Unit 4 : Advanced Text Processing Tools
4.1 Introduction to Regular Expressions (Basic and Extended)
4.2 Pattern Matching using grep, egrep, and fgrep
4.3 Stream Editing with sed (search, replace, line deletion, insertion)

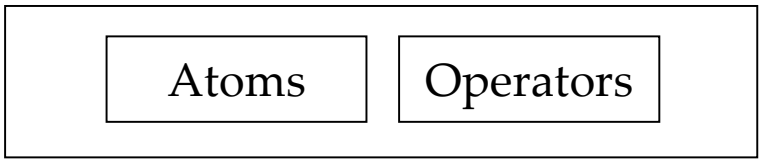
Advance filtering utilities:

- **REGULAR EXPRESSIONS:**
  - It is consist of a sequence of characters that is used to match against text.
  - Regular expression consists of atoms and operators.
  - Atom specifies **what we are looking for**.
  - Operators are used to **combine atoms into complex expression**.

Note: In Linux, **`grep` regular expressions (regex)** are patterns used to search for text. Instead of searching for an exact word, regex allows you to search for patterns such as numbers, words starting with a specific letter, repeated characters, etc.

Basic Syntax: **grep "pattern" filename**
Example: grep "hello" file.txt
Here, Searches for lines containing the word **hello**.

- **Structure of Regular Expression:**

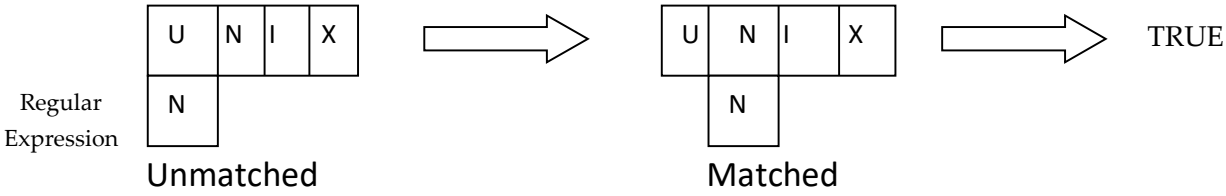


- **ATOMS:**
  - Atoms available in regular expression are:
    - ✓ Single character
    - ✓ Dot character
    - ✓ Class character
    - ✓ Anchor
    - ✓ Back reference

1) Single character:

- A single character matches itself.
- Length of single character atom is one.
- Comparison is done **character by character** and if **match found** then it returns **TRUE, else FALSE**.

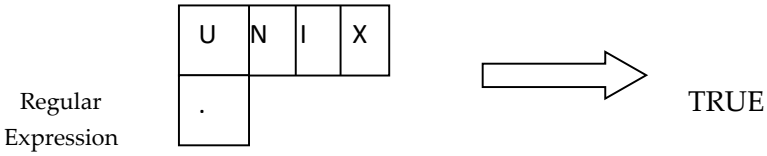
- Example,
  The successful and unsuccessful matches of regular expression with a text.



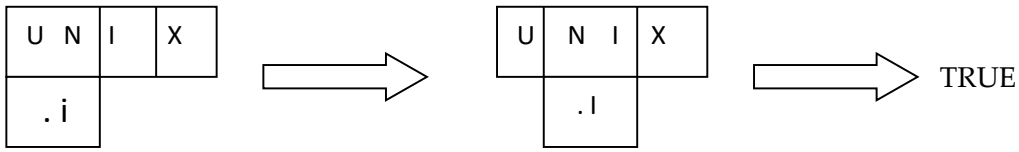
(Successful Search)

2) Dot Character:

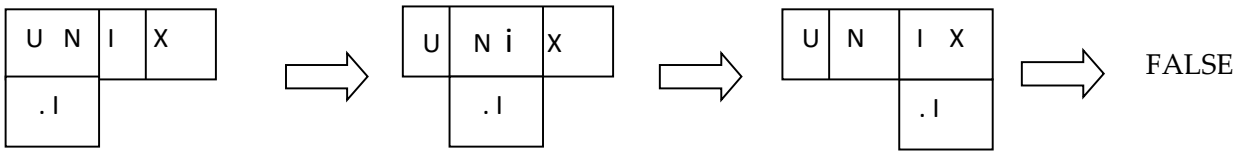
- This atom is denoted by dot (.)
- It is also single character atom that matches any single character except new line character.
- If any **single character occurs anywhere** in a text then the **pattern match succeed otherwise failed**.
- Example,
  Successful match of regular expression:



(Successful Search)

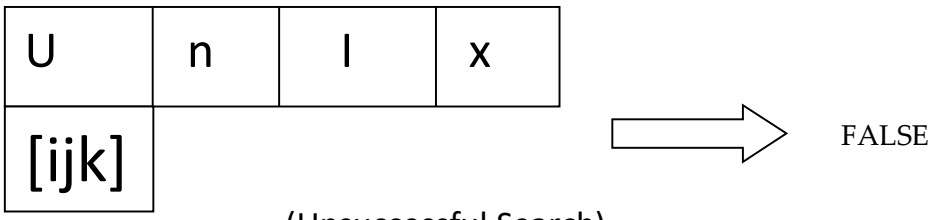


(Successful Search)

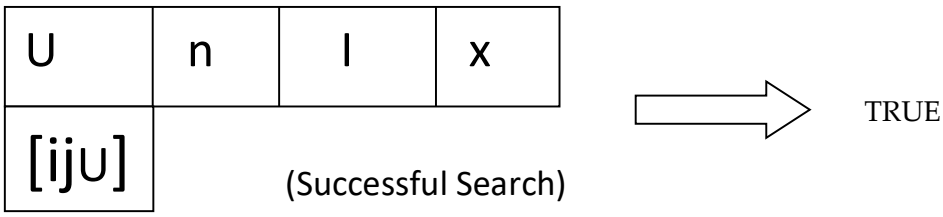


3) Class Character:

- It defines set of ASCII characters within a pair of square bracket.
- Length of this character atom is one.
- If any of the character within the class matches in a text then the pattern match is succeeded otherwise not.
- Example



(Unsuccessful Search)



(Successful Search)

- Regular expression that matches only vowels:
  [aeiouAEIOU]
- Regular expression that matches any text which contains numeric digit 0-9:
  [0-9]
- Regular expression that matches only alphabets:
  [A-Za-z]
- Regular expression that matches any character other than digit is:
  [^0-9] **//no digits only characters**
- Regular expression that matches character other than vowel is:
  [^AEIOUaeiou]
- Regular expression that matches all digits and dash:
  [0-9\-]
- Regular expression that matches text that contains alphabets digit and ^
  [\^0-9A-Za-z]
- Regular expression that matches any character other than alphabet is:
  [^A-Za-z]
- Regular expression that matches any special character
  [^A-Za-z0-9]
- Regular expression that matches only capital vowel:
  [AEIOU]

4) **Anchor**:
   - Anchors are the atoms that are not matched to text but it defines where the character in the regular expression must be located in a text.
   - There are 4 types of anchor:

Anchor	Meaning

^	It matches pattern at the beginning of line
$	It matches pattern at the end of line
\<	It matches pattern at the beginning of word
\>	It matches pattern at the end of word

Example:

- ✓ ^a → It matches a line that starts with character 'a'.
- ✓ a$ → It matches a line that ends with character 'a'.
- ✓ \<a → It matches a line in which any word starts with character 'a'.
- ✓ a\> → It matches a line in which any word ends with character 'a'.

5) Back Reference:

- It is used **to match one or more characters to text**, previously saved in a buffer. We can use **up to 9 buffers**.
- So we can use **9 back reference (\1,\2,…,\9**).
- Back references are used with **save operator.**

➢ **OPERATORS:**

- It plays powerful role in creation of regular expression.
- To combine atoms with operator, we can create more complex regular expression.
- 5 types of operators:
  - ✓ Sequence Operator
  - ✓ Alternation Operator
  - ✓ Repetition Operator
  - ✓ Group Operator
  - ✓ Save Operator

1) Sequence Operator:

- This operator concatenates a series of atoms in a regular expression. Then this operator is implicitly included between them.
- Whenever we use one or more atoms one after another in a regular expression then there is one sequence operator between each of them.
- Examples of sequence operator:

Expression	Meaning
Software	Matches pattern 'Software'.
^The	Matches pattern 'The' at the **beginning of the line.**
\<[a-z]..[0-9]\>	Matches **4-characters** pattern that starts with **small alphabet and end with digit.**

2) Alternation Operator:

- This operator is **denoted by pipe (|)**
- It is used to define **one or more alternatives of patterns**.

- Examples,

Expression	Meaning
Hardware | Software	Matches pattern 'Hardware' or 'Software'.
Unix | UNIX	Matches pattern 'UNIX' or 'unix'.

3) Repetition Operator:

- It is represented by a pair of escaped curly braces i.e. \{…\}
- It specifies that the atom or expression written before may be repeated.
- It should be written as: **atom/expression \{m, n\}**
- It shows that **previous** atom or expression will be **repeated from m to n times**.
- The expression written in escaped curly braces is known as **repetition operator**.
- It is also known as **braced regular expression** (BRE).
- Example,

◆ Regular Expression Table Explained		
Regular Expression	**Meaning in Simple Words**	**Example**
a{5,10} OR a\{5,10\}	**The letter 'a' must repeat between 5 and 10 times.**	**aaaaa (5 times)**
		aaaaaaaaaa (10 times)
		aaa (only 3 times, not enough)
[a-zA-Z]{15,20} OR [a-zA-Z]\{15,20\}	**Any alphabet (small a–z or capital A–Z) must repeat between 15 to 20 times.**	**abcdefghijklmno (15 letters)**
		ABCDEFGHIJKLMNOPQRST (20 letters)
		hello (only 5 letters)
.{5,10} Or .\{5,10\}	**Any character (letter, digit, symbol, space) must repeat between**	**abcde (5 characters)**

	5 to 10 times.	1234567890 (10 characters)
		hi (only 2 characters, not enough)
[a-zA-Z]{10} OR [a-zA-Z]\{10\}	Exactly 10 alphabets in a row (only letters, no digits/symbols).	ABCDEFGHIJ (10 letters)
		helloworld (10 letters)
		hello12 (false because digits included)
[a-zA-Z]{5,} OR [a-zA-Z]\{5,\}	5 or more alphabets. (No upper limit).	hello (5 letters)
		programming (11 letters)
		hi (only 2 letters, not enough)
[a-zA-Z]{,5} OR [a-zA-Z]\{,5\}	At most 5 alphabets (0 to 5 allowed).	`` (empty, 0 letters)
		cat (3 letters)
		apple (5 letters)
		banana (6 letters, not allowed)
Ch* (same as ch\{0,\}) OR Ch* is equivalent to ch\{0,\}	The string "ch" can appear 0 or more times.	`` (empty)
		ch
		chchch
		c (not complete "ch")

ch{0,1} OR ch\{0,1\}	**The string "ch" can appear either 0 times or 1 time.**	**`` (empty)**
		ch
		chch (too many)
ch{1,} OR ch\{1,\}	**The string "ch" must appear at least 1 time, or more.**	**ch**
		chch
		chchch
		`` (empty, not allowed)

4) **Group Operator:**
   - It is a **pair of opening and closing parenthesis** that allows the **next operator** to be applied to the **whole group.**
   - It can be written as: **(exp1|exp2|exp3|…|exp N) exp**
   - It **concatenates** any of the expression **between parentheses with exp**.
   - Example,

REGULAR EXPRESSION	MEANING
(unix|linux)OS	Match a line which contains pattern unix OS or linux OS
(hard|soft|firm)ware	Match a line which contains pattern hardware or Software or firmware.

Symbol	Meaning	Example
.	Any single character	gr.p matches grep, grap, gr1p
^	Beginning of line	^Linux
$	End of line	Linux$
*	Zero or more occurrences	ab*
[ ]	Character class	[aeiou]
[^]	Negation	[^0-9]
[a-z]	Range	[a-z]
\{n\}	Exactly n times	a\{3\}
\{m,n\}	Between m and n times	a\{2,5\}
\?	Zero or one occurrence	colou\?r
\+	One or more occurrences	a\+
|	OR operator	cat|dog

5) **Save Operator:**

- In Linux **regular expressions**, the **save operator** is written as \(...\)
- Whatever is inside \(...\) is **saved (remembered) in a buffer**.
- You can use it later in the same expression by writing \1, \2, … up to \9.
    - \1 means "repeat whatever was saved in the first \(...\)".
    - \2 means "repeat whatever was saved in the second \(...\)".

Think of it like **copy-paste inside regex**. You save a part once and later call it again.

[Note:
In **traditional Linux regular expressions (like grep, sed)**, only **\1 to \9** are supported.
That means you can save **maximum 9 groups** using the save operator \(...\) and recall them later.

Why only 9?

- These tools are based on **POSIX Basic Regular Expressions (BRE)**.
- POSIX BRE defines only **9 back-references**.
- So buffers are fixed: \1, \2, …, \9.

]

General Form

\(exp1\)\(exp2\)…\(exp9\)exp

- exp1 goes into buffer 1 → referred by \1
- exp2 goes into buffer 2 → referred by \2
- and so on (max 9 buffers).

Examples

1. **Match a line that starts and ends with the same character**

    ^\(.\).*\1$
    - \(.\) → saves the **first character** in buffer 1.
    - .* → allows anything in between.
    - \1 → must match the same character again at the end.

    Example:
    - aba → starts with a, ends with a.
    - 1231 → starts with 1, ends with 1.
    - hello → starts with h, ends with o. (no match)

Understanding:

Regex:

^\(.\).*\1$

How it works:

Input:

a b c a

```
  ^       ^
  |       |
\(.\)    \1
```

Step 1: \(.\) → saves first 'a' in buffer \1
Step 2: .* → matches 'bc'
Step 3: \1 → must be 'a' again
Answer: Match

2. **Match repeated letters like "hello", "programming" (double letters)**

   \(.\)\1.*

   - \(.\) → saves a character in buffer 1.
   - \1 → immediately repeats the same character.

   Example:
   - hello → ll matched.
   - programming → mm matched.
   - world → no double letters.

Understanding: Match double letters (like "hello", "programming")
Regex:

\(.\)\1.*

How it works:
Input:

```
  h  e  l  l  o
 ^  ^
 |  |
\(.\) \1
```

Step 1: \(.\) → saves 'l' in buffer \1
Step 2: \1 → expects 'l' again
Step 3: .* → matches rest 'o'
Answer: Match

3. **Match palindromes like madam, nayana, 12321**

   \(.\)\(.\).\2\1.*

   - \(.\) → saves **first character** into buffer 1.
   - \(.\) → saves **second character** into buffer 2.
   - . → middle character (any, not saved).
   - \2\1 → must repeat the **second and first characters in reverse order**.

Example:

- madam → (m in \1, a in \2, then middle d, then a = \2, m = \1).
- 12321 → works the same.

Understanding:

Regex:

\(.\)\(.\).\2\1.*

How it works:

Input:

m a d a m
^ ^ ^ ^
| | | |
\(.\)\(.\) \2 \1

Buffer1 = 'm'
Buffer2 = 'a'
Middle = 'd'
\2 = 'a'
\1 = 'm'
Answer: Match

4. **Match same digit at start and end (like 11, 1abcfd1, 4hd4)**

   ^\([0-9]\).*\1$

   - \([0-9]\) → saves first digit into buffer 1.
   - .* → anything in the middle.
   - \1 → must be the same digit at the end.

   Example:
   - 11 → starts and ends with 1.
   - 1abcfd1 → first and last 1.
   - 4hd4 → first and last 4.
   - 123 → starts with 1 but ends with 3.

- \(...\) → saves part of regex.
- \1, \2, ... → recall saved parts later.
- Useful for **matching repeated or mirrored patterns** like palindromes, repeated letters, or same start and end.

Understanding:

Regex:

^\([0-9]\).*\1$

How it works:

Input:

```
  1 a b c 1
  ^       ^
  |       |
\(.\)     \1
```

Step 1: Save first '1' in buffer \1
Step 2: .* matches 'abc'
Step 3: \1 expects '1' again at end
Answer: Match

grep

- **Full form:** Globally search Regular Expression and Print.
- It is a **pattern matching tool** in Linux.
- **Use:** Searches text for a pattern and shows matching lines (with line numbers or filenames if needed).
- **Pattern = Regular Expression.**

Syntax:

grep [options] pattern filename(s)

- Works like a **filter** – can take input, search, and send output to a file.

Examples:

1. who | grep kumar > file2 → finds "kumar" from who command output and saves in file2.
2. grep "sales" file1 → finds "sales" in file1.
3. grep "director" f1 f2 → searches "director" in two files.
4. grep 'jai sharma' f1 → quotes needed if pattern has spaces.
5. Use **double quotes** if pattern has variables or command substitution.
6. If pattern not found, grep shows nothing.

[
Note:

Use of double quotes

In Linux,

- **Single quotes (' ')** → take the text **exactly as it is** (no variable or command substitution).
- **Double quotes (" ")** → allow **variables ($var)** or **command substitution ($(command))** to be expanded before running grep.

Example with Variable

word="sales"

grep "$word" file1

- Here "$word" → becomes **sales**.
- So command is same as:
- grep sales file1

Answer: Matches lines containing sales in file1.

If you had used **single quotes**:
grep '$word' file1

- It will search for the **literal text** $word (not the value of variable).

Example with Command Substitution

grep "$(whoami)" users.txt

- $(whoami) → expands to current username, e.g. ajay.
- Command becomes:

grep rakesh users.txt

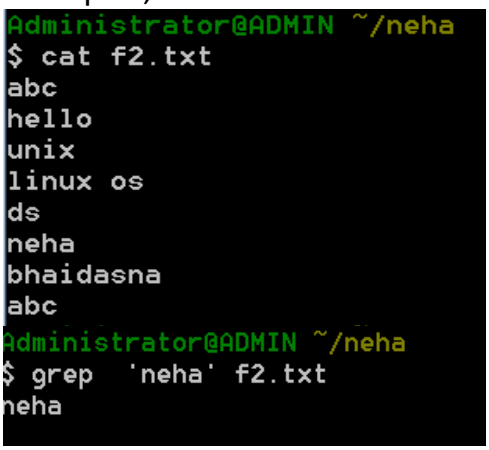
Answer: Finds lines containing your username in users.txt.
If you used single quotes:
grep '$(whoami)' users.txt

- It will search for the literal text $(whoami) instead of your username.

]

GREP COMMAND OPTIONS:

➢ Examples,



OPTIONS	SIGNIFICANCE
-i	**Ignores** case for matching $grep –i 'NEHA file1 
-v	**Doesn't** display lines **matching expression**

	$grep –v ‘director’ file1>f1 #selects unmatched lines Administrator@ADMIN ~/neha $ grep -v 'neha' f2.txt abc hello unix linux os ds bhaidasna abc

-n	Display **line number** along with line
	$grep –n ‘marketing’ file1
	
	#displays line number with selected lines at which line that line is present
	
	Administrator@ADMIN ~/neha
	$ grep -n 'abc' f2.txt
	1:abc
	8:abc
	
	Administrator@ADMIN ~/neha
	$ grep -n 'neha' f2.txt
	6:neha

-c	Displays count of **occurrences** $grep –c ‘director’ file1 #output:3 $ grep –c director emp*.lst # counts lines containing pattern Output: emp.lst:4 emp1.lst:2 empold.lst:6 emp2.lst:6
-l	Display list of **filenames only** $ grep -l 'abc' *.txt #displays name of file containing pattern
-e exp	Specifies expression exp with this option. It can use multiple times. $grep –e “Agarwal” -e “aggarwal” -e “agarwal” file1

	# -e is used to select multiple patterns at a time
-x	Matches pattern with **entire line**
-E	Treats pattern as an extended regular expression(ERE)

- **BASIC REGULAR EXPRESSION (BRE):**

SYMBOLS OR EXPRESSION	MATCHES
*	Zero or more occurrences of the previous character
g*	Nothing or g,gg,ggg etc
.	A **single characte**r OR Matches any **one character**
.*	Nothing or any number of characters
[pqr]	A single **character p,q or r** Matches any **one of a set** characters
[c1-c2]	A single character within the **range** represented by c1 and c2 Matches any one of a range characters
[1-3]	A digit between **1 and 3**
[^pqr]	A single character which is **not** a p,q or r
[^a-zA-Z]	A non-alphabetic character
^pat	Pattern pat at **beginning of line**
pat$	Pattern pat at **end of line**
bash$	bash at **end of line**
^bash$	bash as the only word in line
^$	**Lines containing nothing**

```
Administrator@ADMIN ~/neha
$ cat test.txt
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.

Administrator@ADMIN ~/neha
$ grep '^cat' test.txt
cat COMMAND for file oriented operations.

Administrator@ADMIN ~/neha
$ grep 'ons.$' test.txt
cat COMMAND for file oriented operations.
```

- **THE CHARACTER CLASS:**
  - $ **grep "[aA]g[ar][ar]wal" f1**

- $ **grep "[aA]gg*[ar][ar]wal" f1**

- **THE DOT (.)**
  $ **grep "j.*saxena" f1**
  $ **grep a.*Agarwal f1**
  2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
  $ grep a.Agarwal f1

- **SPECIFYING PATTERN LOCATION(^ and $)**
  - **$ - for matching at the end of line**
  - Example,
    $ grep "6...$" f1
    2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
    1006 | chanchal singhvi | director | sales | 13/9/87 | 6700

    $ grep "2...$" f1
    2567 | anju agarwal | accountant | purchase | 12/7/76 |2000

  - **^ - for matching at the beginning of a line**
  - Example,
    $ grep "^[^2]" f1 # Displays the files which are not beginning with 2
    $ ls –l | grep "^d" #shows only directory
    $ ls -l | grep ^d

```
drwxrwxr-x   3 nirzari  nirzari   4096 Jul 28 02:26 d1
dr-xr-xr-x   2 nirzari  nirzari   4096 Jul 29 02:13 d3
drwxrwxr-x   2 nirzari  nirzari   4096 Jul 29 05:40 d5
drwxrwxr-x   3 nirzari  nirzari   4096 Jul 14  2014 d6
```

➢ EXTENDED REGULAR EXPRESSIONS(ERE) AND egrep:

- **Solaris** user uses grep for extended regular expression(ERE) with –E option.
- If your system not support this then use egrep without –E
- The ERE set includes 2 special characters:
  1. **+** → It is used to matches **one or more occurrence** of the previous character
  2. **?** → It is used to matches **zero or one occurrence** of the previous character

Example:
$ grep –E "[aA]gg?arwal" f1
$ **grep -E [aA]gg?arwal f1**
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76 |2000

$ grep [aA]gg?arwal f1

$ **egrep [aA]gg?arwal f1**

2476 | anil Agarwal | manager | sales | 12/7/56 | 5000

2567 | anju agarwal | accountant | purchase | 12/7/76 |2000

$ grep –E "#?include+<stdio.h>"

- **<u>MATCHING MULTIPLE PATTERNS (|,(AND))</u>**
  - Pipe is the delimiter for multiple patterns.
  - **Note**: here single quote is must
  - Example,

    $ **grep –E 'sengupta|dasgupta' f1**

    The characters '(' and ')' group pattern, and do same as above

    $ **grep –E '(sen|das)gupta' f1**
  - ERE's when combines with BRE's forms very powerful regular expression.
  - Example: **$ grep –E 'agg?[ar]+wal' file1**

  - **<u>The Extended Regular expression(ERE) used by grep, egrep and awk are as follows:</u>**

EXPRESSION	**SIGNIFICANCE**
Ch +	Matches one or more occurrence of character ch
Ch?	Matches zero or one occurrence of character ch
Exp1|exp2	Matches exp1 or exp2
(x1|x2)x3	Matches x1x3 or x2x3
(lock|ver)wood	Matches lockwood or verwood

```
Administrator@ADMIN ~/neha
$ cat test.txt
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.

Administrator@ADMIN ~/neha
$ egrep m+ test.txt
cp command for copy files or directories.
ls command to list out files and directories with its attributes.

Administrator@ADMIN ~/neha
$ egrep M+ test.txt
cat COMMAND for file oriented operations.

Administrator@ADMIN ~/neha
$ egrep 'cp|ls' test.txt
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
```

- **FGREP : SEARCH A FILE FOR A FIXED -** CHARACTER STRING**:**
  - fgrep command is used to extract **fixed patterns**.
  - Patterns cannot extract from character class or special meta-character. Here f in fgrep stands for fixed pattern.
  - If pattern to be searched is a simple string, or a group of string then fgrep command is recommended.
  - fgrep command is faster than grep and egrep.
  - Example,
    - $ fgrep -x 'manager' f1
    - $ fgrep 'manager' f1
    - $ fgrep manager f1

      1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
      2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
      1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
      2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
    - $ fgrep '[a-z]*' f1
    - $ grep '[a-z]*' f1

      2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
      1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
      1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
      2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
      2567 | anju agarwal | accountant | purchase | 12/7/76 |2000
  - **#more than one pattern can be given by a newline character**
    - $ fgrep 'manager
      sales' f1

      2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
      1006 | chanchal singhvi | director | sales | 13/9/87 | 6700

1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000

- # to take pattern from a file
  $ fgrep -f f2 f1
  2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
  1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
  1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
  2476 | anil Agarwal | manager | sales | 12/7/56 | 5000

 # points to be noted:

1. In egrep, if a pattern contains some special characters then it must be quoted.
2. –f option to extract pattern from a file also works in egrep.

 # Some more grep commands:

1. grep '.' F1 #displays all lines except blank line.
2. grep '\.' F1 #hides special meaning of . ,dot can occur anywhere in a line
3. quotes compulsory in variable substitution as:
   $a=1
   $ grep "$a" f1 #display lines containing 1.
4. quotes compulsory in command substitution:
   $ grep "`echo hello`" f1
   $ grep " `echo sales` " f1
   2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
   1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
   1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
   2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
5. $ grep 'mm*' f1 # * means 0 or more occurrence of previous character i.e. 0 or
   More time 'm' is occurred
6. $ grep '^.\{8\}$' f1 # Display the lines from file having length 8
7. $ grep '^.\{5,15\}$' f1 # Display the lines from file having Minimum length 5 and
   Maximum length 15
8. $ grep '^\(.\).*\1' f1

SED : THE STREAM EDITOR:

- A special editor for **modifying files automatically**.
- To write a program to make changes in a file, sed tool is use.
- Sed command is the ultimate stream editor.
- Sed command performs non-interactive operations on a data stream.
- Sed command uses **instructions** to act on text.

- An instruction combines an address for selecting lines, with an action to be taken on them.
- 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 'address action' file(s)**
- The address and action are enclosed within single quotes

sed commands:

- **The sed s**upports several commands. Commands are used to apply on specified lines. They are

1. **Print**
2. **Quit**
3. **Line number**
4. **Modify**
5. **Files**
6. **Substitute**

1. **Print :**

   It is denoted by character **p.**

   It prints selected lines on standard output.

   p (print) action shows all lines as well as selected lines ,

   so selected lines will comes twice to remove **delicacy –n** option is used with p(print action).

   Example:

```
Administrator@ADMIN ~/neha
$ cat g1.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.

Administrator@ADMIN ~/neha
$ sed '1,3p' g1.txt
unix is great os. unix is opensource. unix is free os.
unix is great os. unix is opensource. unix is free os.
learn operating system.
learn operating system.
unix linux which one you choose.
unix linux which one you choose.
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

Administrator@ADMIN ~/neha
$ sed  -n '1,3p' g1.txt
unix is great os. unix is opensource. unix is free os.
learn operating system.
unix linux which one you choose.
```

A special character ($) is used print last line of an input file.

```
Administrator@ADMIN ~/neha
$ sed  -n '$p' g1.txt
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

- command **p without any option** displays **all lines** from particular file. **$ sed -n p g1.txt**
- select non contiguous group of lines of input file then command is as follow:

```
Administrator@ADMIN ~/neha
$ cat g1.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.
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.
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.

Administrator@ADMIN ~/neha
$ sed -n '1,3p
5,9p
$p' g1.txt
unix is great os. unix is opensource. unix is free os.
learn operating system.
unix linux which one you choose.
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.
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.
```

It will display **1 to 3 lines**, **5 to 9** and **last line** from file g1.txt

- **Selecting lines from anywhere:** **$ sed -n '9,11p' f1**
- **Negating the action(!):** **$**Sed –n '3,$!p' f1

[don't print line 3 to the end]

2. Quit:

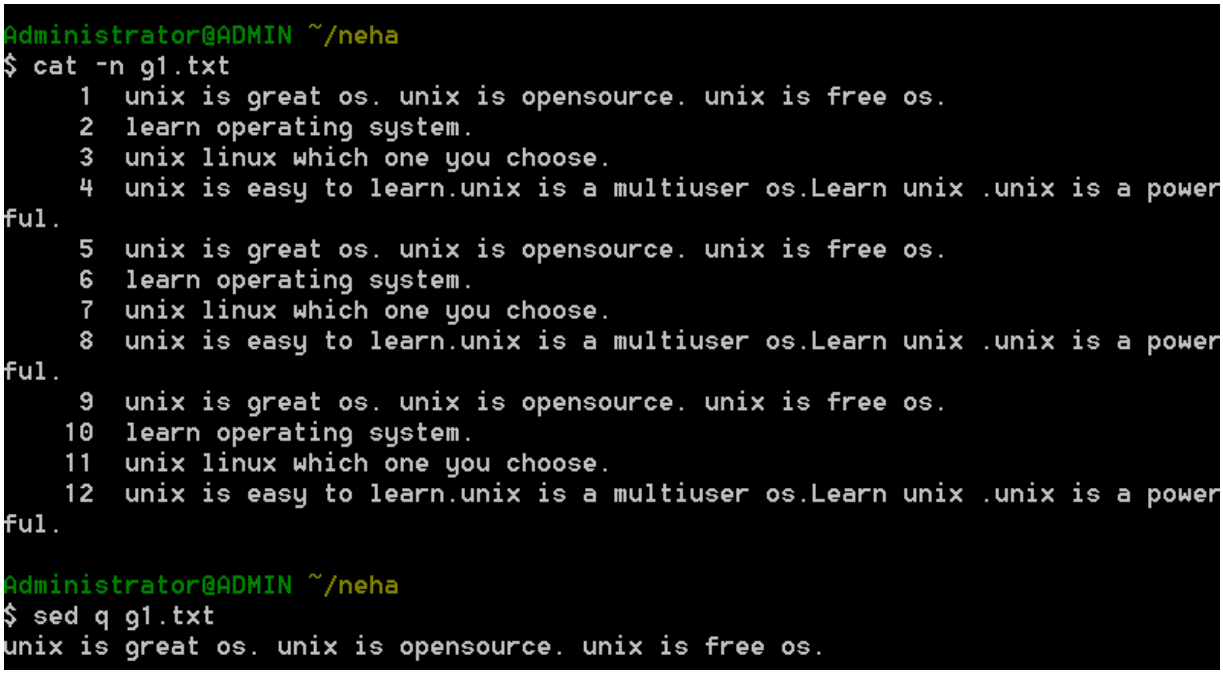
This command denoted by **character q**.

It uses a single address i.e. it does not allow range of address.

It quits after reading up to address lines.

Example:

q without address prints first line.



LINE ADDRESSING **:$ sed '3q' f1** #quits after line number 3,here 3 is address and q(quit) is action

3. Line number:

It is denoted by equal (=).

It write line number of addressed line at the beginning of line.

It similar to -n option of grep. but here line numbers are written in separate line.

Example;

```
Administrator@ADMIN ~/neha
$ grep -n 'linux' g1.txt
3:unix linux which one you choose.
7:unix linux which one you choose.
11:unix linux which one you choose.

Administrator@ADMIN ~/neha
$ sed '=' g1.txt
1
unix is great os. unix is opensource. unix is free os.
2
learn operating system.
3
unix linux which one you choose.
4
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
5
unix is great os. unix is opensource. unix is free os.
6
learn operating system.
7
unix linux which one you choose.
8
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
9
unix is great os. unix is opensource. unix is free os.
10
learn operating system.
11
unix linux which one you choose.
12
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

To print only line numbers -n option is used.

```
Administrator@ADMIN ~/neha
$ sed -n '=' g1.txt
1
2
3
4
5
6
7
8
9
10
11
12
```

To print last line number $ is used with -n.

```
Administrator@ADMIN ~/neha
$ sed -n '$=' g1.txt
12
```

4. Modify:

There are different purpose of this command.
it allows you to insert, append, change or delete lines.
They do not modify just a part of a line that means they work on entire line.

This command has different options.

- **Insert command (i) :** it is denoted by character i. It is insert one or more lines directly to the output before the address lines.

```
Administrator@ADMIN ~/neha
$ cat > sed1
aaa
bbb
ccc
ddd
eee
fff

Administrator@ADMIN ~/neha
$ sed '2i\hhh' sed1
aaa
hhh
bbb
ccc
ddd
eee
fff

Administrator@ADMIN ~/neha
$ sed '1i\hhh' sed1
hhh
aaa
bbb
ccc
ddd
eee
fff
```

- **append command (a) :** it is denoted by character **a**.
  It is similar to insert command except that it writes the text directly to the output after the specified line.

```
$ cat  sed1
aaa
bbb
ccc
ddd
eee
fff

Administrator@ADMIN ~/neha
$ sed '1a\&&&' sed1
aaa
&&&
bbb
ccc
ddd
eee
fff
```

```
Administrator@ADMIN ~/neha
$ sed 'a\&&&' sed1
aaa
&&&
bbb
&&&
ccc
&&&
ddd
&&&
eee
&&&
fff
&&&
```

```
Administrator@ADMIN ~/neha
$ sed 'a\&&&' sed1 > sed2

Administrator@ADMIN ~/neha
$ cat sed2
aaa
&&&
bbb
&&&
ccc
&&&
ddd
&&&
eee
&&&
fff
&&&
```

- **change command (c) :** it is denoted by character c.
  It replaces address/matched line with new text.

```
Administrator@ADMIN ~/neha
$ cat sed1
aaa
bbb
ccc
ddd
eee
fff

Administrator@ADMIN ~/neha
$ sed '1c\unix' sed1
unix
bbb
ccc
ddd
eee
fff
```

```
Administrator@ADMIN ~/neha
$ sed -e '1i\
> hi' -e '3a\
> hello' sed1
hi
aaa
bbb
ccc
hello
ddd
eee
fff
```

- **delete command (d) :** it is denoted by character d.

```
Administrator@ADMIN ~/neha
$ sed '/bbb/d' sed1
aaa
ccc
ddd
eee
fff
```

5. Files:

File command is used to read or write data from other file respectively.
There are two types of commands :

- **read file :** it is denoted by **r fname.** When a user wants to insert common content of a file after specified line of an input file then this command is useful.
  It reads text from file **fname** and place its content after a specified line of input file.

```
$ cat test.txt
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.

Administrator@ADMIN ~/neha
$ sed 'r names' test.txt
cat COMMAND for file oriented operations.
kush
nirav
vidhi
kavya
jenil
cp command for copy files or directories.
kush
nirav
vidhi
kavya
jenil
ls command to list out files and directories with its attributes.
kush
nirav
vidhi
kavya
jenil

Administrator@ADMIN ~/neha
$ sed '1r names' test.txt
cat COMMAND for file oriented operations.
kush
nirav
vidhi
kavya
jenil
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
```

```
Administrator@ADMIN ~/neha
$ sed '/kavya/r test.txt' names
kush
nirav
vidhi
kavya
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
jenil
```

```
Administrator@ADMIN ~/neha
$ sed '1r test.txt' names
kush
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
nirav
vidhi
kavya
jenil
```

```
Administrator@ADMIN ~/neha
$ sed '1 !r test.txt' names
kush
nirav
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
vidhi
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
kavya
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
jenil
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
```

- **Write file :** it is denoted by **w fname.**
  The write file command makes possible to write the selected lines in a separate file.
  To write selected lines of input file then command id as below:
  Here in output it writes lines from names file to names.out file having pattern 'kavya'.

```
Administrator@ADMIN ~/neha
$ ls
emp.txt  f2.txt  f4.txt  f6.txt  g1.txt  names      number      sed1  stud.dat
err_msg  f3.txt  f5.out  f7.txt  n1      nohup.out  number.lnk  sed2  test.txt

Administrator@ADMIN ~/neha
$ sed -n '/kavya/w names.out' names

Administrator@ADMIN ~/neha
$ cat names.out
kavya
```

  similarly if you want to write top 5 lines from input file to the output file command is as below:

  $ sed '1,5w f4.txt' names.out

  A user can create multiple output files that contain selected lines of input file.

  $ sed -n '/linux/w file <enter>
  > /unix/w ufile' f1

OR

$ sed -ne '/linux/w file' -e '/unix/w ufile' f1

It will writes lines that contain pattern linux to a file and lines that contain unix pattern from ufile to a f1.

6. Substitute commands: (s)

It is denoted by character S.

It scan lines for search pattern and substitution it with replacement string.

This command is similar to the search and replacement feature of text editor.

This feature provides us to add, delete or change text in one or more lines.

The format of the substitution command is as follow:

[address or scanned_pattern] s/search_pattern/replace_string/[flags(s)]

Here if address is not specified, the substitution will be performs for all lines containing first occurrence of search_pattern may be regular expression or literal string.

Both search_pattern and replace_string are delimited by slash(/).

The replace_string is a string that consist of either ordinary character or an atom or meta-characters or combination of them.

Example: To replace 1st occurrences of "command" with '###' command is as below:

```
Administrator@ADMIN ~/neha
$ cat test.txt
cat COMMAND for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.

Administrator@ADMIN ~/neha
$ sed 's/command/###/' test.txt
cat COMMAND for file oriented operations.
cp ### for copy files or directories.
ls ### to list out files and directories with its attributes.
```

```
Administrator@ADMIN ~/neha
$ sed -e '1,3s/files/@@@/' test.txt
cat COMMAND for file oriented operations.
cp command for copy @@@ or directories.
ls command to list out @@@ and directories with its attributes.
```

Flag(g) : To replace all occurrence user need to use global (g) flag at the end of the instruction. This referred as global substitution.

Example : **$sed 's/for/###/g' test.txt**

Remembered Pattern :

- **USING MULTIPLE INSTRUCTIONS(-e and –f)**

  - **Sed -n –e '1,2p' -e '7,9p' -e '$p' f1** # Giving address action from a file to sed.
  - **$ cat instr.fil**
    1,2p
    7,9p
    $p
  - $ sed -n -f instr.fil f1
  - $ sed -n -f instr.fil1 -f instr.fil2 f1
  - $ sed -n -e '/saxena/p' -f instr.fill -f instr.fil2 emp?.lst

- **CONTEXT ADDRESSING**
  - $ sed –n '/director/p' f1
  - $ sed -n '/dasgupta/,/saksena/p' f1
  - $ sed -n '1,/dasgupta/p' f1
  - $ sed -n '/[aA]gg*[ar][ar]wal/p' f1
  - $ sed -n '/dasgupta/p > /sales/p' f1
    2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
    1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
    1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
    1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
    2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
  - $ sed –n '/50…..$/p' f1

- **WRITING SELECTED LINES TO A FILE:**
  - W (write) command is used to write the selected lines to a separate file.
  - Without –n selected lines will be written to the respective file; -n is used to suppress printing of all lines on the terminal.
  - $ sed -n '/director/w dlist' f1
  - $ sed –n '/director/w dlist
    /manager/w mlist
    /executive/w elist' f1
  - $ sed -n '1,55w f1
    501,$w F2' f.main

- **TEXT EDITING**
  - Here, we have some editing commands available in sed's action component.
  - **Inserting and changing lines (i , a, c)**
    - $ sed '1i\
      > #include <stdio.h>\
      > #include<unistd.h>
      > ' foo.c> $$
    - $ mv $$ foo.c; head –2 foo.c

#include <stdio.h>
#include<unistd.h>

- $ sed ‘a\
‘ emp.lst

TELNET VERSION:

- $ cat f3
  nirzari pts/1 Aug 26 02:29 (192.168.0.64)
- $ sed '1i\
  > hello user\
  > hiii
  > ' f3
  hello user
  hiii
  nnn pts/1 Aug 26 02:29 (192.168.0.64)
- $ cat f3
  nnn pts/1 Aug 26 02:29 (192.168.0.64)
- $ sed '1i\
  hello user\
  hiii
  ' f3>$$
- $ cat $$
  hello user
  hiii
  nnn pts/1 Aug 26 02:29 (192.168.0.64)
- $ sed '1i\
  > hiiii\
  > be ok
  > ' f2 >$$
- $ cat $$
  hiiii
  be ok
  sales

➢ **DELETING LINES(D)**

- $ sed ‘/director/d’ f1 > olist -n option not to be used with d

TELNET VERSION:

- $ sed '/director/d' f1
  2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
  1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
  2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
  2567 | anju agarwal | accountant | purchase | 12/7/76|2000
- $ cat f1
  2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000

1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76|2000

- $ sed -n '/director/!p' f1 > olist
- $ sed -n '/director/!p' f1
  2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
  1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
  2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
  2567 | anju agarwal | accountant | purchase | 12/7/76|2000

- **DELETING BLANK LINES**
  - $ sed '/^[]*$/d' f1 # a space and a tab inside []

- **<u>SUBSTITUTION (S)</u>**
  - [ADDRESS]s / EXPRESSION1 / EXPRESSION2 / FLAGS
  - $ cat > j3
    a b c d
    a b c
    a b
    a
  - $ sed 's/a/A/' j3
    A b c d
    A b c
    A b
    A
  - $ cat j3
    a b c d
    a b c
    a b
    a
  - $ sed 's/a/A/g' j3
    A b c d
    A b c
    A b
    A
  - $ sed 's/|/:/' f1 | head -2
  - $ sed 's/|/:/g' f1 | head -2

<u>TELNET VERSION:</u>

- **$ sed 's/|/:/' f1 | head -2**
  2233 : a.k. shukla | g.m | sales | 12/12/52 | 6000
  1006 : chanchal singhvi | director | sales | 13/9/87 | 6700
- **$ sed 's/|/:/g' f1 | head -2**

2233 : a.k. shukla : g.m : sales : 12/12/52 : 6000
1006 : chanchal singhvi : director : sales : 13/9/87 : 6700

- **$ sed '1,3s/|/:/g' f1 first 3 lines only**
- **$ sed '1,5s/director/member /' f1**
- **$ sed 's/[Aa]gg*[ar][ar]wal/Agarwal/g' f1**
- **$ sed 's/^/2/' f1| head –n 1**
- **$ sed 's/$/.00/' f1 | head –n 1**

TELNET VERSION:

- **$ sed 's/^/2/' f1**

22233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
21006 | chanchal singhvi | director | sales | 13/9/87 | 6700
21265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
22476 | anil Agarwal | manager | sales | 12/7/56 | 5000
22567 | anju agarwal | accountant | purchase | 12/7/76 |2000

- $ cat f1

2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76 |2000

- **$ sed 's/$/.00/' f1**

2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000.00
1006 | chanchal singhvi | director | sales | 13/9/87 | 6700.00
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600.00
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000.00
2567 | anju agarwal | accountant | purchase | 12/7/76 |2000.00

- **PERFORMING MULTIPLE SUBSTITUTION**
  - **$ sed 's/<I>/<EM>/g**
    > s/<B>//g
    > s/<U>/<EM>/g' form.html
  - **$ sed 's/<I>/<EM>g**
    > s/<EM>/<STRONG>g' form.html
- **COMPRESSING MULTIPLE SPACES:**
  - **$ Sed 'S/ *|/|/g' emp.lst | tee empn.lst | head –n 3**

- **THE REMEMBERED PATTERN(//)**
  1) $ sed 's/director/member/' f1
  2) $ sed ' /director/s//member/' f1
  - **$ sed '/director/s//member/' f1**

2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
1006 | chanchal singhvi | member | sales | 13/9/87 | 6700
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600

2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76|2000

- **$ sed ' /director/s/director/member/' f1**
- **$ sed 's/|//g' f1 removes every | from file**

TELNET VERSION:

- **$ sed 's/|//g' f1**

2233 a.k. shukla g.m sales 12/12/52 6000
1006 chanchal singhvi director sales 13/9/87 6700
1265 s.n. dasgupta manager sales 12/8/67 5600
2476 anil Agarwal manager sales 12/7/56 5000
2567 anju agarwal accountant purchase 12/7/76 2000

- **$ sed –n ' /marketing/s/director/member /p' f1**

NOTE: The significance of // depends on its position in the instruction. If it is in the source string, it implies that the scanned pattern is stored there. If the target string is //, it means that the source pattern is to be removed.

- **BASIC REGULAR EXPRESSION REVISITED:**
  - 3 types of expressions:
    1) The Repeated Pattern → This uses a single, &, to make the entire source pattern appear at the destination also.
    2) The Interval Regular Expression (IRE) → This expression uses the characters { and } with a single pair of numbers between them.
    3) The Tagged Regular Expression (TRE) → This expression groups pattern with (and) and represents them at the destination with numbered tags.

THE REPEATED PATTERN (&)

- **$ sed 's/director/executive director/' f1**
- **$ sed 's/director/executive &/' f1**
- **$sed '/director/s//executive &/' f1**

TELNET VERSION:

- **$ sed 's/director/executive director/' f1**

2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
1006 | chanchal singhvi | executive director | sales |13/9/87 | 6700
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76|2000

- **$ sed 's/director/executive &/' f1**

2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000

1006 | chanchal singhvi | executive director | sales | 13/9/87 | 6700
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76 |2000

- **$ sed '/director/s//executive &/' f1**

2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
1006 | chanchal singhvi | executive director | sales | 13/9/87 | 6700
1265 | s.n. dasgupta | manager | sales | 12/8/67 | 5600
2476 | anil Agarwal | manager | sales | 12/7/56 | 5000
2567 | anju agarwal | accountant | purchase | 12/7/76|2000

INTERVAL REGULAR EXPRESSION (IRE)

1) **ch\{m\}** → The meta-character ch can occur m times
2) **ch\{m,n\}** → Here, ch can occur between m and n times
3) **ch\{m,\}** → Here, ch can occur at least m times

- These are used to mention no of character/character set reputation info. Note that interval regular expression and extended reg require -E option with grep
- **Note:** In order to use this set of regular expressions you have to us -E with grep command and -r option with sed commands.
- **{n} → n occurrence of previous character**
- **{n,m} → n to m times occurrence of previous character**
- **{m, } → m or more occurrence of previous character.**

TELNET VERSION:

- **$ ls -l | grep -E 't{3}'**

-rw-rw-r-- 1 nirzari nirzari 5 Sep 3 06:51 ttt

- **$ ls -l | grep 't\{3\}'**

-rw-rw-r-- 1 nirzari nirzari 5 Sep 3 06:51 ttt

- **$ ls -l | grep -E 't\{3\}'**
- **$ ls -l | grep 't{3}'**

Example 1:

Find all the file names which contain “t” and t repeats for 3 times consecutively.

$ ls -l | grep -E ‘t{3}’

-E option is used to extend regexp understanding for grep.

Example 2:

Find all the file names which contain l letter in filename with 1 occurrence to 3 occurrences consecutively.

- $ ls -l | grep -E ‘l{1,3}’
- $ ls | grep -E 'l{1,3}'

file2
l1

ll
olist

- $ ls | grep -E l{1,3}
  grep: l3: No such file or directory

Example 3:

Find all the file names which contains k letter 5 and more in a file name.

$ ls -l | grep -E 'k{5,}'

This is bit tricky, let me explain this. Actually we had given a range i.e 5 to infinity (Just given only comma after 5).

1) # to select lines that contains mobile numbers, from file teledir.txt
   $ grep '[0-9]\{10\}' teledir.txt
2) To have list of file that have the write bit set for either for group or others:
   $ ls –l | sed –n '/^.\{5,8\}w/p'
3) $ sed –n '/.\{101,\}/p' f1 # line length at least 101
4) $ grep '^.\{101,150\}$' f2 # line length between 101 and 150

THE TAGGED REGULAR EXPRESSION (TRE)

- # the name like amit Sharma will be substituted as Sharma amit
- $ sed 's/\([a-z]*\) *\([a-z]*\)/\2, \1/' teledir.txt | sort
- $ sed 's/\(m\)\(a\)/\2\1/' f1
  2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
  1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
  1265 | s.n. dasgupta | amnager | sales | 12/8/67 | 5600
  2476 | anil Agarwal | amnager | sales | 12/7/56 | 5000
  2567 | anju agarwal | accountant | purchase | 12/7/76 |2000
- $ sed 's/\(m\)\([a-z]*\)/\2\1/' f1
  2233 | a.k. shukla | g.m | sales | 12/12/52 | 6000
  1006 | chanchal singhvi | director | sales | 13/9/87 | 6700
  1265 | s.n. dasgupta | anagerm | sales | 12/8/67 | 5600
  2476 | anil Agarwal | anagerm | sales | 12/7/56 | 5000
  2567 | anju agarwal | accountant | purchase | 12/7/76 |2000