
Experiment 15

Aim: Filters using Regular Expressions , An Advanced Filter: Pattern Matching (grep), egrep, fgrep sed (-n, -e, -f, context addressing, writing selected lines to a file, deleting lines, substitution).

grep command in Unix/Linux

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).

Syntax:**grep [options] pattern [files]****Options Description**

- c** : This prints only a count of the lines that match a pattern
- h** : Display the matched lines, but do not display the filenames.
- i** : Ignores, case for matching
- l** : Displays list of a filenames only.
- n** : Display the matched lines and their line numbers.
- v** : This prints out all the lines that do not matches the pattern
- e exp** : Specifies expression with this option. Can use multiple times.
- f file** : Takes patterns from file, one per line.
- E** : Treats pattern as an extended regular expression (ERE)
- w** : Match whole word
- o** : Print only the matched parts of a matching line, with each such part on a separate output line.

Sample Commands

Consider the below file as an input.

```
$cat > geekfile.txt
```

```
unix is great os. unix is opensource. unix is free os.
```

```
learn operating system.
```

```
Unix linux which one you choose.
```

```
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

1. Case insensitive search : The -i option enables to search for a string case insensitively in the give file. It matches the words like “UNIX”, “Unix”, “unix”.

```
$grep -i "UNiX" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

Unix linux which one you choose.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

2. Displaying the count of number of matches: We can find the number of lines that matches the given string/pattern

```
$grep -c "unix" geekfile.txt
```

Output:

```
2
```

3. Display the file names that matches the pattern: We can just display the files that contains the given string/pattern.

```
$grep -l "unix" *
```

or

```
$grep -l "unix" f1.txt f2.txt f3.txt f4.txt
```

Output:

```
geekfile.txt
```

4. Checking for the whole words in a file: By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.

```
$ grep -w "unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

```
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
$ grep -o "unix" geekfile.txt
```

Output:

```
unix
unix
unix
unix
unix
unix
```

6. Show line number while displaying the output using grep -n : To show the line number of file with the line matched.

```
$ grep -n "unix" geekfile.txt
```

Output:

```
1:unix is great os. unix is opensource. unix is free os.
```

```
4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

7. Inverting the pattern match : You can display the lines that are not matched with the specified search sting pattern using the -v option.

```
$ grep -v "unix" geekfile.txt
```

Output:

```
learn operating system.
```

```
Unix linux which one you choose.
```

8. Matching the lines that start with a string : The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
$ grep "^unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

9. Matching the lines that end with a string : The \$ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

```
$ grep "os$" geekfile.txt
```

10.Specifies expression with -e option. Can use multiple times :

```
$grep -e "Agarwal" -e "Aggarwal" -e "Agrawal" geekfile.txt
```

11. -f file option Takes patterns from file, one per line.

```
$cat pattern.txt
```

```
Agarwal
```

```
Aggarwal
```

```
Agrawal
```

```
$grep -f pattern.txt geekfile.txt
```

The pattern is specified as a regular expression. A regular expression is a string of characters that is used to specify a pattern matching rule. Special characters are used to define the matching rules and positions.

#1) Anchor Characters: ‘^’ and ‘\$’ at the beginning and end of the pattern are used to anchor the pattern to the start of the line, and to the end of the line respectively.

Example: “^Name” matches all lines that start with the string “Name”. The strings “\<” and “\>” are used to anchor the pattern to the start and end of a word respectively.

#2) Wildcard Character: ‘.’ is used to match any character.

Example:“^.\$” will match all lines with any single character.

#3) Character Range: A set of characters enclosed in a '[' and ']' pair specify a range of characters to be matched.

Example: "[aeiou]" will match all lines that contain a vowel. A hyphen can be used while specifying a range to shorten a set of consecutive characters. E.g. "[0-9]" will match all lines that contain a digit. A caret can be used at the beginning of the range to specify a negative range. E.g. "[^xyz]" will match all lines that do not contain x, y or z.

#4) Repetition Modifier: A '*' after a character or group of characters is used to allow matching zero or more instances of the preceding pattern.

The grep command supports a number of options for additional controls on the matching:

- -i: performs a case-insensitive search.
- -n: displays the lines containing the pattern along with the line numbers.
- -v: displays the lines not containing the specified pattern.
- -c: displays the count of the matching patterns.

Examples:

- Match all lines that start with 'hello'. E.g: "hello there"

```
$ grep "^hello" file1
```

- Match all lines that end with 'done'. E.g: "well done"

```
$ grep "done$" file1
```

- Match all lines that contain any of the letters 'a', 'b', 'c', 'd' or 'e'.

```
$ grep "[a-e]" file1
```

- Match all lines that do not contain a vowel

```
$ grep "[^aeiou]" file1
```

- Match all lines that start with a digit following zero or more spaces. E.g: " 1." or " 2."

```
$ grep "[0-9]" file1
```

- Match all lines that contain the word hello in upper-case or lower-case

```
$ grep -i "hello"
```

Displaying the lines before the match

Some times, if you are searching for an error in a log file; it is always good to know the lines around the error lines to know the cause of the error.

```
grep -B 2 "Error" file.txt
```

This will prints the matched lines along with the two lines before the matched lines.

Displaying the lines after the match

```
grep -A 3 "Error" file.txt
```

This will display the matched lines along with the three lines after the matched lines.

Displaying the lines around the match

```
grep -C 5 "Error" file.txt
```

This will display the matched lines and also five lines before and after the matched lines.

Searching for a string in all files recursively


You can search for a string in all the files under the current directory and sub-directories with the help -r option.

```
grep -r "string" *
```

If you wish to search for a phrase that starts and end with some specific expression

you can do that using *. For instance, if we wanted to search for all instances where a phrase begins with free and ends with source we could use the following command.

```
grep -n 'free.*source' LinuxAndUbuntu.txt
```



```
rishabh@LinuxAndUbuntu: ~/Desktop
rishabh@LinuxAndUbuntu:~/Desktop$ grep -n 'free.*source' LinuxAndUbuntu.txt
2:free and open source software
rishabh@LinuxAndUbuntu:~/Desktop$
```

As you can see in the image above that we searched for a phrase that begins with free and ended with a source. We found a match that was 'free and open source'. So that was a quick and concise tutorial on how to use the grep command. The grep command is extremely powerful with many options.

Printing only matched string

You are also allowed to **only print the matched string on the standard output (instead of complete lines that get displayed by default)**. This feature can be accessed using the -o command line option.

```
$ grep -o [string-to-be-searched] [filename]
```

For example, if the requirement is to search a string "linux" in a file (but complete lines should not be printed), then we will use the following command.

```
$ grep -o "linux" [file-name]
```

```
HTF@howtoforge:~$ grep -o "linux" test_file1.txt
linux
HTF@howtoforge:~$
```

We can also use wildcard characters such as * and .* to grep more than one string. For example, If we want to grep a group of words starting from "how" and ending at "linux", then we can use the following command.

```
$ grep -o "how.*linux" [file-name]
```

```
HTF@howtoforge:~$ grep -o "how.*linux" test_file1.txt
howtoforge provides user-friendly linux
HTF@howtoforge:~$
```

Display position in Grep

The grep command also allows you to display the byte-offset of the line in which the matched string occurs. This feature can be accessed using the -b command line option. But for the better usage of this option, you can use it with the -o command line option, which will display the exact position of the matched string.

```
$ grep -o -b [string-to-be-searched] [filename]
```

For example:

```
$ grep -o -b "for" test_file1.txt
```

Here is the output:

```
HTF@howtoforge:~$ grep -o -b "for" test_file1.txt
16:for
43:for
218:for
HTF@howtoforge:~$
```

How to match sets of character using grep

The dot (.) matches any single character. You can match specific characters and character ranges using [...] syntax. Say you want to Match both ‘Vivek’ or ‘vivek’:

```
grep '[vV]ivek' filename
```

OR

```
grep '[vV][iI][Vv][Ee][kK]' filename
```

You can also match digits (i.e match vivek1 or Vivek2 etc):

```
grep -w '[vV]ivek[0-9]' filename
```

You can match two numeric digits (i.e. match foo11, foo12 etc):

```
grep 'foo[0-9][0-9]' filename
```

You are not limited to digits, you can match at least one letter:

```
grep '[A-Za-z]' filename
```

Display all the lines containing either a “w” or “n” character:

```
grep [wn] filename
```

Within a bracket expression, the name of a character class enclosed in “[:]” and “[:.]” stands for the list of all characters belonging to that class. Standard character class names are:

- **[[:alnum:]]** – Alphanumeric characters.
- **[[:alpha:]]** – Alphabetic characters
- **[[:blank:]]** – Blank characters: space and tab.
- **[[:digit:]]** – Digits: ‘0 1 2 3 4 5 6 7 8 9’.
- **[[:lower:]]** – Lower-case letters: ‘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’.
- **[[:space:]]** – Space characters: tab, newline, vertical tab, form feed, carriage return, and space.
- **[[:upper:]]** – Upper-case letters: ‘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’.



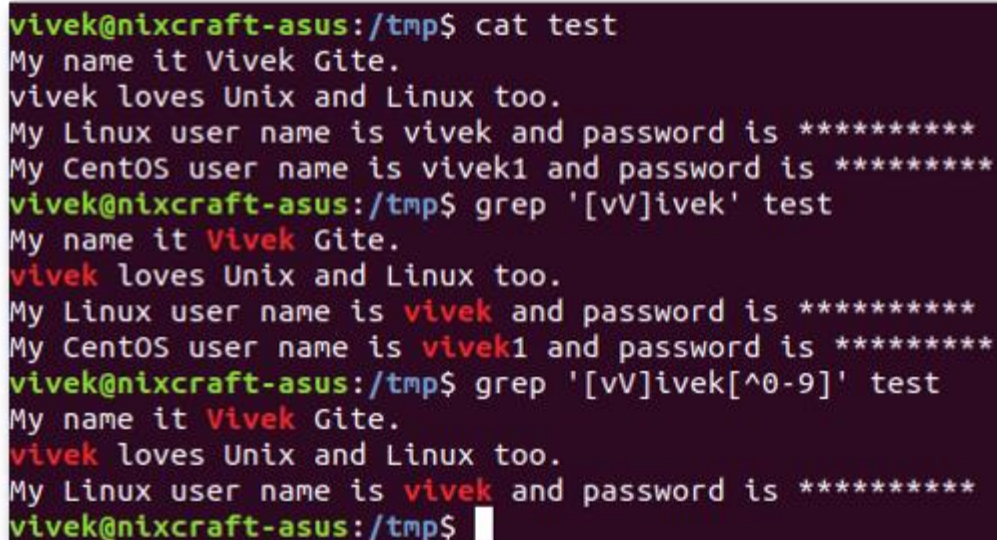
In this example match all upper case letters:

```
grep '[:upper:]' filename
```

How negates matching in sets

The ^ negates all ranges in a set:

```
grep '[vV]ivek[^0-9]' test
```



```
vivek@nixcraft-asus:/tmp$ cat test
My name it Vivek Gite.
vivek loves Unix and Linux too.
My Linux user name is vivek and password is *****
My CentOS user name is vivek1 and password is *****
vivek@nixcraft-asus:/tmp$ grep '[vV]ivek' test
My name it Vivek Gite.
vivek loves Unix and Linux too.
My Linux user name is vivek and password is *****
My CentOS user name is vivek1 and password is *****
vivek@nixcraft-asus:/tmp$ grep '[vV]ivek[^0-9]' test
My name it Vivek Gite.
vivek loves Unix and Linux too.
My Linux user name is vivek and password is *****
vivek@nixcraft-asus:/tmp$
```

© www.cyberciti.biz

Using grep regular expressions to search for text patterns

Wildcards

You can use the “.” for a single character match. In this example match all 3 character word starting with “b” and ending in “t”:

```
grep '\<b.t\>' filename
```

Where,



- \< Match the empty string at the beginning of word
- \> Match the empty string at the end of word.

Print all lines with exactly two characters:

```
grep '^..$' filename
```

Display any lines starting with a dot and digit:

```
grep '^\. [0-9]' filename
```

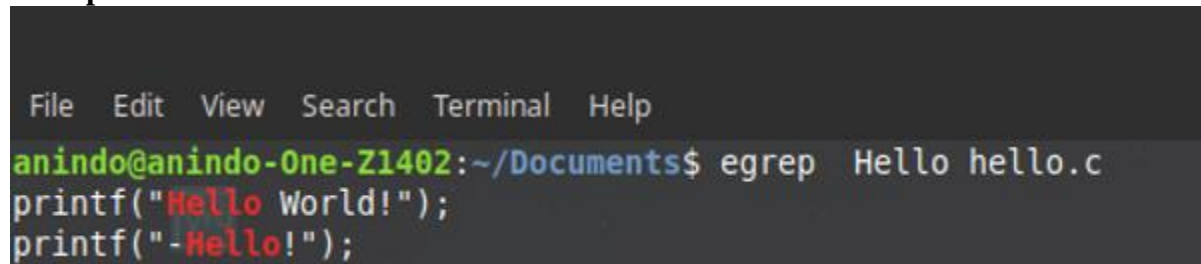
egrep command in Linux with examples

egrep is a pattern searching command which belongs to the family of **grep** functions. It works the same way as **grep -E** does. It treats the pattern as an extended regular expression and prints out the lines that match the pattern. If there are several files with the matching pattern, it also displays the file names for each line.

Syntax:

```
egrep [ options ] 'PATTERN' files
```

Example:

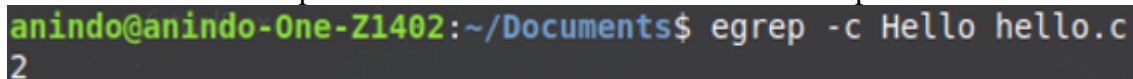


```
File Edit View Search Terminal Help
anindo@anindo-One-Z1402:~/Documents$ egrep Hello hello.c
printf("Hello World!");
printf("-Hello!");
```

Note: The *egrep* command is used mainly due to the fact that it is faster than the *grep* command. The *egrep* command treats the meta-characters as they are and do not require to be escaped as is the case with *grep*. This allows reducing the overhead of replacing these characters while pattern matching making *egrep* faster than *grep* or *fgrep*.

Options: Most of the options for this command are same as **grep**.

- **-c:** Used to counts and prints the number of lines that matched the pattern and not the lines.



```
anindo@anindo-One-Z1402:~/Documents$ egrep -c Hello hello.c
2
```

- **-v:** It prints the lines that does not match with the pattern.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -v Hello hello.c
#include "stdio.h"

int main(){
return 0;
}
```

- **-i:** Ignore the case of the pattern while matching.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -i hello hello.c
printf("Hello World!");
printf("-Hello!");
```

- **-l:** Prints only the names of the files that matched. It does not mention the matching line numbers or any other information.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -l Hello hello.c
hello.c
```

- **-L:** Prints only the names of the files that did not have the pattern. Opposite of **-l** flag.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -L Hello hello.c myfile.txt
myfile.txt
```

- **-e:** Allows to use a '-' sign in the beginning of the pattern. If not mentioned the shell tries to execute the pattern as an option and returns an error.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -e -Hello hello.c
printf("-Hello!");
```

- **-w:** Prints only those lines that contain the whole words. Word-constituent characters are letters, digits and underscore. The matching substring must be separated by non-word constituent characters.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -w Hello hello.c
printf("Hello World!");
printf("-Hello!");
```

- **-x:** Prints only those lines that matches an entire line of the file.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -x 'return 0;' hello.c
return 0;
```

- **-m NUMBER:** Continue to search for matches till the count reaches NUMBER mentioned as argument.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -m 1 Hello hello.c
printf("Hello World!");
```

- **-o:** Prints only the matched parts of the line and not the entire line for each match.



```
anindo@anindo-One-Z1402:~/Documents$ egrep -o Hello hello.c
Hello
Hello
```

- **-n:** Prints each matched line along with the respective line numbers. For multiple files, prints the file names along with line numbers.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -n Hello hello.c
4:printf("Hello World!");
5:printf("-Hello!");
```

- **-r:** Recursively search for the pattern in all the files of the directory. The last argument is the directory to check. '.' (dot) represents the current directory.

```
anindo@anindo-One-Z1402:~/Documents$ egrep -r -i 'h*' .
./myfile.txt:hi
./hello.c:#include "stdio.h"
./hello.c:
./hello.c:int main(){
./hello.c:printf("Hello World!");
./hello.c:printf("-Hello!");
./hello.c:return 0;
./hello.c:}
```

How Do I Test Sequence?

You can test how often a character must be repeated in sequence using the following syntax:

```
{N}
{N,}
{min,max}
```

Match a character “v” two times:

```
egrep "v{2}" filename
```

The following will match both “col” and “cool”:

```
egrep 'co{1,2}l' filename
```

The following will match any row of at least three letters ‘c’.

```
egrep 'c{3,}' filename
```

grep regex operator	Meaning	Example
.	Matches any single character.	grep '.' file grep 'foo.' input
?	The preceding item is optional and will be matched, at most, once.	grep 'vivek?' /etc/passwd
*	The preceding item will be matched zero or more times.	grep 'vivek*' /etc/passwd
+	The preceding item will be matched one or more times.	ls /var/log/ grep -E "^[a-z]+\log."
{N}	The preceding item is matched exactly N times.	egrep '[0-9]{2}' input
{N,}	The preceding item is matched N or more times.	egrep '[0-9]{2,}' input
{N,M}	The preceding item is matched at least N times, but not more than M times.	egrep '[0-9]{2,4}' input
—	Represents the range if it's not first or last in a list or the ending point of a range in a list.	grep ':/bin/[a-z]*' /etc/passwd

<code>^</code>	Matches the empty string at the beginning of a line; also represents the characters not in the range of a list.	<code>grep '^vivek' /etc/passwd</code> <code>grep '[^0-9]*' /etc/passwd</code>
<code>\$</code>	Matches the empty string at the end of a line.	<code>grep '\$' /etc/passwd</code>
<code>\b</code>	Matches the empty string at the edge of a word.	<code>vivek \bvivek' /etc/passwd</code>
<code>\B</code>	Matches the empty string provided it's not at the edge of a word.	<code>grep '\B/bin/bash /etc/passwd</code>
<code><</code>	Match the empty string at the beginning of word.	<code>grep "<="" kbd="" style=""box-sizing: inherit;"></code>
<code>></code>	Match the empty string at the end of word.	<code>grep 'bash\>' /etc/passwd</code> <code>grep '\>' /etc/passwd</code>

grep vs egrep

egrep is the same as grep -E. It interprets PATTERN as an extended regular expression.

Examples:

We'll start with something simple. Let's say we wish to **identify any line with two or more vowels in a row**. In the example below the multiplier {2,} applies to the preceding item which is the range.

```
egrep '[aeiou]{2,}' mysampled.txt
Robert pears 4
Lisa peaches 7
```

Anne mangoes 7

Greg pineapples 3

How about any line with a 2 on it which is not the end of the line. In this example the multiplier + applies to the . which is any character.

```
egrep '2.+' mysampled.txt  
Fred apples 20
```

We want to check that the character 'p' appears exactly 2 times in a string one after the other. For this the syntax would be:

```
cat sample | grep -E p\{2}
```

```
guru99@guru99-VirtualBox:~$ cat sample|grep -E p\{2}  
apple  
guru99@guru99-VirtualBox:~$
```

Note: You need to add -E with these regular expressions.

Searching for all characters 't'

```
guru99@guru99-VirtualBox:~$ cat sample|grep t  
bat  
ant  
eat  
pant  
taste
```

Suppose we want to filter out lines where character 'a' precedes character 't'

We can use command like

```
cat sample|grep "a\+t"
```

```
guru99@guru99-VirtualBox:~$ cat sample|grep "a\+t"  
bat  
eat  
guru99@guru99-VirtualBox:~$
```

fgrep command in Linux with examples



The *fgrep* filter is used to search for the fixed-character strings in a file. There can be multiple files also to be searched. This command is useful when you need to search for strings which contain lots of regular expression metacharacters, such as “^”, “\$”, etc.

Syntax:

```
fgrep [options] [ -e pattern_list] [pattern] [file]
```

Options with Description:

- **-c** : It is used to print only a count of the lines which contain the pattern.
- **-h** : Used to display the matched lines.
- **-i** : During comparisons, it will ignore upper/lower case distinction.
- **-l** : Used to print the names of files with matching lines once, separated by new-lines. It will not repeat the names of files when the pattern is found more than once.
- **-n** : It is used precede each line by its line number in the file (first line is 1).
- **-s** : It will only display the error messages.
- **-v** : Print all lines except those contain the pattern.
- **-x** : Print only lines matched entirely.
- **-e pattern_list** : Search for a string in pattern-list (useful when the string begins with a “-”).
- **-f pattern-file** : Take the list of patterns from pattern-file.
- **pattern** : Specify a pattern to be used during the search for input.
- **file** : A path name of a file to be searched for the patterns. If no file operands are specified, the standard input will be used.

Below are the examples with options to illustrate the fgrep command:

Consider below file as input. Here it is create using cat command and “*name of the file is para*”.

```
Hi, @re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.  
Geeks*forgeeks is best for learni\ng.
```

-c option: Displaying the count of number of matches. We can find the number of lines that match the given string.

Example:

```
$fgrep -c "usin.g" para
```

Output:

```
1
```

```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para  
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.  
Geeks*forgeeks is best for learni\ng.  
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -c "usin.g" para  
1  
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

-h option: To display the matched lines.



Example:

```
fgrep -h "usin.g" para
```

Output:

Hi, @re you **usin.g** geeks*forgeeks for learni\ng computer science con/cepts.

```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -h "usin.g" para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

-i option: Used in case insensitive search. It ignore upper/lower case distinction during comparisons. It matches words like : “geeks*forgeeks”, “Geeks*forgeeks”.

Example:

```
fgrep -i "geeks*forgeeks" para
```

Output:

Hi, @re you usin.g **geeks*forgeeks** for learni\ng computer science con/cepts.

Geeks*forgeeks is best for learni\ng.

```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -i "geeks*forgeeks" para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

-l option: It will display the file names that match the pattern. We can just display the files that contains the given string/pattern.

Example:

```
fgrep -l "geeks*forgeeks" para para2
```

Output:

para

```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -l "geeks*forgeeks" para
para
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

-n option: Precede each line by its line number in the file. It shows line number of file with the line matched.

Example:

```
$ fgrep -n "learni\ng" para
```


Output:

```
1:Hi, @re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
```

```
2:Geeks*forgeeks is best for learni\ng.
```

```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -n "learni\ng" para
1:Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
2:Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

-v option: It is used to display all lines except those that contain the pattern. It will print all lines except those that contain the pattern.

Example:

```
fgrep -v "@re" para
```

Output:

```
Geeks*forgeeks is best for learni\ng.
```

```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para
Hi,@re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -v "@re" para
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

-x option: It will display only lines matched entirely.

Example 1:

```
fgrep -x "@re" para
```

Output:**Example 2:**

```
fgrep -x "Geeks*forgeeks is best for learni\ng." para
```

Output:

```
Geeks*forgeeks is best for learni\ng.
```



```
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ cat para
Hi, @re you usin.g geeks*forgeeks for learni\ng computer science con/cepts.
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -x "@re" para
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$ fgrep -x "Geeks*forgeeks is best for learni\ng." para
Geeks*forgeeks is best for learni\ng.
paras@paras:~/Desktop/coding/my-project/GFG articles/frep$
```

Sed Command in Linux/Unix with examples

SED command in UNIX is stands for stream editor and it can perform lot's of function on file like, searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace. By using SED you can edit files even without opening it, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.

- SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
- SED command in unix supports regular expression which allows it perform complex pattern matching.

Syntax:

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

Example:

Consider the below text file as an input.

```
$cat > geekfile.txt
```

```
unix is great os. unix is opensource. unix is free os.
learn operating system.
unix linux which one you choose.
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

Sample Commands

1. **Replacing or substituting string :** Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word “unix” with “linux” in the file.

```
$sed 's/unix/linux/' geekfile.txt
```

Output :

```
linux is great os. unix is opensource. unix is free os.
learn operating system.
linux linux which one you choose.
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

Here the “s” specifies the substitution operation. The “/” are delimiters. The “unix” is the search pattern and the “linux” is the replacement string.

By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.

2. **Replacing the nth occurrence of a pattern in a line** : Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word “unix” with “linux” in a line.

```
sed 's/unix/linux/2' geekfile.txt
```

Output:

```
unix is great os. linux is opensource. unix is free os.  
learn operating system.  
unix linux which one you choose.  
unix is easy to learn.linux is a multiuser os.Learn unix .unix is a powerful.
```

3. **Replacing all the occurrence of the pattern in a line:** The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

```
$sed 's/unix/linux/g' geekfile.txt
```

Output :

```
linux is great os. linux is opensource. linux is free os.  
learn operating system.  
linux linux which one you choose.  
linux is easy to learn.linux is a multiuser os.Learn linux .linux is a powerful.
```

4. **Replacing from nth occurrence to all occurrences in a line:** Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth... “unix” word with “linux” word in a line.

```
$sed 's/unix/linux/3g' geekfile.txt
```

Output:

```
unix is great os. unix is opensource. linux is free os.  
learn operating system.  
unix linux which one you choose.  
unix is easy to learn.unix is a multiuser os.Learn linux .linux is a powerful.
```

5. **Replacing string on a specific line number:** You can restrict the sed command to replace the string on a specific line number. An example is

```
$sed '3 s/unix/linux/' geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.  
learn operating system.  
linux linux which one you choose.  
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

The above sed command replaces the string only on the third line.

6. Replacing string on a range of lines: You can specify a range of line numbers to the sed command for replacing a string.

```
$sed '1,3 s/unix/linux/' geekfile.txt
```

Output:

```
linux is great os. unix is opensource. unix is free os.  
learn operating system.  
linux linux which one you choose.  
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

Here the sed command replaces the lines with range from 1 to 3. Another example is

```
$sed '2,$ s/unix/linux/' geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.  
learn operating system.  
linux linux which one you choose.  
linux is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful
```

Here \$ indicates the last line in the file. So the sed command replaces the text from second line to last line in the file.

7. Deleting lines from a particular file: SED command can also be used for deleting lines from a particular file. SED command is used for performing deletion operation without even opening the file

Examples:

1. To Delete a particular line say n in this example

Syntax:

```
$ sed 'nd' filename.txt
```

Example:

```
$ sed '5d' filename.txt
```

2. To delete a last line

Syntax:

```
$ sed '$d' filename.txt
```

3. To Delete line from range x to y

Syntax:

```
$ sed 'x,yd' filename.txt
```

Example:



```
$ sed '3,6d' filename.txt
```

5. To Delete from nth to last line

Syntax:

```
$ sed 'nth,$d' filename.txt
```

Example:

```
$ sed '12,$d' filename.txt
```

6. To Delete pattern matching line

Syntax:

```
$ sed '/pattern/d' filename.txt
```

Example:

```
$ sed '/abc/d' filename.txt
```

Delete Lines

To delete lines, the delete (d) flag is your friend.

The delete flag deletes the text from the stream, not the original file.

```
$ sed '2d' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
First line
Second line
Third line
Fourth line
likegeeks@likegeeks-VirtualBox ~/Desktop $ sed '2d' myfile
First line
Third line
Fourth line
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
First line
Second line
Third line
Fourth line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here we delete the second line only from myfile.

What about deleting a range of lines?

```
$ sed '2,3d' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ sed '2,3d' myfile
This is a test.
This is the fourth test.
likegeeks@likegeeks-VirtualBox ~/Desktop $
```



Here we delete a range of lines, the second and the third.

Another type of ranges:

```
$ sed '3,$d' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ sed '3,$d' myfile
This is a test.
This is the second test.
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here we delete from the third line to the end of the file.

All these examples never modify your original file.

```
$ sed '/test 1/d' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
This is a test.
This is the second test.
This is the thrid test.
This is the fourth test.
This is another line
likegeeks@likegeeks-VirtualBox ~/Desktop $ sed '/test 1/d' myfile
This is another line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here we use a pattern to delete the line if matched on the first line.

If you need to delete a range of lines, you can use two text patterns like this:

```
$ sed '/second/,/fourth/d' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
This is a test.
This is the second test.
This is the thrid test.
This is the fourth test.
This is another line
likegeeks@likegeeks-VirtualBox ~/Desktop $ sed '/second/,/fourth/d' myfile
This is a test.
This is another line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

From the second to the fourth line are deleted.

To run multiple sed commands, you can use the -e option like this:

```
$ sed -e 's/This/That/; s/test/another test/' ./myfile
```



The screenshot shows a text editor window titled 'myfile (~/Desktop)' containing the text 'This is a test'. Below it, a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' shows the command `sed -e 's/This/That/; s/test/another test/' myfile` being executed. The output in the terminal is 'That is a another test'.

Sed command must be separated by a semicolon without any spaces.

Also, you can use a single quotation to separate commands like this:

```
$ sed -e '  
> s/This/That/  
> s/test/another test/' myfile
```

The screenshot shows the same text editor window with 'This is a test'. The terminal window shows the command `sed -e ' > s/This/That/ > s/test/another test/' ./myfile` being executed. The output is 'That is a another test'.

The same result, no big deal.

Reading Commands From a File

You can save your sed commands in a file and use them by specifying the file using -f option.

```
$ cat mycommands  
s/This/That/  
s/test/another test/
```

```
$ sed -f mycommands myfile
```



Context addressing in Sed

The p flag prints each line contains a pattern match, you can use the -n option to print the modified lines only.

```
$ cat myfile
$ sed -n 's/test/another test/p' myfile
```

Writing selected lines to a file

The w flag saves the output to a specified file:

```
$ sed 's/test/another test/w output' myfile
```




```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ sed 's/test/another test/w output' myfile
This is a another test.
This is a different one.
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat output
This is a another test.
```

The output is printed on the screen, but the matching lines are saved to the output file.

Add character at the beginning of each line using sed command.

For example **to add # in front of each line**

we can use sed command with following syntax:

```
$ sed 's/^/#/' file.txt
#add
#character
#at the
#beginning of
#each line
```

replace # with ' ' (space) to add space in front of each line:

```
$ sed 's/^/ /' file.txt
add
character
at the
beginning of
```

Redirect the output produced by sed command to save it to a file:

```
$ sed 's/^/ /' file.txt > new-file.txt
$ cat new-file.txt
add
character
at the
beginning of
each line
```

Assignment

1. Difference between grep, egrep, fgrep?