terminal colors otherwise. The deprecated environment variable `GREP_COLORS` is still supported, but its setting does not have priority; it defaults to ‘`01;31`’ (bold red) which only covers the color for matched text. `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 of each file stops 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 of each file stops 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 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. For example, the following shell script makes use of it:

```
while grep -m 1 PATTERN
do
echo xxxx
done < FILE
```

But the following probably will not work because a pipe is not a regular file :
This probably will not work.

```
cat FILE |
while grep -m 1 PATTERN
do
echo xxxx
done
```

When `grep` stops after *num* matching lines, it outputs any trailing context lines. Since context does not include matching lines, `grep` will stop when it encounters another matching line. 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 matching lines, 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`'s. 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.)

2.1.4 Output Line Prefix Control

When several prefix fields are to be output, the order is always file name, line number, and byte offset, regardless of the order in which these options were specified.

`-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. When `grep` runs on MS-DOS or MS-Windows, the printed byte offsets depend on whether the `-u (--unix-byte-offsets)` option is used.

`-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
```

`-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 option causes `grep` to report byte offsets as if the file were a Unix-style text file, i.e., the byte offsets ignore carriage returns that were stripped. This will produce results identical to running `grep` on a Unix machine. This option has no effect unless the `-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 NULL 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.

2.1.5 Context Line Control

Regardless of how these options are set, `grep` will never print any given line more than once. If the `-o (--only-matching)` option is specified, these options have no effect and a warning is given upon their use.

`-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`*
`-num`
`--context=`*`num`*

Print *`num`* lines of leading and trailing output context.

`--group-separator=`*`string`*

When `-A, -B` or `-C` are in use, print *`string`* instead of `--` between groups of lines.

`--no-group-separator`

When `-A, -B` or `-C` are in use, do not print a separator between groups of lines.

Here are some points about how `grep` chooses the separator to print between prefix fields and line content :

• Matching lines normally use '`:`' as a separator between prefix fields and actual line content.
• Context (i.e., non-matching) lines use '`-`' instead.
• When context is not specified, matching lines are simply output one right after another.
• When context is specified, lines that are adjacent in the input form a group and are output one right after another, while by default a separator appears between nonadjacent groups.

• The default separator is a '`--`' line; its presence and appearance can be changed with the options above.
• Each group may contain several matching lines when they are close enough to each other that two adjacent groups connect and can merge into a single contiguous one.

2.1.6 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 a file's data or metadata indicate that the file contains binary data, assume that the file is of type *`type`*. Non-text bytes indicate binary data; these are either output bytes that are improperly encoded for the current locale or null input bytes when the `–z` (`--null-data`) option is not given.
By default, *`type`* is '`binary`', and when `grep` discovers that a file is binary, it suppresses any further output, and instead, 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`', when `grep` discovers that a file is `binary` it assumes that the rest of the `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.

When *`type`* is '`binary`', `grep` may treat non-text bytes as line terminators even without the `-z (--null-data)` option. This means choosing '`binary`' versus '`text`' can affect whether a pattern matches a file. For example, when *`type`* is '`binary`' the pattern '`q$`' might match '`q`' immediately followed by a null byte, even though this is not matched when *`type`* is '`text`'. Conversely, when *`type`* is '`binary`', the pattern '`.`' (period) might not match a null byte.

`Warning:` The `-a (--binary-files=`*`text`*`)` option 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. On the other hand, when reading files whose text encodings are unknown, it can be helpful to use `-a` or to set '`LC_ALL='C''` in the environment, in order to find more matches even if the matches are unsafe for direct display.

`-D` *`action`*
`--devices=`*`action`*

If an input file is a device, `FIFO`, or `socket`, use *`action`* to process it. If *`action`* is 'read', all devices are read just as if they were ordinary files. If

action is ‘skip’, devices, FIFOs, and sockets are silently skipped. By default, devices are read if they are on the command line or if the `-R (--dereference-recursive)` option is used, and are skipped if they are encountered recursively and the `-r(--recursive)` option is used. This option has no effect on a file that is read via standard input.

`-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 file systems disallow this, and will cause `grep` to print error messages for every directory or silently skip them). If *action* is ‘`skip`’, directories are silently skipped. If *action* is ‘`recurse`’, `grep` reads all files under each directory, recursively, following command-line symbolic links and skipping other symlinks; this is equivalent to the `-r` option.

`--exclude=`*glob*

Skip any command-line file with a name suffix that matches the pattern *glob*, using wildcard matching; a name suffix is either the whole name, or any suffix starting after a ‘`/`’ and before a non-‘`/`’. When searching recursively, skip any sub-file whose base name matches *glob*; the base name is the part after the last ‘`/`’. A pattern can use ‘`*`’, ‘`?`’, and ‘`[`’...‘`]`’ as wildcards, and \ to quote a wildcard or backslash character literally.

`--exclude-from=`*file*

Skip files whose name matches any of the patterns read from *file* (using wildcard matching as described under `--exclude`).

`--exclude-dir=`*glob*

Skip any command-line directory with a name suffix that matches the pattern *glob*. When searching recursively, skip any subdirectory whose base name matches *glob*. Ignore any redundant trailing slashes in *glob*.

`-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 name matches *`glob`*, using wildcard matching as described under `--exclude`.

`-r`
`--recursive`

For each directory operand, read and process all files in that directory, recursively. Follow symbolic links on the command line, but skip symlinks that are encountered recursively. Note that if no file operand is given, `grep` searches the working directory. This is the same as the '`--directories=recurse`' option.

`-R`
`--dereference-recursive`

For each directory operand, read and process all files in that directory, recursively, following all symbolic links.

2.1.7 Other Options

`--line-buffered`

Use line buffering on output. This can cause a performance penalty.

`-U`
`--binary`

Treat the file(s) as `binary`. By default, under MS-DOS and MS-Windows, `grep` guesses whether a file is `text` or `binary` as described for the `--binary-files` option. If `grep` decides the `file` is a `text` file, it strips carriage returns 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 input and output data as sequences of lines, each terminated by a zero byte (the ASCII NULL character) instead of a newline.

2.2 Exit Status

Normally the exit status is 0 if a line is selected, 1 if no lines were selected, and 2 if an error occurred. However, if the `-q` or `--quiet` or `--silent` option is used and a line is selected, the exit status is 0 even if an error occurred. Other `grep` implementations may exit with status greater than 2 on error.

2.3 `grep` Programs

`grep` searches the named input files for lines containing a match to the given pattern. By default, `grep` prints the matching lines. A file named `-` stands for standard input. If no input is specified, `grep` searches the working directory. If given a command-line option specifying recursion; otherwise, `grep` searches standard input. There are four major variants of `grep`, controlled by the following options.

`-G`
`--basic-regexp`

> Interpret the pattern as a basic regular expression (BRE). This is the default.

`-E`
`--extended-regexp`

> Interpret the pattern as an extended regular expression (ERE). (`-E` is specified by POSIX.)

`-F`
`--fixed-strings`

> Interpret the pattern as a list of fixed strings (instead of regular expressions), separated by newlines, any of which is to be matched. (`-F` is specified by POSIX.)

`-P`
`--perl-regexp`

> Interpret the pattern as a Perl-compatible regular expression (PCRE). This is highly experimental and ‘`grep -P`’ may warn of unimplemented features.
> 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.

2.4 Options We Have Implemented

There are several options for the use of grep which we have implemented in this project. These are the following.

`-e` *`pattern`*

`--regexp=`*`pattern`*

> Use *`pattern`* as the pattern. If this option is used multiple times or is combined with the `-f` (`--file`) option, search for all patterns given. (`-e` is specified by POSIX.)

`-f` *`file`*

`--file=`*`file`*

> Obtain patterns from *`file`*, one per line. If this option is used multiple times or is combined with the `-e` (`--regexp`) option, search for all patterns given. The empty file contains zero patterns, and therefore matches nothing. (`-f` is specified by POSIX.)

`-i`

`-y`

`--ignore-case`

> Ignore case distinctions, so that characters that differ only in case match each other. Although this is straightforward when letters differ in case only via lowercase-uppercase pairs, the behavior is unspecified in other situations. For example, uppercase “S” has an unusual lowercase counterpart “” (Unicode character U+017F, LATIN SMALL LETTER LONG S) in many locales, and it is unspecified whether this unusual character matches “S” or “s” even though uppercasing it yields “S”. Another example: the lowercase German letter “ß” (U+00DF, LATIN SMALL LETTER SHARP S) is normally capitalized as the two-character string “SS” but it does not match “SS”, and it might not match the uppercase letter “” (U+1E9E, LATIN CAPITAL LETTER SHARP S) even though lowercasing the latter yields the former. `-y` is an obsolete synonym that is provided for compatibility. (`-i` is specified by POSIX.)

`-v`

`--invert-match`

Invert the sense of matching, to select non-matching lines. (`-v` is specified by POSIX.)

`-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. (`-c` is specified by POSIX.)

`-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 each file stops on the first match. (`-l` is specified by POSIX.)

`-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 each file stops on the first match.

`-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 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

`-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 the command-line options and the bug-reporting address, then exit.

`-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.

`-n`
`--line-number`

Prefix each line of output with the 1-based line number within its input file. (`-n` is specified by POSIX.)

2.5 Options We Intend to Implement in Future

The rest of the programs we have discussed about in previous sections(See Section 2.1) will be implemented in our simulation project.

3 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`” (BRE), “`extended`” (ERE) and “`perl`” (PCRE). In GNU `grep`, there is no difference in available functionality between the `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-compatible regular expressions give additional functionality, and are documented in the *pcresyntax* and *pcrepattern* manual pages, but work only if PCRE is available in the system.

3.1 Fundamental Structure

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.

A regular expression may be followed by one of the several repetition operators :
‘`.`’ The period ‘.’ matches any single character.
‘`?`’ 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.
‘`{,`*m*`}`’ The preceding item is matched at most *m* times. This is a GNU extension.
‘`{`*n*`,`*m*`}`’ The preceding item is matched at least *n* times, but not more than *m* times.

The empty regular expression matches the empty string. Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated expressions.
Two regular expressions may be joined by the infix operator ‘`|`’; the resulting regular expression matches any string matching either alternate expression.
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 sub-expression. An unmatched ‘`)`’ matches just itself.

3.2 Regular Expressions We have Implemented

The only regular expression operator we have implemented is the ‘`*`’ operator by which the preceding item will be matched for zero or more times.

For example:- `grep –i #include *.C`

Here, in ‘`.C`’, ‘`.`’ is not used as a regular expression operator but used as the extension name of input file. For this example, `grep` will search for the pattern ‘`#include`’ in all the files present in current working directory.

3.2 Regular Expressions We Intend to Implement in Future

The rest of the regular expressions we have discussed about in previous sections(See Section 3.1) will be implemented in our simulation project in future.