

**Index**

<b>SR NO.</b>	<b>NAME</b>	<b>PAGE NO.</b>
<b>4.1</b>	<b>Introduction to Regular Expressions (Basic and Extended)</b>	<b>2</b>
<b>4.2</b>	<b>Pattern Matching using grep, egrep, and fgrep</b>	<b>12</b>
<b>4.3</b>	<b>Stream Editing with sed (search, replace, line deletion, insertion)</b>	<b>27</b>



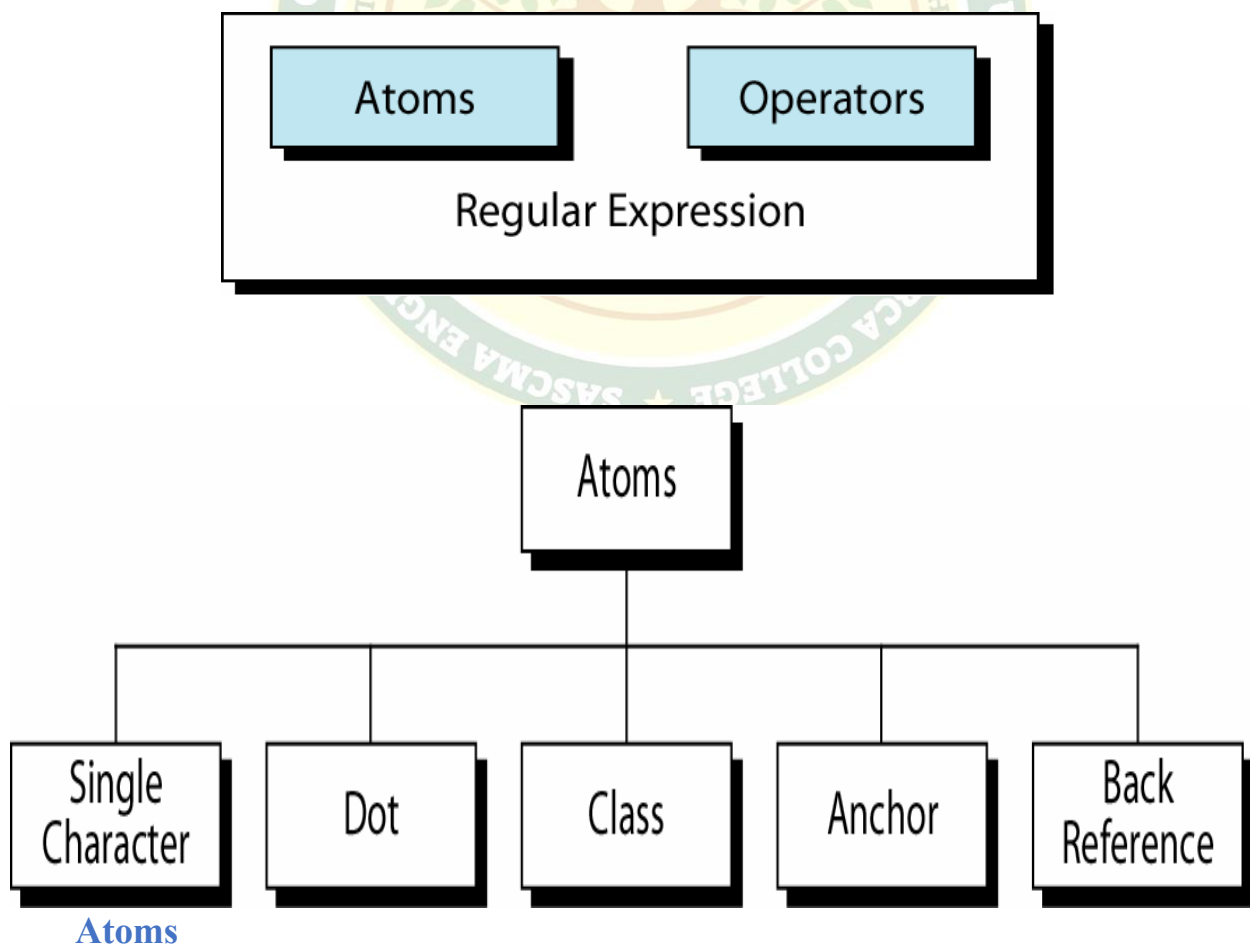
## 4.1 Introduction to Regular Expressions (Basic and Extended)

A regular expression is a pattern consisting of a sequence of characters that matched against the text. UNIX evaluates text against the pattern to determine if the text and the pattern match. If they match, the expression is true and a command is executed. Some of the most powerful UNIX utilities , such as grep and sed, use regular expressions.

Some of the most powerful UNIX utilities , such as grep and sed, use regular expressions.

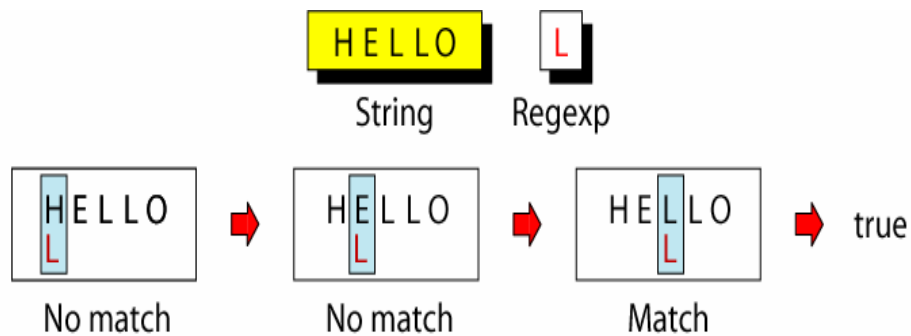
### Regular Expression

A regular expression is like a mathematical expression. A mathematical expression is made of operands (data) and operators. A regular expression is made of atoms and operators. The atom specifies what we are looking for and where in the text the match is to be made. The operator combines atoms into complex expressions.

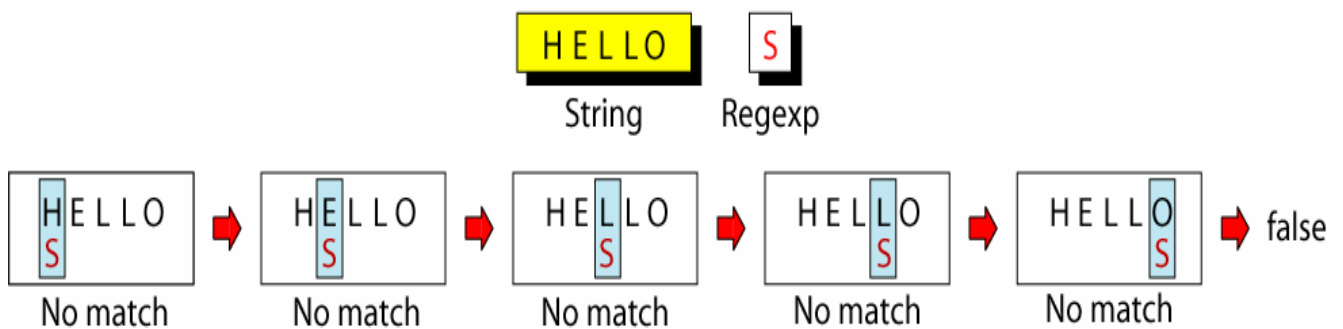


### Single-Character Pattern Example

The simplest atom is a single character.



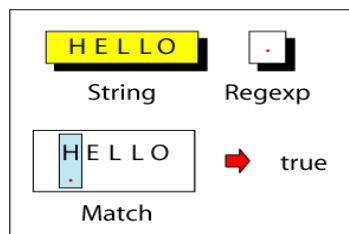
(a) Successful Pattern Match



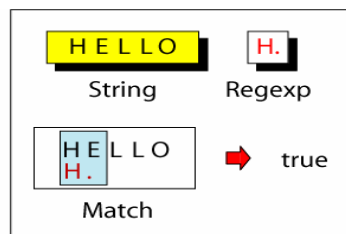
(b) Unsuccessful Pattern Match

### Dot Atom Example

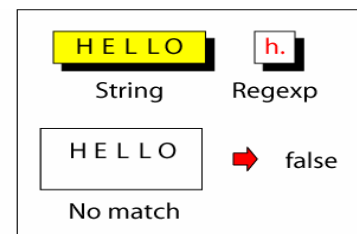
A dot matches any single character except the new line character(\n).



(a) Single-Character



(b) Combination-True



(c) Combination-False

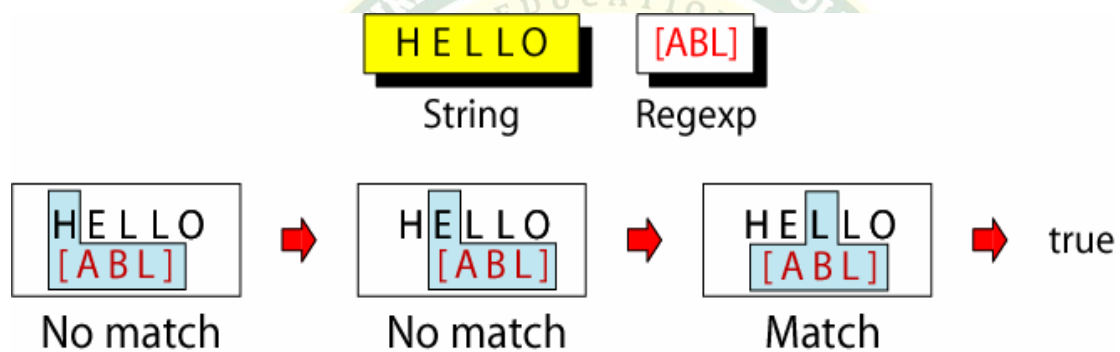
### Class Atom Example

The class atom defines a set of ASCII characters, any one of which may match any of the characters in the text.

The character set to be used in the matching process is enclosed in brackets.

A range of text characters is indicated by a dash (-). [a-d]

^ is an exclusion operator. To specify any character other than a vowel, we use [^aeiou].



### Example of Classes

The escape character (\) is used when the matching character is one of the other two tokens: - and ^.

RegExpr		Means	RegExpr		Means
[A-H]	→	[ABCDEFGH]	[^AB]	→	Any character except A or B
[A-Z]	→	Any uppercase alphabetic	[A-Za-z]	→	Any alphabetic
[0-9]	→	Any digit	[^0-9]	→	Any character except a digit
[ a ]	→	[ or a	[ a ]	→	] or a
[0-9\ -]	→	digit or hyphen	[^\^]	→	Anything except ^

## Anchors

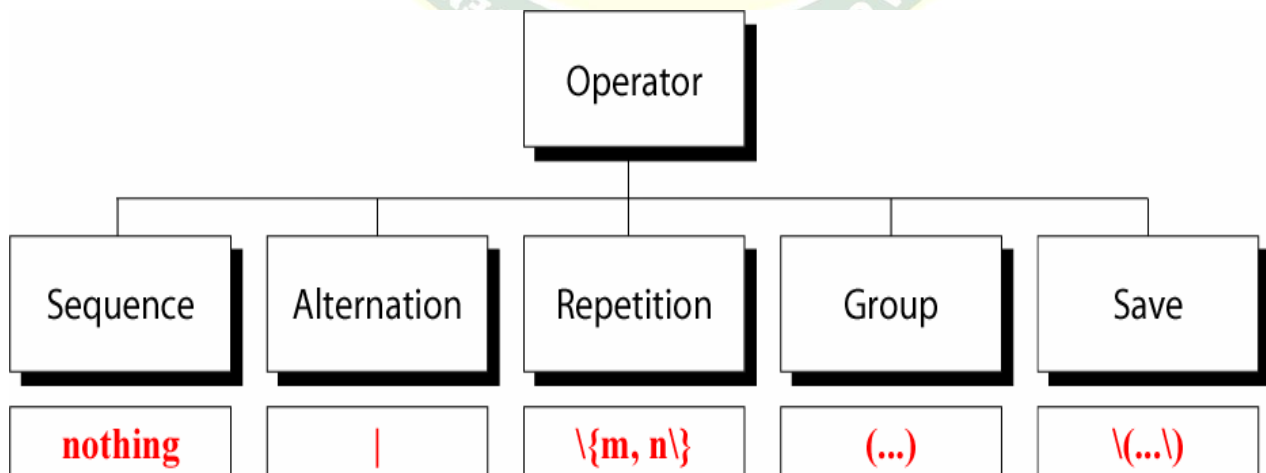
Anchors are atoms that are used to line up the pattern with a particular part of a string.

Anchors are not matched to the text, but define where the next character in the pattern must be seen.

Anchor		Means	Example
		Beginning of line	
		End of line	
		Beginning of word	
		End of word	

## Operators

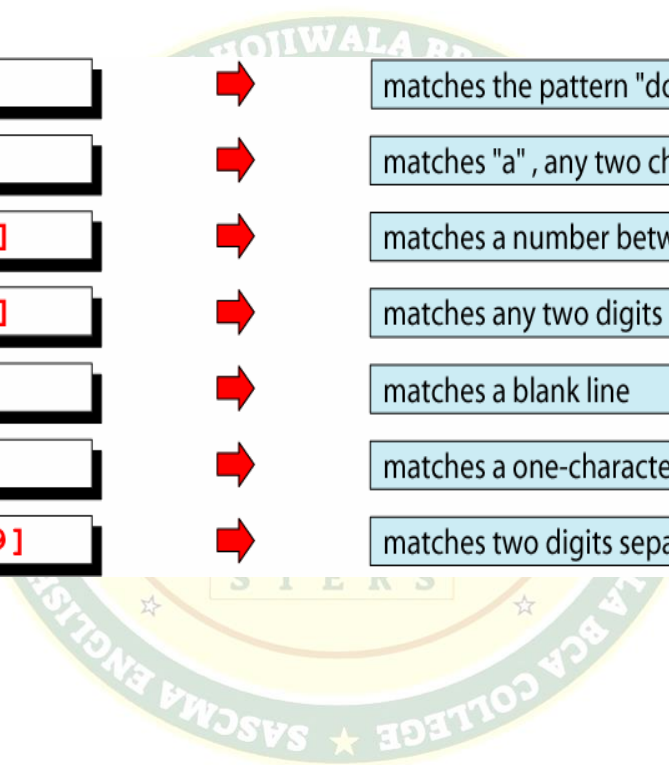
We can combine atoms with operators.



### Example of Sequence Operator

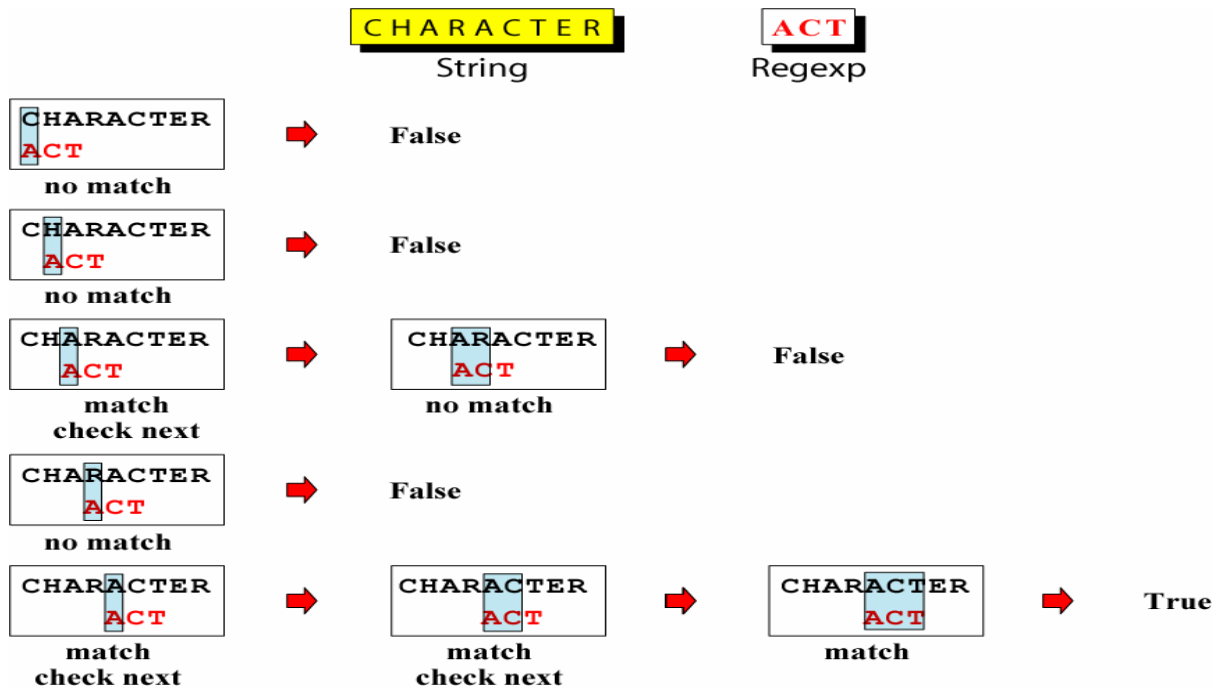
The sequence operator is nothing.

This means that if a series of atoms are shown in a regular expression, it is implied that there is an invisible sequence operator between them.



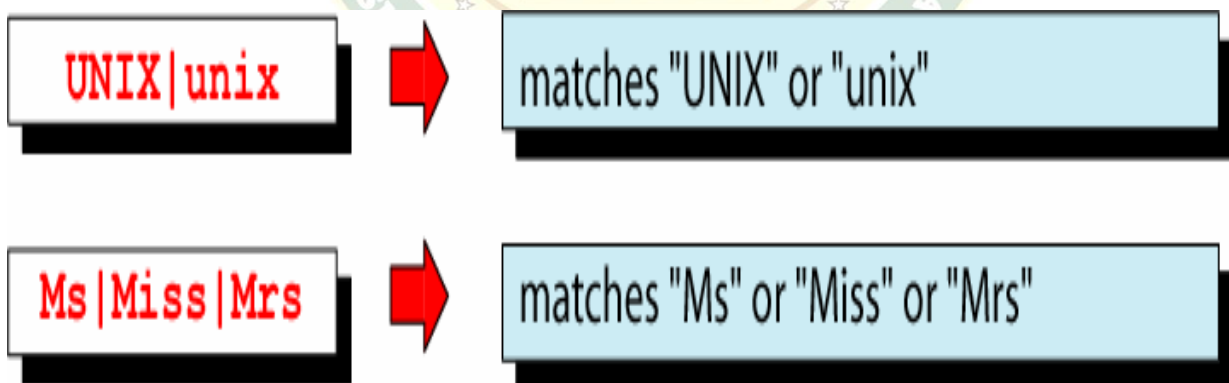
<code>dog</code>	→	matches the pattern "dog"
<code>a..b</code>	→	matches "a" , any two characters, and "b"
<code>[2-4][0-9]</code>	→	matches a number between 20 and 49
<code>[0-9][0-9]</code>	→	matches any two digits
<code>^\$</code>	→	matches a blank line
<code>^.\$</code>	→	matches a one-character line
<code>[0-9]-[0-9]</code>	→	matches two digits separated by a "-"

### Evaluation of a String Using Sequence Operator

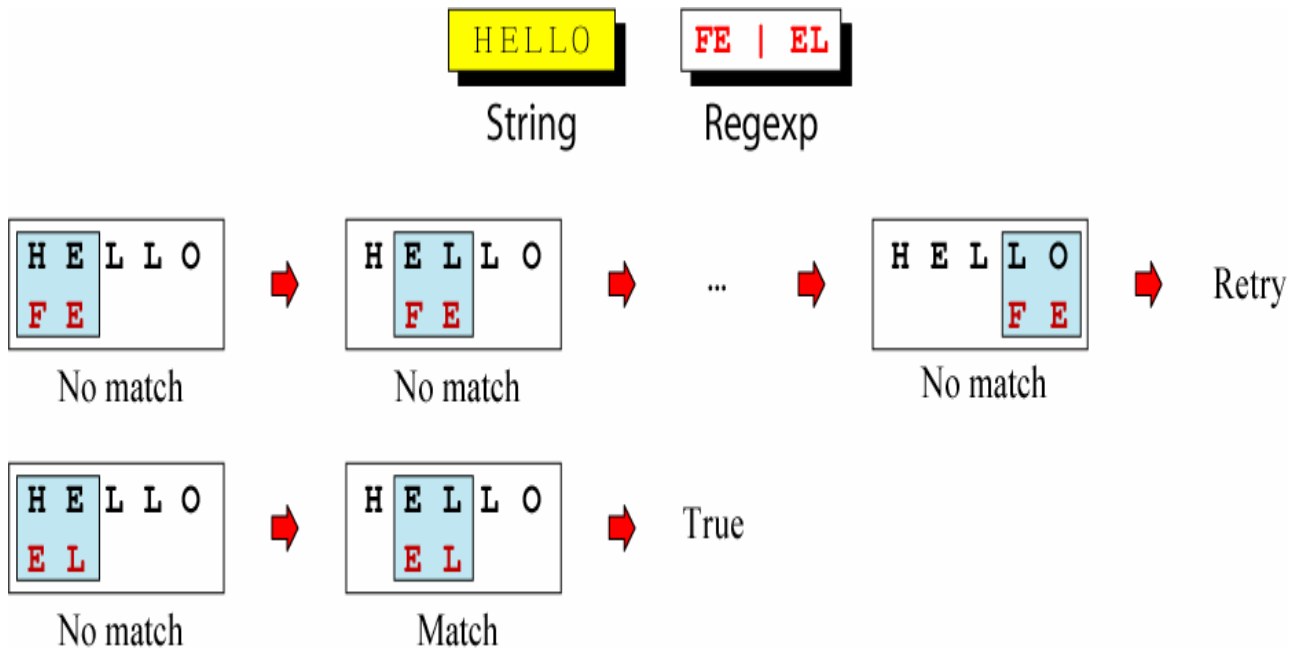


### Alternation Operator

The alternation operator is used to define one or more alternatives.



## Matching Alternation Operators



## Repetition Operator

The repetition operator specifies that the atom or expression immediately before the repetition may be repeated.

m is a minimum number of repetitions.

n is a maximum number of repetitions.

$$\{m, n\}$$

matches previous character m to n times.

$$A \setminus \{3, 5\}$$

matches "AAA", "AAAA", or "AAAAA"

$$BA\{3, 5\}$$

matches "BAAA", "BAAAA", or "BAAAAA"



## Basic Repetition Forms

### Formats

`\{m\}`

matches previous atom exactly m times

`\{m, \}`

matches previous atom m times or more

`\{, n\}`

matches previous atom n times or less

### Examples

`CA\{5\}`

CAAAAA

`CA\{3, \}`

CAAA, CAAAA, CAAAAA, ...

`CA\{, 2\}`

C, CA, CAA



### Example of Short Form Repetition Operators

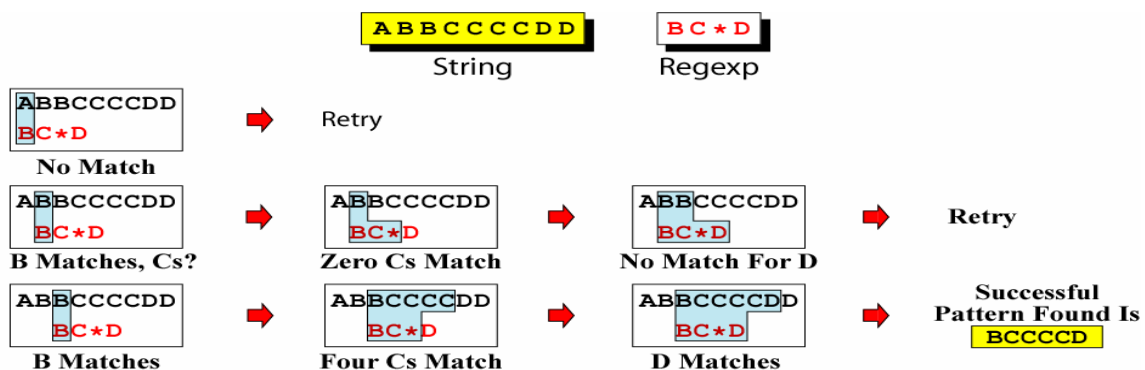
#### Formats

*	→	special case: matches previous atom zero or more times
+	→	special case: matches previous atom one or more times
?	→	special case: matches previous atom 0 or one time only

#### Examples

BA*	→	B, BA, BAA, BAAA, BAAAA, ...
B.*	→	B, BA ... BZ, BAA ... BZZ, BAAA ... BZZZ, ...
.*	→	zero or more characters
.+	→	one or more characters
[0-9]?	→	zero or one digit

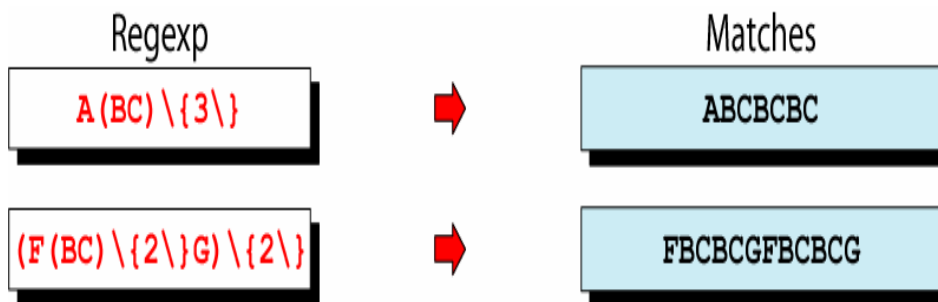
### Repeating Pattern Matching



## Group Operator

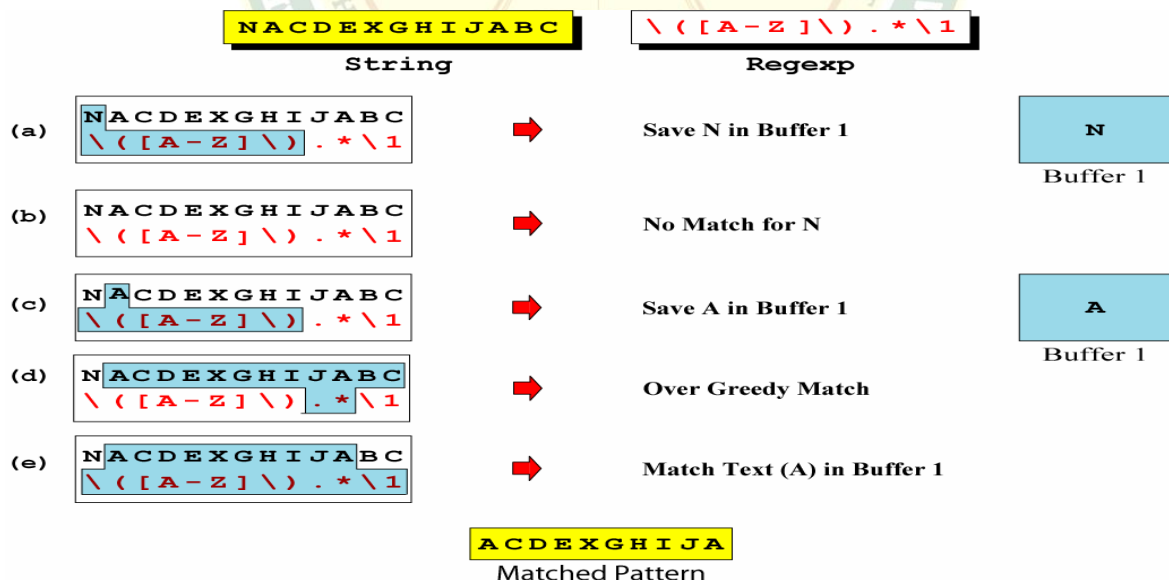
The group operator is a pair of opening and closing parentheses.

When a group of characters is enclosed in parentheses, the next operator applies to the whole group.



## Saving

The save operator `\( )` copies a matched text string to one of nine buffers for later reference.



## 4.2 Pattern Matching using grep, egrep, and fgrep

### Introduction

Till now, we discussed different filter utilities. In this chapter, we will learn about a very powerful filter utility known as grep. The grep stands for globally search a regular expression and print it. It is also known as pattern matching utility. It is used to search a file for a particular pattern of characters, and display all records/lines that contain a pattern. The pattern that is searched in the file is referred to as the regular expression.

### Pattern matching utility: grep

It is a filter utility that performs various tasks as follow:

- ✓ It scans a file for the occurrences of a pattern and displays lines in which scanned pattern is found.
- ✓ It scans a file for the occurrences of a pattern and displays lines in which scanned pattern does not found.
- ✓ It scans files for the occurrences of a pattern and displays name of files which contains a pattern in them.
- ✓ It also displays count of lines which contains pattern.

The general syntax for the **grep** command is as follows:

#### syntax:

```
grep [options] pattern [filename(s)]
```

It is used to select and extract lines from a file and print only those lines that match a given pattern. In the above syntax square bracket indicates optional part. The filename(s) and options are optional and pattern is compulsory in the **grep** command. Here, a pattern is a simple string or more complex which contains metacharacters, a special character for pattern matching. A pattern is also known as regular expression.

Without a filename grep expects standard input. As a line is input, **grep** searches for the regular expression in the line and displays the line if it contains that regular expression. Execution stops when the user indicates end of input by pressing <ctrl+ d>.

For example, a user supply a command at shell prompt as follow:

```
$ grep 'unix'
```

```
unix and shell programming < enter>
```

```
unix and shell programming red hat linux<enter>
```

```
unix OS< enter>
```

```
unix OS
```

```
<ctrl+ d>
```

```
$
```

**grep** requires an expression to represent the pattern to be searched for, followed by one or more filenames.

The first argument is always treated as the expression, and the other arguments are considered as filenames.

### Specifying regular expression:

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.

An expression formed with some special and ordinary characters, which is expanded by a command, and not by the shell to match more than one string. A regular expression is always quoted to prevent its interpretation by the shell. Regular expressions can be used to specify very simple patterns of characters to highly complex ones. Some very simple patterns are shown in table-(a.12):

**Table-(a.12): Example of simple regular expression**

Regular expression	Meaning
A	It display all lines that contain character “A”.
“Unix”	It display all lines that contain pattern “Unix”

Consider the following examples:

- ✓ Let us assume that the input file f1 as follow:

```
$ cat f1
```

sco Unix

The red hat linux user name user 1

\$

If a user wish to display lines of file f1 which contains pattern 'Unix' then the command is as follow

\$grep Unix f1

sco Unix

\$

- ✓ If you want to locate lines of file f1 which contains character" then the command is as follow:

\$ grep x f1

sco Unix

The red hat linux

\$

More complex regular expressions can be specified by the grep's metacharacters, always written in the quotes, shown in table-(b.12).

**Table-(b.12): grep metacharacters**

Character	use
[...] or [...]	It matches any one single character within a square bracket.
^pattern	It matches a pattern at the beginning of each line.
Pattern\$	It matches a pattern at the end of each line.
.(dot)	It matches any single character except new-line character.
\(backslash)	It indicates that grep should ignore the special meaning of the character following it in regular expression
\<pattern	It matches a pattern at the beginning of any word in a line.
Pattern\>	It matches a pattern at the end of any word in a line.
ch*	It matches zero or more occurrence of character ch.
ch\{m\}	The preceding character ch is occurred m-times.
ch\{m,\}	The preceding character ch is occurred at least m times.
ch\{m,n\}	The preceding character ch is occurred between m and n times.



<code>\(exp\)</code>	It matches expression exp for later referencing with \1,\2...
----------------------	---

Consider the following examples which use grep metacharacters:

- ✓ To display lines of file f1 which contains pattern as user/ or user2 or user3 then the command is as follow:

```
$grep user [123]f1
```

```
user name user 1
```

```
$
```

It displays lines of file f1 which contains pattern user 1.

- ✓ You can display lines which begins with pattern The then the command is as follow:

```
$grep "The" f1
```

```
The red hat linux
```

```
$
```

It displays lines of file f1 which start with pattern The.

- ✓ Similarly, if you wish to match a pattern at the end of each line then the command is as follow:

```
$grep 'Unix$' f1
```

```
sco Unix
```

```
$
```

It displays lines of file f1 which end with pattern Unix.

- ✓ You can use dot. to match any character in a line. For example, consider a file f2 as follow:

```
$cat f2
```

```
Unix and shell programming
```

```
#blank line contains only new-line character
```

```
red hat linux
```

```
Unix OS
```

```
vb.net
```

program and process

\$

Here, file f2 contains blank and non-blank line. If you wish to remove blank line from the output then the command is:

```
$grep '.' f2
```

Unix and shell programming red

hat linux

Unix OS

vb.net

program and process

\$

It displays all lines which contains any character in a line except blank-line (contains only new line character).

- ✓ You can protect special meaning of grep metacharacter using back-slash. For example, a user wish to display lines which contains '!' character anywhere in a line then the command is as follow:

```
$ grep '\.' f2
```

vb.net

\$

It displays lines which contains '.' in a line.

- ✓ To display lines of file /2 which contains pattern 'program' then the command is as follow:

```
$ grep 'program' f2
```

Unix and shell programming

program and process

\$

But, if you want to display lines of file f2 which contains word 'program' that means it is not a part of any string then the command is as follow:

```
$grep '\<program\>' f2
```

program and process



\$

- ✓ The \* (asterisk) refers to the immediately preceding character. It matches zero or more occurrences of previous character. The pattern a\* matches a null string, single character a\* and any number of as.

i.e. (nothing) a aa aaa aaaa .....

- ✓ A user can locate lines which contains characters repeated more than one times then the command is:

```
$grep 'mm*' f2
```

Unix and shell programming  
program and process

\$

It locates lines in which character 'm' repeated one or more times.

You can display lines of file f1 which contains exact 8 characters then the command is as follow:

```
$grep '^.\{8\}$' f1
```

sco Unix

\$

To display lines of input file which contains characters between 5 and 15 then the command is like this:

```
$grep '^.\{5,15\}$' f2
```

red hat linux

Unix Os

vb.net

It displays lines of file f2 that contains character between 5 and 15.

- ✓ To display lines which contains pattern at the beginning of line would occur in the same line anywhere then you can use save operator with back references as follow:

```
$grep '^(.). *1'f2
```

program and process

It displays lines of file f2 which contains any character occur at the beginning of line would also occur anywhere in the same line. The output shows that 1<sup>st</sup> character 'p'

occur in the same line therefore we get such output.

- ✓ **grep** is silent and simply returns the prompt when a pattern is not found in a file.

```
$grep hello f2                                #No hello found
```

It displays nothing that means hello pattern do not present in file f2.

- ✓ **grep** also accept output of other command. For example, a user want to display filenames of working directory having permission read and write to owner, group and other user then the command is like this:

```
$ ls-l  grep '^rw-rw-rw-'  
-rw-rw-rw- 1 bharat  bharat    43      Apr 3 17:25 fl  
-rw-rw-rw- 2 bcal    tybcasems 77      Apr 4 11:02 fl.In  
-rw-rw-rw- 2 bcal    tybcasem5 77      Apr 4 11:02 12  
-rw-rw-rw- 1 bhrat   bhrat     34     Jul 18 2013 f3
```

- ✓ When **grep** is used with a series of strings, it interprets the first argument as the pattern and the rest as filenames along with the output. For example, consider a command as follow:

```
$grep red hat linux
```

It indicates that argument red is considered as pattern and other arguments hat and linux are considered as filenames.

- ✓ Quote is compulsory when a pattern contains more than one word. For example, consider the following command:  
grep "hello world" filename
- ✓ Quote is also compulsory when a pattern contains special characters that can be interpreted by search utility i.e. **grep**) not by the shell. You can generally use either single or double quotes, but if command substitution or variable evaluation is involved, you must use double quotes.

Consider an example which contains variable substitution in double-quote as follow:

```
$a=1  
$grep "$a" fl
```

It prints all lines of file f1 that contains 1. Consider another example which uses command substitution in double-quote as follow:

```
$grep "echo if" f1
```

It prints all lines of file f1 which contains pattern if in line.

### Options :

The **grep** utility can be used with many options, a few of which are discussed below:

**(1)-c (count):** It prints count of matching lines for each input file.

✓ For example, a command is follow:

```
$grep -c '.' f2
```

```
5
```

```
$
```

It counts all non-empty lines of file f2.

✓ Consider another command as follow:

```
$grep -c '^$' f2 2
```

```
$
```

It counts all empty lines (consist of only new-line character) of file f2.

**(2)-l (list):** It displays only the names of files in which a pattern has been found.

For example, consider a command as follow:

```
$grep -l '.' *
```

It displays names of all files of current directory that contains any character in

✓ You can print names of all files of current directory that contains pattern echo anywhere in a file then the command is as follow:

```
$grep -l 'echo' *
```

**(3) -n (number):** It can be used to display the line numbers containing the pattern, along with the lines.

If you want to print line number before matched line then the command is as follow:

```
$ grep-n' Unix f2
```

```
1: Unix and shell programming
```

```
5: Unix OS
```

```
$
```

It prints two column output, each column delimited by colon (;). In the 1<sup>st</sup> column, line number will be displayed and 2<sup>nd</sup> column contains content of matched lines.

✓ You can give more than one filename as input files as follow:

```
$ grep-n 'Unix' f1 f2
```

```
f1: 1:sco Unix
```

```
f2:1: Unix and shell programming
```

```
f2:5: Unix OS
```

```
$
```

It prints output in three columns, each column delimited by colon (:). The 1<sup>st</sup> field contains name of file, 2nd column contains line number and last column contains content of matched line.

**(4). -v (inverse):** The -v option select all but not the lines containing the pattern.

✓ Sometimes, a user is interested only on unmatched lines then he used -v option as follow:

```
$grep -cv '^$' f2
```

```
5
```

```
$
```

It counts all non-empty lines (contains only new-line character) of file f2.

✓ Consider another command as follow:

```
$grep -v 'Unix' f1
```

```
The red hat linux
```

```
user name user1
```

```
$
```

It displays lines of file f1 which do not contains Unix pattern.

**(5)-i (ignore):** It ignores case in pattern matching.

- ✓ For example, you want to print lines that contains pattern unix in any case then the command is as follow:

```
$grep -i 'unix' f1
```

```
sco Unix
```

```
$
```

It displays lines of file f1 having unix pattern in any case.

**(6) -h (hide):** It omits filenames when handling multiple files.

- ✓ For example, consider an example as follow:

```
$grep -h 'Unix' f1 f2
```

```
sco Unix
```

```
Unix and shell programming Unix OS
```

```
$
```

It displays lines of files f1 and f2 which contains pattern Unix. It does not display filename before, matched line i.e. it hides name of a file.

**(7)-e Reg Exp :** You can specify regular expression with this option. You can use this option multiple times.

- ✓ For example, you want to locates lines of file which contains patter either Unix or linux then the command is as follow:

```
$ grep -e 'Unix'-e 'linux' f2
```

```
Unix and shell programming
```

```
red hat linux
```

```
Unix OS
```

```
$
```

**(8) -f fname:** A list of strings to be match is stored in file name.

- ✓ For example, consider a patfile as follow:

```
$ cat patfile Unix linux
```

```
$
```



It contains list of pattern in a separate lines. Now, we want to locate lines of file f1 that contains any of the pattern given in file patfile then the command is as follow:

```
$ grep -f patfile f1
```

```
sco Unix
```

```
The red hat linux
```

```
$
```

## Grep family

There is a small family of grep utility which includes **egrep** and **fgrep**. These two utilities operate in a similar way to grep but each has its own particular usage, and there are small differences in the way that each work.

Both utilities search for specific pattern in either the standard input stream or a series of input files supplied at command-line.

## egrep

egrep stands for extended grep. It was invented by Alfred Aho. It extends grep's pattern-matching capabilities in two major ways.

- ✓ It admits alternates
- ✓ It enables regular expressions to be bracketed/grouped using the pair of parenthesis (i.e. (...)), also known as factoring.

It offers all the options and regular expression metacharacters of **grep**, but its most useful feature is the facility to specify more than one pattern for search. While **grep** uses some more characters that are not recognized by **egrep**, **egrep** includes some additional extended metacharacters not used by either grep or **sed** utilities that are given in table-(c.12).

Expression	Meaning
ch+	It matches one or more occurrence of character ch.
ch?	It matches zero or one occurrence of character ch.
exp1\exp2	It matches expression exp1 or exp2.
(x1\ x2)x3	It matches expression x1x3 or x2x3.

Let us consider the following examples which uses extended metacharacter:

- ✓ To display lines which contains any character that occur one or more time then the command is as follow:

```
$egrep m+ f2
```

```
Unix and shell programming  
program und process  
$
```

It prints lines of file f2 that contains character 'm' occur one or more times.

- ✓ If you want to locates lines which contains one of more patterns then you can use alternate metacharacter as follow:

```
$egrep 'Unix\linux' f1
```

```
sco Unix
```

```
The red hat linux  
$
```

It displays lines of file f1 which contains pattern either Unix or linux.

- ✓ Sometimes, you want to display lines which contains either software or hardware then the command is as follow:

```
$egrep "(soft| hard) ware" f1
```

**NOTE:** In grep, if a pattern contains some special characters then it must be quoted.

#### **-f option: Storing pattern in a file**

egrep provides a facility to take patterns from a file. If there are number of pattern that you have to match; **egrep** offers the **-f (file)** option to take such patterns from the file. For example, a file patfile contains patterns in which each pattern is delimited by'|' as follow:

```
$ cat patfile
```

```
Unix linux
```

```
$
```

Now, you can execute **egrep** with the -f option in this way:

```
$grep -f patfile f1
```

```
sco Unix
```

```
The red hat linux
```

```
$
```

Here, the command takes the pattern/expression from file patfile and display matched lines of file f1.

## fgrep

**fgrep** stands for fixed/fast **grep**. The **fgrep** utility can normally only search for fixed strings i.e. character string without embedded metacharacters. However, some implementations of the **fgrep** utility allow it to be used with a few metacharacters - check your version to make sure. **fgrep** accepts multiple patterns, both from the command line and a file, but unlike **grep** and **egrep**, does not accept regular expressions. So, if the pattern to be search is a simple string, or a group of them,

**fgrep** is recommended. It is arguably faster than **grep** and **egrep**, and should be used when using fixed strings.

Alternative patterns in **fgrep** are specified by separating one pattern from another using the new-line character. This is unlike in **egrep**, which uses the '|' to delimit two expressions. You may either specify these patterns in the command line itself, or store them in a file.

- ✓ For example consider a file patfile which contains list of pattern delimited by new-line character as follow:

```
$ cat patfile
```

```
Unix
```

```
linux
```

```
$
```

We can use this file using -f option as follow:

```
$fgrep -f patfile f1
```

```
sco Unix
```

```
The red hat linux
```

```
$
```



- ✓ You can achieved same output without using file patfile by supplying patterns at command-line as follow:

```
$fgrep 'Unix <enter>
```

```
> linux'fl < enter>
```

```
sco Unix
```

```
The red hat linux
```

```
$
```

- ✓ The disadvantage with **grep** family is that none of them has separate facilities to identify fields. This limitation is overcome by **awk** utility.

Limitation of **grep** family:

The **grep** family has following limitation.

- ✓ It cannot be used to add, delete or change a line.
- ✓ It cannot be used to print only part of a line.
- ✓ It cannot read only part of a file.

It cannot select a line based on the contents of the previous or the next line. There is only one buffer, and it holds only the current line.

Following table-(d.12) shows the atoms used in regular expression by **grep** family:

**Table-(d.12):Atoms used by grep family**

Atoms	<b>grep</b>	<b>fgrep</b> ☆	<b>egrep</b>
Character	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Dot	<input type="checkbox"/>	<b>X</b>	<input type="checkbox"/>
Class	<input type="checkbox"/>	<b>X</b>	<input type="checkbox"/>
Anchors	<input type="checkbox"/>	<b>X</b>	<input type="checkbox"/>
Back Reference	<input type="checkbox"/>	<b>X</b>	<input type="checkbox"/>

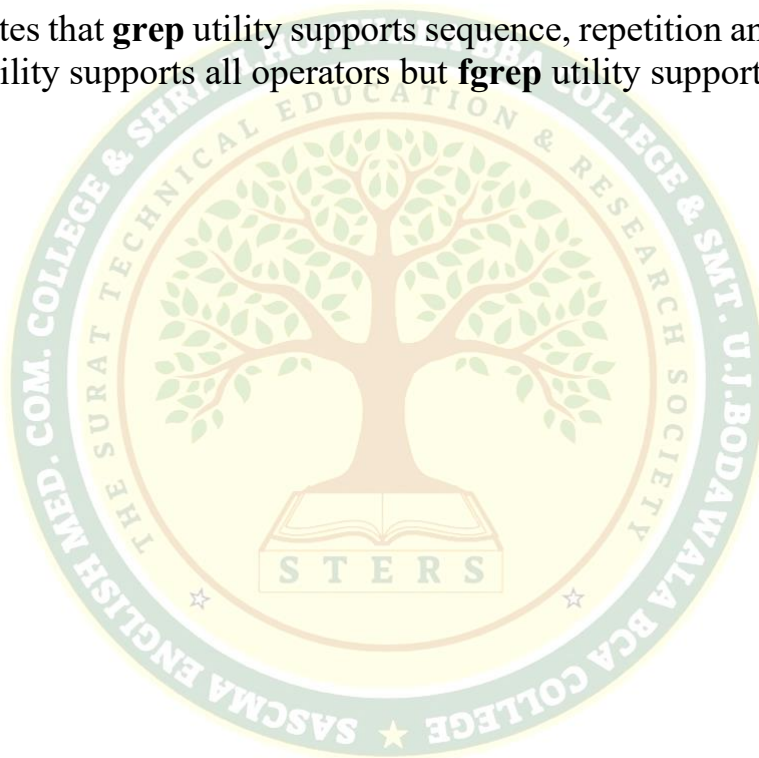
As shown in table-(d.12), both **grep** and **egrep** utilities allows all the atoms in regular expression whereas **fgrep** utility supports only character atom.

Similarly, table-(e. 12) shows the operators used in regular expression by **grep** family:

Table-(e.12): Operators used by grep family

Operators	grep	fgrep	egrep
Sequence	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Repetition	<input type="checkbox"/>	<b>X</b>	<input type="checkbox"/>
Alteration	<b>X</b>	<b>X</b>	<input type="checkbox"/>
Group	<b>X</b>	<b>X</b>	<input type="checkbox"/>
Save	<input type="checkbox"/>	<b>X</b>	<input type="checkbox"/>

Table-(e.12) indicates that **grep** utility supports sequence, repetition and save operators, **egrep** utility supports all operators but **fgrep** utility supports only sequence operator.



### 4.3 Stream Editing with sed (search, replace, line deletion, insertion)

The SED command (short for Stream Editor) is one of the most powerful tools for text processing in Linux and Unix systems. It's commonly used for tasks like search and replace, text transformation, and stream editing.

#### Sed Command Syntax:

The **basic syntax** for using the **SED command** in Linux is:

```
sed [OPTIONS] 'COMMAND' [INPUTFILE...]
```

where,

#### Commonly Used SED Command Options

Below are some of the most frequently used **SED command options**, let's check them out:

Option	Description
<b>-i</b>	Edit the file in-place (overwrite)
<b>-n</b>	Suppress automatic printing of lines.
<b>-e</b>	Allows multiple commands.
<b>-f</b>	Reads sed commands from a file.
<b>-r</b>	Use extended regular expressions.

Here are some basic **SED commands** that will help you get started with text manipulation. 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.

### 1. Sample Commands

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

To replace only the **nth occurrence** of a word in a line, use the following syntax:

```
sed 's/old_word/new_word/n' filename
```

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

Here, we will use the **g** flag to replace **all the occurrences** of a pattern in a line. Let's check out the syntax below:

```
sed 's/old_word/new_word/g' filename
```

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. Parenthesize First Character of Each Word**

This sed example prints the first character of every word in parenthesis.

```
echo "Welcome To The Geek Stuff" | sed 's/^(b[A-Z]\)(\1)/g'
```

**Output:**

(W)elcome (T)o (T)he (G)eek (S)tuff

**6. 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.

**7. Duplicating the Replaced Line with /p flag**

The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.

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

**Output:**

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

**8. Printing Only the Replaced Lines**

Use the -n option along with the /p print flag to display only the replaced lines. Here the -n option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time.

```
sed -n 's/unix/linux/p' geekfile.txt
```

**Output:**

linux is great os. unix is opensource. unix is free os.

linux linux which one you choose.

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

If you use -n alone without /p, then the sed does not print anything.

## 9. 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.

## 10. 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
```

**4. To Delete from nth to last line****Syntax:**

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

**Example:**

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

**5. To Delete pattern matching line****Syntax:**

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

**Example:**

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

