

The background features abstract, overlapping green geometric shapes in various shades of green, primarily located on the left and right sides of the slide. The central area is white.

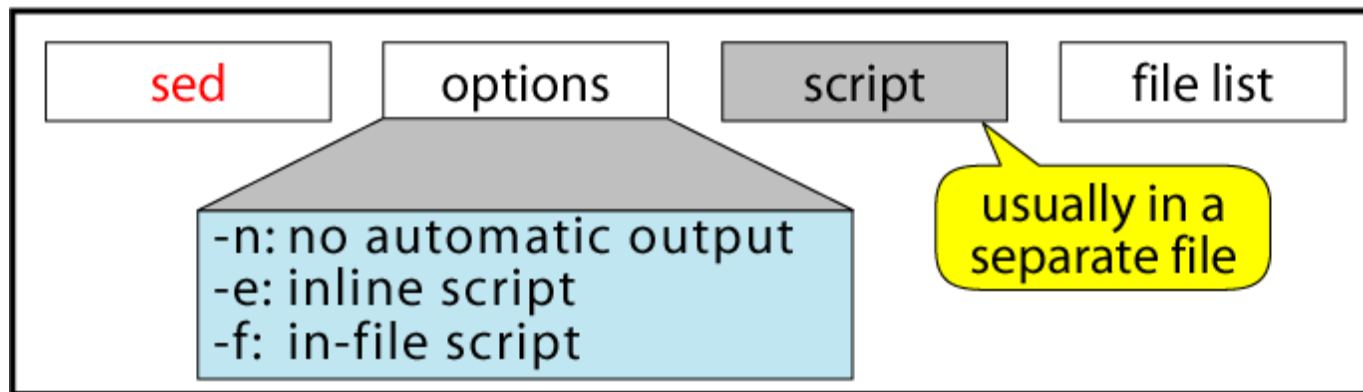
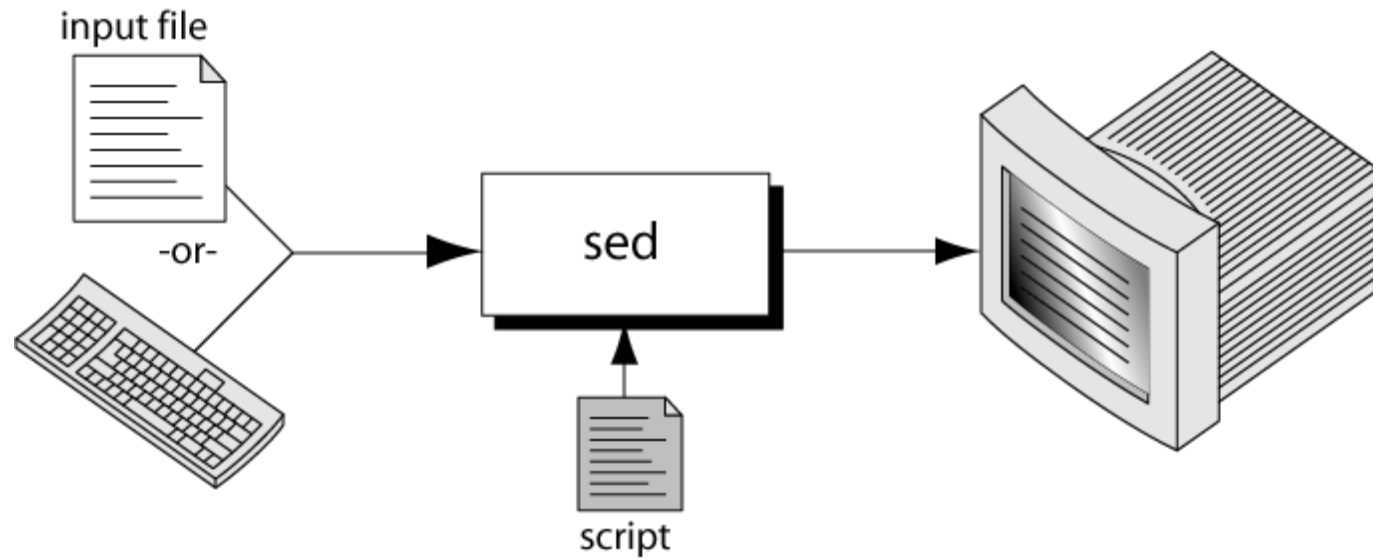
# SED

Amit Patel

# Introduction

- ▶ Full Form: Stream Editor
- ▶ Designed by Lee McMohan
- ▶ Derived from the `ed` line editor , original editor of UNIX.
- ▶ It is Not a true editor, Means does not change anything in a original file.
- ▶ Rather it scans the input file, line by line and apply a list of instruction (called Sed Script) to each line in the input file.
- ▶ The script, which is generally separate file, cannot be included in the sed command line if it is one line command.
- ▶ It is multipurpose tool which combines the work of several filters.
- ▶ Filters behaviour is governed by the options we use; but sed has very few options and its power is derived from the ease with which we can select lines and instruction to act on the selected lines.

# SYNTAX



# Script

- ▶ The sed utility is called like any other utility.
- ▶ In addition to the input data, sed require one or more instruction that provide editing criteria.
- ▶ When there is only one command, it may be entered from keyboard.
- ▶ Sometime instructions are placed in a file known as SED SCRIPT.
- ▶ Each instruction in sed script contain an address and a command.

# Script Format

## ► Inline Script format

- When the script fits in a row, its instruction can be included in the command.

```
sed -e 'address command' input_file
```

- Here “ address command” is known as sed instruction.

## ► In file Script

- For longer script or executed repeatedly then can be placed in script file

```
sed -f script.sed input_file
```

- The file is created using text editor and saved. We suffix filename with “.sed” to indicate that it is sed script.

# Instruction Format

- ▶ Each instruction consists of an address and a command

|         |   |         |
|---------|---|---------|
| Address | ! | Command |
|---------|---|---------|

- ▶ The address selects the line to be processed by the command.
- ▶ The ! is an **optional address complement**. When it is not present, the address must exactly match a line to select the line.
- ▶ When the complement operator is present, any line that does not match the address is selected

For example: 2,14 s/a/b

30 ! d

| COMMAND | DESCRIPTION                       |
|---------|-----------------------------------|
| i       | Insert                            |
| a       | Append                            |
| c       | Change                            |
| d       | Delete                            |
| N q     | Quit after reading first 10 lines |
| p       | Print                             |
| n       | Suppress Print . Use with -p      |

# Comment

- ▶ A comment is a script line that document or explain one or more instruction in a script.
- ▶ It is helpful (assist) the reader and ignored by sed.
- ▶ Begin with a command token, which is pound sign (#).
- ▶ If comment is multiline, each line start with comment token.
- ▶ For example

```
# This is a sample script
```

```
2,14 s/a/b
```

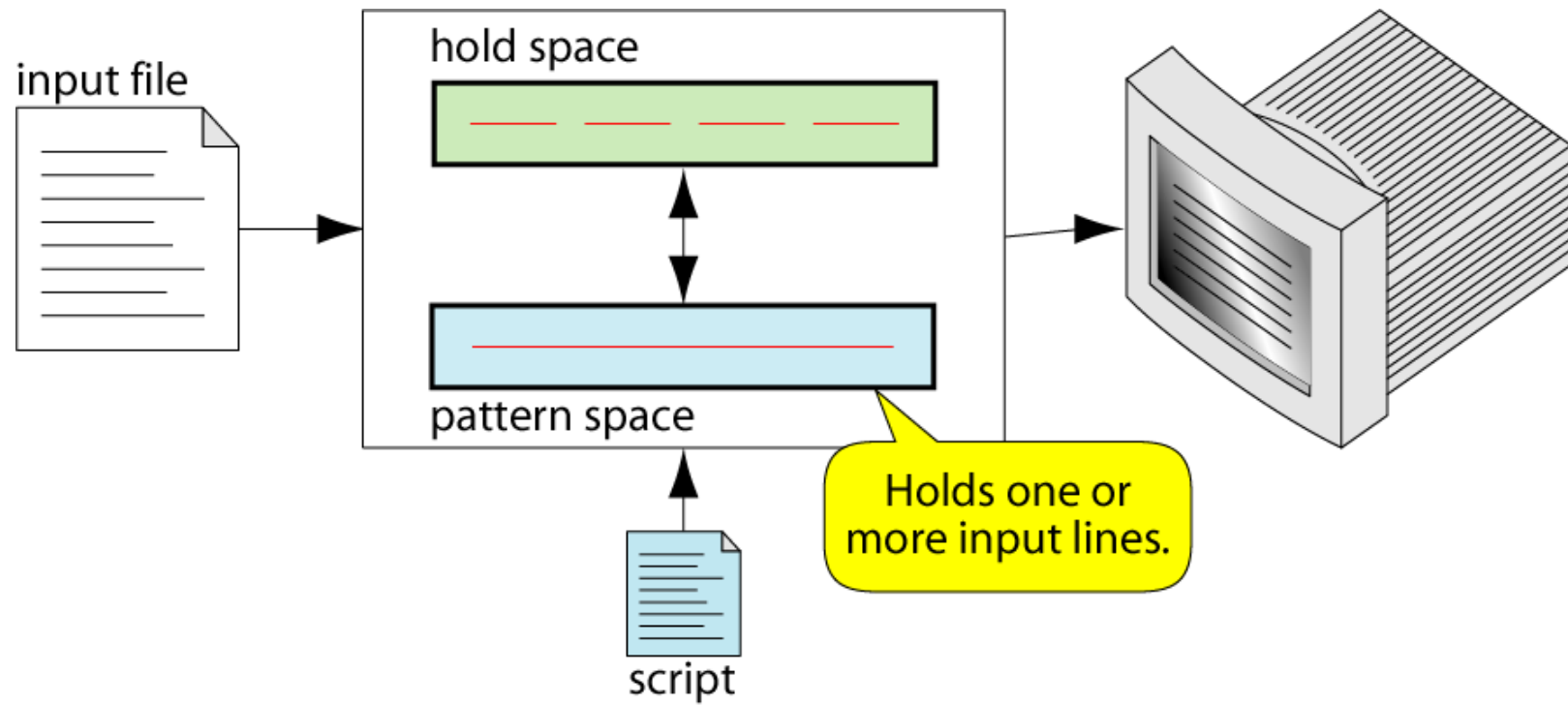
```
30d
```

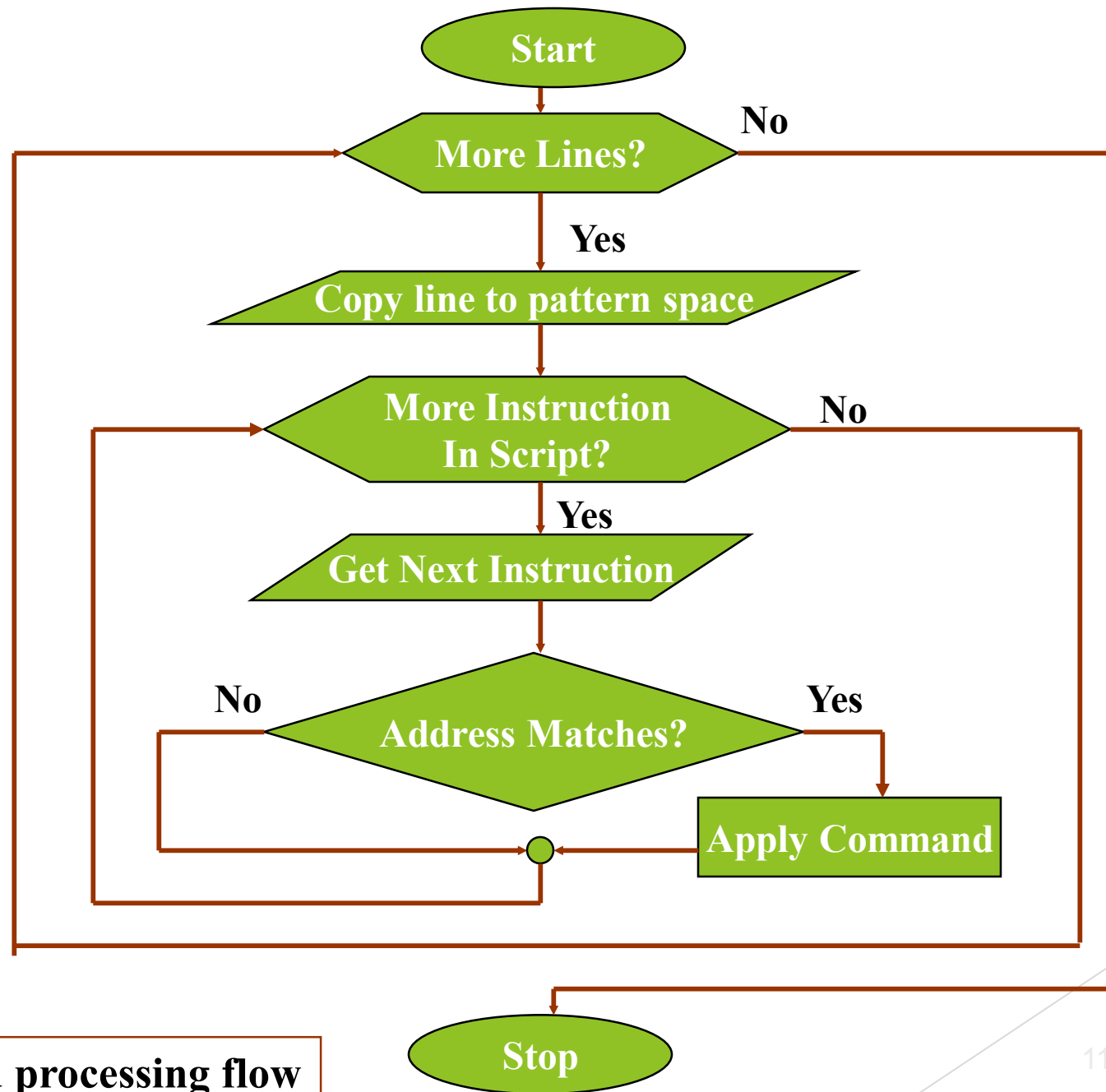
```
40d
```



# Operation

- ▶ Sed does not change the input file. All modified output is written to standard output and to be saved must be redirected to a file.
- ▶ Sed read line of input
- ▶ For each line sed performs following operations
  - ▶ Copies an input line to the pattern space. Pattern space buffer capable of holding one or more text lines for processing
  - ▶ Applies all the instructions in the script, one by one, to all pattern space lines that match the specified address in the instruction.
  - ▶ Copies the contents of the pattern space to standard output.





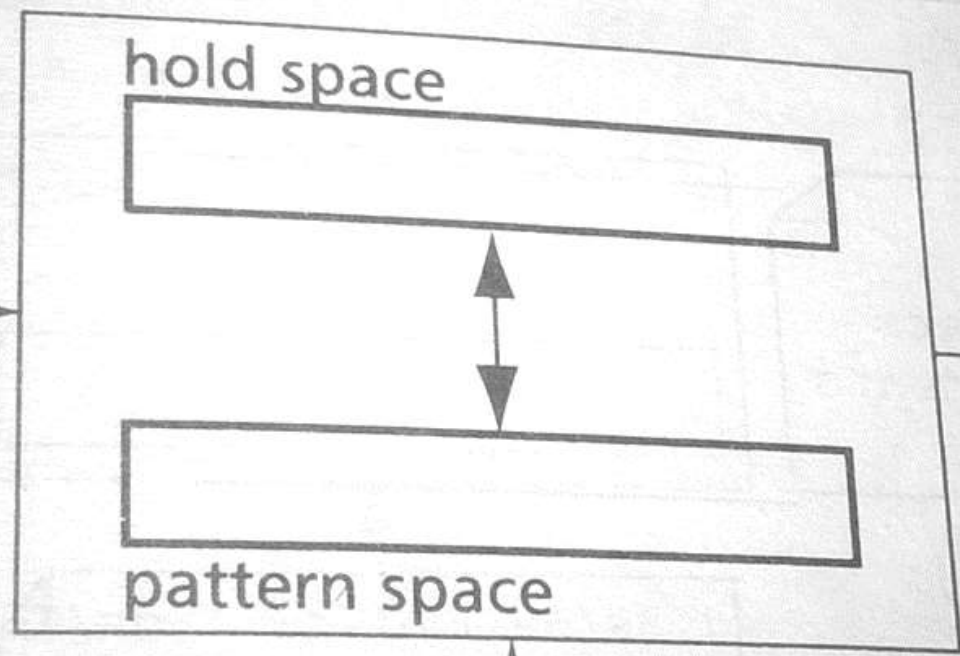
Sed processing flow

# Operation

- ▶ There are two loop in this processing cycle.
- ▶ One loop processes all of the instruction against the current line.
- ▶ The second loop processes all lines.

Hello friends  
Hello guests  
Hello students  
Welcome

hello.dat



1,3s/Hello/Greetings/  
2,3s/friends/buddies/

hello.sed

```
$ sed -f hello.sed hello.dat
```



hello.dat

Hello guests  
Hello students  
Welcome

Hello Friends

Greetings Friends

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

Hello friends  
Hello students  
Welcome

Hello guests

Greetings guests

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

Hello friends  
Hello guests  
Welcome

Hello students

Greetings students

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

Hello friends  
Hello guests  
Hello students  
Welcome

Welcome

Welcome

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

1,3s/Hello/Greetings/  
2,3s/friends/buddies/

# Example

- ▶ Consider the previous slide, Where we used script , which contain instruction with range addresses with the substitution command.
- ▶ In the first line , “Greeting” is substituted for “Hello” in the range address 1 to 3 (1,3).
- ▶ The second instruction substitutes “buddies” for “friends” in lines 2 to 3.
- ▶ At the execution of script, following actions are taken.
  - ▶ The first line (“Hello Friends”) is copied to the pattern space. The script is applied, instruction by instruction to the content of pattern space.
    - ▶ Script line 1: The address ( 1-3 ) matches the input line (1), so the instruction is applied. The pattern space now contain “Greetings Friends”
    - ▶ Script line 2: The address (2-3) doesn’t match so this instruction is not applied.
    - ▶ End of script and “Greeting Friends” send to the output.

# Example

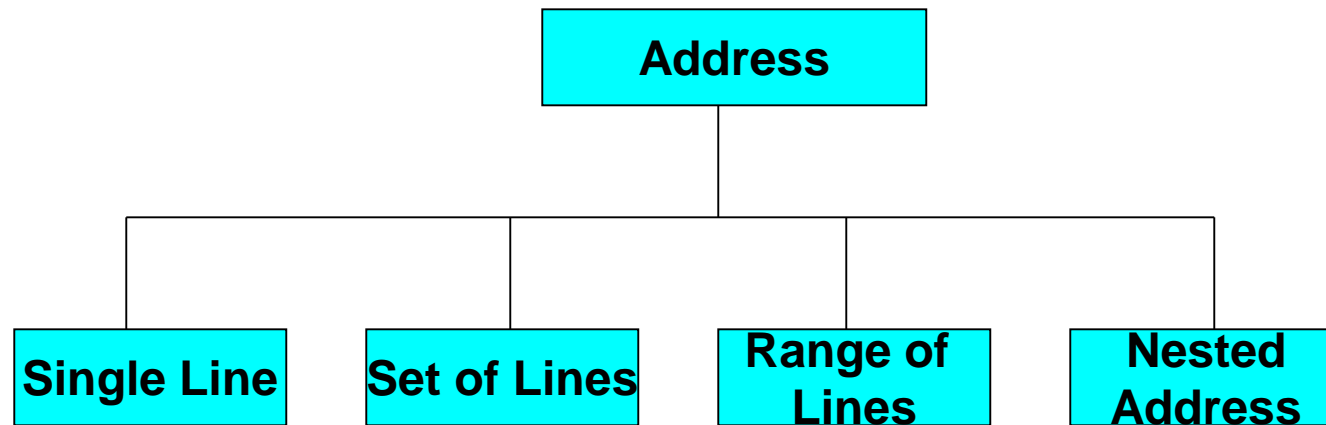
- ▶ The second line (Hello Guest) is copied to the pattern space, replacing its contents. The script is applied, instruction by instruction, to the contents of the pattern space.
  - ▶ Script line 1: The addresss (1-3) matches the input line (2), so the instruction is applied. The Pattern space now contain “Greeting Guest”
  - ▶ Script line 2: The address (2 or 3) matches the input line (2), so the instruction is applied . There is no “friend” so not match.
  - ▶ End of script: the pattern space is send lien to the monitor.

Same procedure apply to other statement.



# sed Instruction: Addresses

- ▶ The address in an instruction **determines which lines in the input file are to be processed by the commands in the instruction.**
- ▶ It can be one of 4 types.

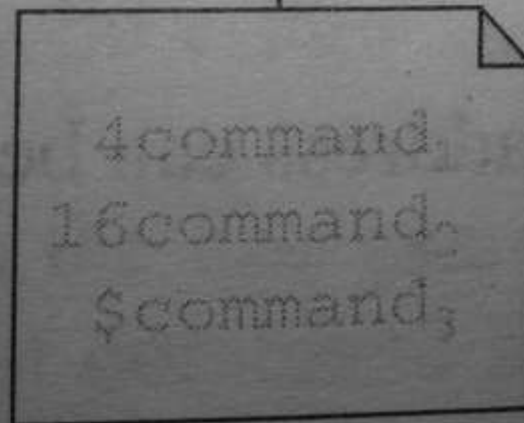
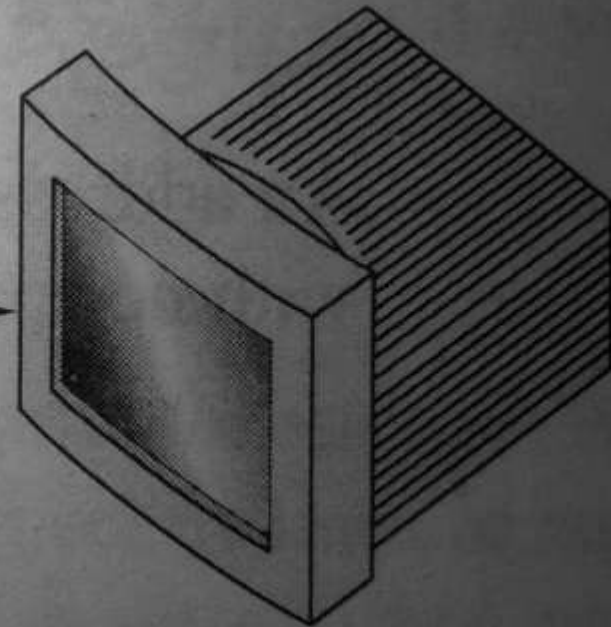
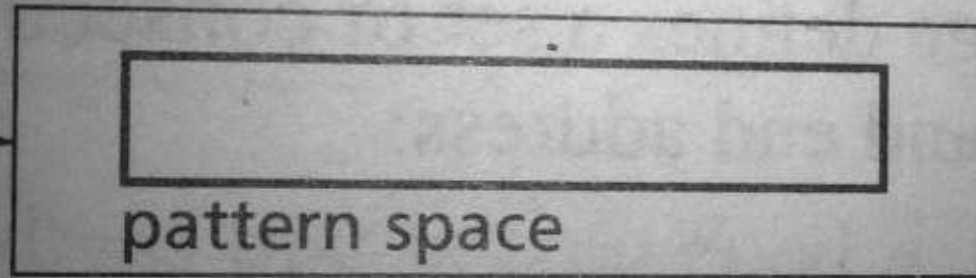
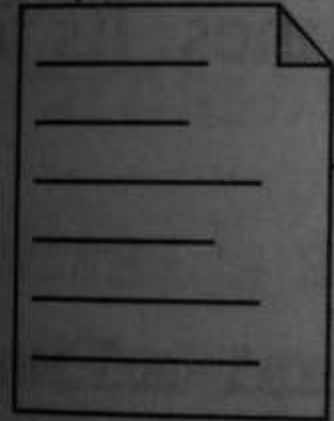


# Single Line Addresses

- ▶ A single line address specifies one and only one line in the input file.
- ▶ There are 2 formats.
  - ▶ A line number
  - ▶ A Dollar sign ( \$ )Which specifies the last line in the input file

In diagram, command1 in the first instruction applies only to line 4. In the second instruction command2 applies to line 16 and in the last instruction command3 applies to the last line only.

input file



# Single Line Addresses

- **Example 1:** Show first 3 line using sed and head command.

`sed '3 q' input-file`      **OR**      `head -3 file.`

Here address is 3 and action is “q (quit)”. It print first 3 lines and after that quit.

Same like “Head -3 file”.

To print it with -p option we have to write it like following ways

`sed '3 p' input-file`

Give wrong output.

# Limitation of -p Option

- ▶ If “p (print)” action is use with single line or line of address, it prints the line which match with address twice and all other line of file print only once.
- ▶ **For example:** Print only line 3

```
$ cat 9bca
```

```
Hi
```

```
Hello
```

```
How r you
```

```
Ok...Good Bye
```

```
Take care
```

```
Have a nice day
```

```
$ sed '3p' 9bca
```

```
Hi
```

```
Hello
```

```
How r you
```

```
How r you
```

```
Ok...Good Bye
```

```
Take care
```

```
Have a nice day
```

# Single Line Addresses

- ▶ To overcome this problem of printing duplicating lines, we should use the `-n` option whenever we use “p” command as a action

- ▶ So final output command is : `$ sed -n “3p” 9bca`

- ▶ **Example 2:** Show only last line

`sed -n -e '$ p' input-file` (-e is for inline script and its optional)

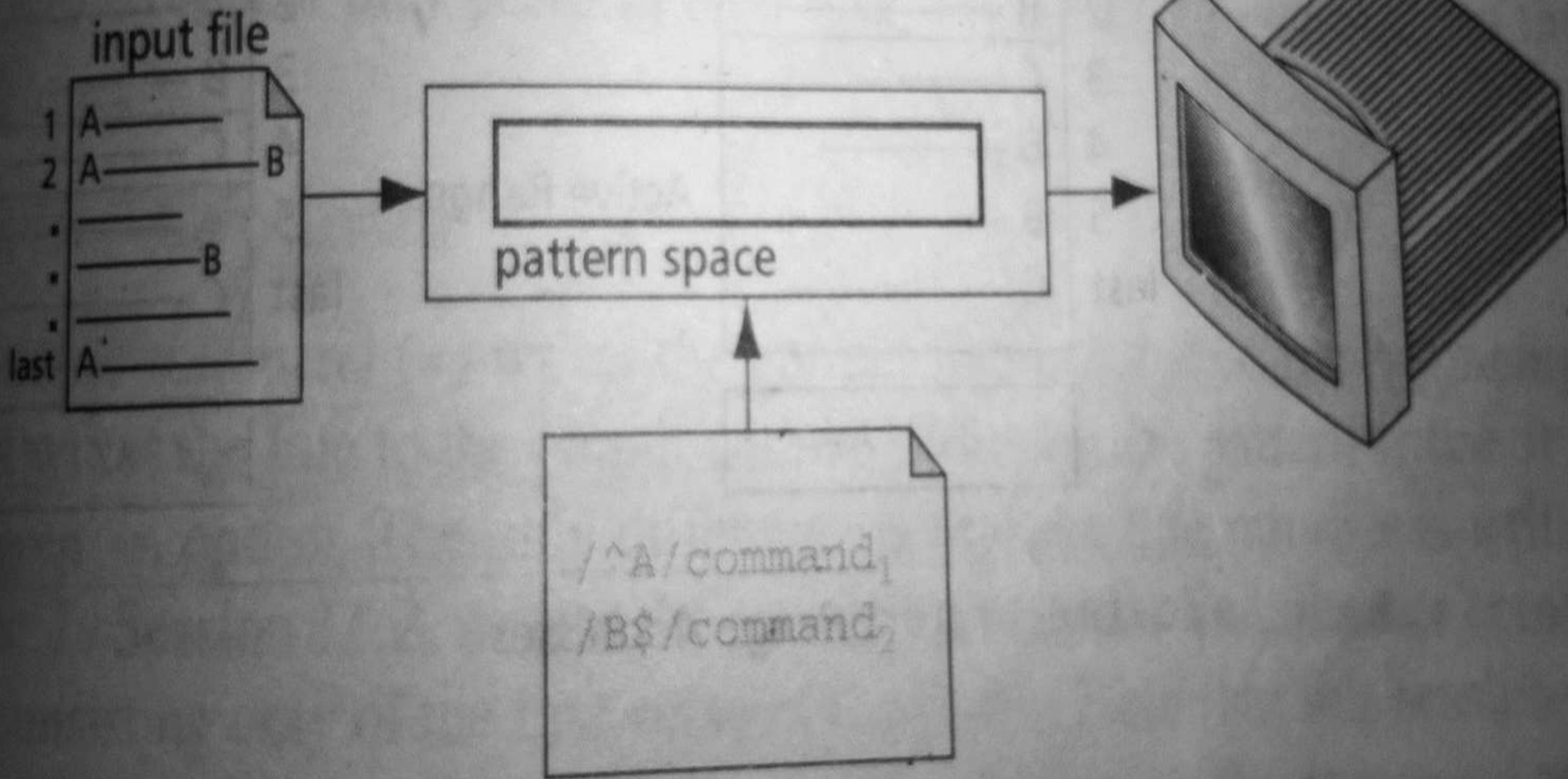
- ▶ **Example 3:** Substitute “Take care” with “TC” on line 5

`sed -e '5s/Take care/TC/' input-file`

Here “s” action define “Substitute” Follow with string we want to substitute with the replacement string.

# Set Of Line Addresses

- ▶ A set of line address is a regular expression that may **match zero or more lines**, not necessarily consecutive in the input file.
- ▶ The regular expression is written between two slashes.
- ▶ Any line in the input file that matches the regular expression is processed by the instruction command.
- ▶ Two points are important.
  - ▶ The RE may match several lines that may or may not be consecutive.
  - ▶ Even if a line matches, the instruction may not affect the line. Means substitute command in matching line may not find the data to be replaced.





# Set of Line Address

- **For example:** Print only line 1 to 3

```
$ cat 9bca
```

Hi

Hello

How r you

Ok...Good Bye

Take care

Have a nice day

```
$ sed '1,3p' 9bca
```

Hi

Hi

Hello

Hello

How r you

How r you

Ok...Good Bye

Take care

Have a nice day

# Set of Line Addresses

- ▶ To overcome this problem of printing duplicating lines, we should use the `-n` option whenever we use “p” command as a action
- ▶ So final output command is : `$ sed -n “1,3p” 9bca`

**NOTE :** Use `-n` option whenever you use the `p` command.

**Example 2:** Display First three line of file `9bca` using `sed` and `head` command.

`sed -n -e ‘1,3 p’ input-file`   **OR**   `head -3 9bca`

- ▶ We can also use negation operator (!) which can be used with any action. So we also write above command following way also

`sed -n -e ‘4!p’ 9bca`   **OR**   `sed -n ‘4,!p’ 9abc`

# Set Of Line Addresses

**Example 3:** Display lines 3 through 6 from file 9bca using sed and head command.

```
sed -n -e '3,6 p' input-file  OR  head -6 9bca | tail +3
```

- ▶ Here we can easily extract lines from the middle of the file using sed.
- ▶ It is very easy to extract line using sed then “head” and “tail” filter with “pipe”.

**Example 4:** Display lines 1 through 2 and 5 to 6 from file 9bca using sed command.

```
sed -n -e '1,2 p' -e '5,6 p' input-file
```

- ▶ “-e” option allows us to enter as many instruction as we wish, each preceded by the option.

# Note

| head & tail  | sed  |
|--|--|
| <u>Print first 3 line:</u><br><br>Head -3 9abc                           | <u>Print first 3 line</u><br><br>Sed “3,q” 9abc ( , is optional)<br><br>Sed -n “1,3p” 9abc |
| <u>Print last line</u><br><br>Tail -1 9abc                               | <u>Print last line</u><br><br>Sed -n “\$p” 9abc  |
| <u>Print line through 3 to 6 from file</u><br><br>Head -6 9abc   tail +3 | <u>Print line through 3 to 6 from file</u><br><br>Sed -ne “3,6p” 9abc                      |

# Range Address

- ▶ An address range defines a set of consecutive lines.
- ▶ **Format :**      start-address, end-address
- ▶ The start and end address can be a sed line number or a regular expression as in the next example.

Line number,line number

Line number,/regexp/

/regexp/,line number

/regexp/,/regexp/

# Range Address

- ▶ When a line that is in the pattern space matches a start range, it is selected for processing. At this point, sed notes that the instruction is in a range.
- ▶ Each input line is processed by the instruction's command until the stop address matches a line.
- ▶ The line which match the stop address is also processed by the command, but at that point, the range is no longer active.
- ▶ If at future, line the start range again matches, the range is again active until stop address.
- ▶ A special case of range address is “1,\$” , which define every lie from first line to last line (\$).

|      |   |
|------|---|
| 1    | A |
| 2    | B |
| 3    | C |
| 4    | D |
| 5    | B |
| last | C |

Active Range

3,/^A/

|      |   |
|------|---|
| 1    | A |
| 2    | A |
| 3    | B |
| 4    | C |
| 5    | A |
| last | C |

Active Range

Active Range

/^A/,/^B/

# Nested Address

- ▶ A nested address is an address that is contained within another address.
- ▶ While the outer address range, must be either a set of lines or an address range, The nested addresses may be either a single line, a set of lines or another range.
- ▶ **For example:** We want to delete all blank lines between lines 20 and 30.

```
$ sed 20,30 { /^$/d } file
```

- ▶ Here first command “20,30” specifies the range and second command which is enclosed in braces, contain the regular expression for a blank line and action to be performed on this.



# Nested Address

- ▶ **Example 2:** Delete all lines that contain the word “Bye” but only if the line also contain “Good”

```
sed '/Bye/{ /Good/d }' 9abc
```

- ▶ Here the outer address searches for line containing “Bye” while the inner line address look for line “Good”

# Inserting And Changing Text

- ▶ Sed can also use to insert new text and change text in a file.
- ▶ Sed use Insert (i), append (a) and Change (c) commands.

Example 1: Append 2 line in empnew file.

```
$ sed 'a\
```

```
> 110 | OM | MANAGER \
```

```
> 120 | SAI | DIRECTOR
```

```
> ' empnew
```

# Inserting And Changing Text

- ▶ Here first enter the instruction \$a, which append text at the end of file.
- ▶ Then enter “\” before pressing the enter key
- ▶ Each line except the last line has to be terminated by “\” before pressing the enter.
- ▶ Sed identifies the line without the “\” as the last line of input.
- ▶ **Example 2:** Insert double space between each line of file.

```
$ sed 'i\    \'emp.lst
```

# Context Addressing

- ▶ In context addressing, when we specify a single pattern, all line containing the pattern are selected.
- ▶ **For Example:** Print all line which contain “Directors” pattern.

```
$ sed -n '/Directors/p' emp.lst      OR      $ grep 'Directors' emp.lst
```

This method of addressing lines by specifying the context is known as “Context Addressing”.

We also specify a comma separated pair of context addresses to select a group of lines.

**For Example:** Print all line which contain “OM” or “SAI” pattern.

```
$ sed -n '/OM/,/SAI/p' emp.lst
```

# Context Addressing

- **For Example:** Print first line and those line which contain “Directors” pattern.

`$ sed -n '1,/Directors/p' emp.lst`      OR      `$ grep 'Directors' emp.lst`

Sed also accept regular expression as well as anchoring character “^” and “\$”.

# Command

- ▶ There are 25 command that can be used in instruction.
- ▶ All this command is divided into 9 categories.

# Line Number Command

- ▶ The line number command ( = ) write the current line number at the beginning of the line when it writes the line to the output without affecting the pattern space.
- ▶ It is similar to the grep -n option.
- ▶ The only difference is that the line number is written on a separate line.
- ▶ It is apply on set of line address or apply on all row of file . If we don't write set of line address and simply write “=” it apply on all lines of file.

# Line Number Command

- **Example 1:** Display line number before each lines.

```
$ Sed " = " 9abc
```

```
1
```

```
Hello
```

```
2
```

```
Hi
```

```
3
```

```
Good Morning
```

```
4
```

```
Bye...Take Care
```



# Line Number Command

- **Example 2:** Print only line number of lines beginning with “H”

```
$ Sed -n “/^H/ = “ 9abc
```

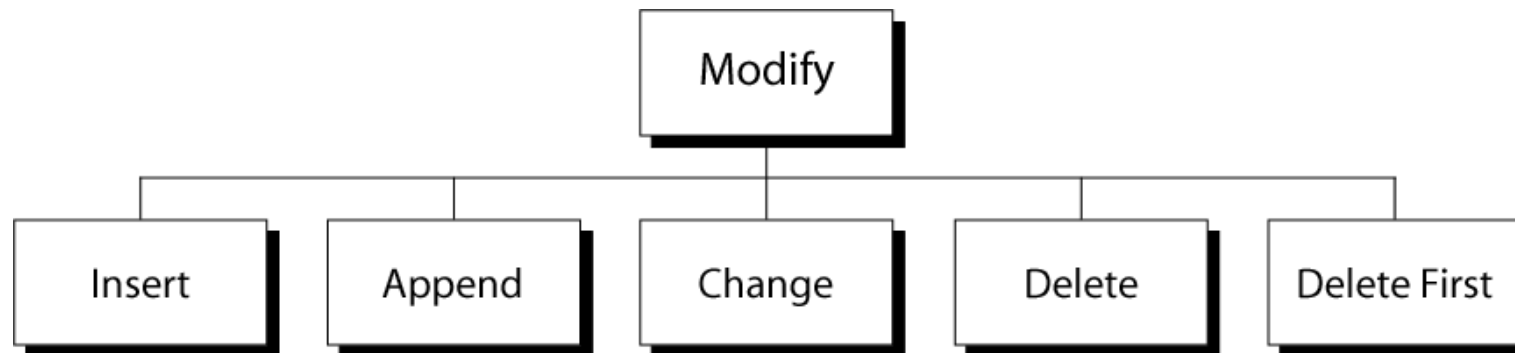
1

2

“-n” option, is used to print only line number without line.

# Modify Command

- ▶ Modify commands are used to insert, append, change or delete one or more whole lines.
- ▶ The modify commands require that any text associated with them be placed on the next line in the script. Therefore script must be in a file; it cannot be coded on the shell command line.
- ▶ It operate on whole line. In other words, they are line replacement commands.
- ▶ Whatever text you supply will completely replace any lines that match the address.
- ▶ We cannot modify just part of time.



# Insert ( i )

- ▶ Adds one or more lines directly to the output before the address.
- ▶ This command can only be used with the single line and a set of lines.
- ▶ Cannot be used with range of address.
- ▶ **Example 1:** Insert “Student Information” as a header in file

Sed “1i\ Student information” stud

Student Information

13bca01

File Name: studetails.txt

Date: 13/07/2015

File Info : Contain information about student.

# Insert ( i )

- ▶ **Example 2:** Insert blank line between each line of file.

Sed “i/ ” stud

Student Information

13bca01

File Name: studetails.txt

Date: 13/07/2015

File Info : Contain information about student.

- ▶ It add blank line between every line, even on first line and last line also. This

# Insert ( i )

Sed script to insert line

```
$ cat insert.sed
```

```
1 i\
```

```
Tuition List\
```

Input data

```
$ cat job.data
```

```
Part-time      1003.99
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

Insert “Tuition List” at 1 line

```
$ sed -f insert.sed job.data
```

```
Tuition List
```

```
Part-time      1003.99
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

# Append ( a )

- ▶ Append is similar to the insert command except that it writes the text directly to the output after the specified line.
- ▶ Like insert, append cannot be used with a range address.
- ▶ Insert and appended text never appear in sed's pattern space.
- ▶ They are written to the output before the specified line (insert) or after the specified line, even if the pattern space is not itself written.
- ▶ Because they are not inserted into pattern space they cannot match a regular expression, nor do they affect sed's internal line counter.

# Insert ( i )

Sed script to insert line

```
$ cat insert.sed
```

```
a \
```

```
-----
```

Input data

```
$ cat job.data
```

```
Part-time      1003.99
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

Append dash line after each line of file

```
$ sed -f insert.sed job.data
```

```
Part-time      1003.99
```

```
-----
```

```
Two-thirds-time 1506.49
```

```
-----
```

```
Full-time      2012.29
```

```
-----
```

# Change ( c )

- ▶ Change replace a matched line with new text.
- ▶ It accept all address type.
- ▶ **Syntax:**

```
[address1[,address2] c\  
text
```



# Change ( c )

Sed script to insert line

```
$ cat tuition.insert.sed
```

```
1 c \
```

```
Part-time      1100.00
```

Input data

```
$ cat tuition.data
```

```
Part-time      1003.99
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

Change first line of file

```
$ sed -f tuition.insert.sed tuition.data
```

```
Part-time      1100.00
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

# Delete Patten Space ( d )

- ▶ The delete command comes in two version.
- ▶ When a lowercase delete command (d) is used, it deletes the entire pattern space.
- ▶ Any script commands following the delete command that also pertain to the deleted text are ignored because the text is no longer in the pattern space.
- ▶ **Syntax:**

[address1[,address2] / d

# Delete Pattern Space ( d )

Sed script to insert line

```
$ cat tuition.insert.sed
```

```
1 d \
```

```
Part-time      1100.00
```

Input data

```
$ cat tuition.data
```

```
Part-time      1003.99
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

Remove “Part-time” date from file

```
$ sed -e '/^Part-time/d' tuition.data
```

```
Two-thirds-time 1506.49
```

```
Full-time      2012.29
```

# Substitute ( s )

- ▶ Pattern substitution is one of the most powerful commands in sed.
- ▶ Substitute replaces text that is selected by a regular expression with a replacement string.
- ▶ It is similar to the search and replace found in the text editors.
- ▶ With it, we can add, delete or change text in one or more lines.
- ▶ Sed search pattern use only subset of regular expression atoms and patterns
- ▶ **Syntax:**

```
[addr1][,addr2] s/search pattern/replacement string/[flags]
```

When a text line is selected, its text is matched to the pattern. If matching text is found, it is replaced by the replacement string.

# Search Pattern

| Atoms          | Allowed |
|----------------|---------|
| Character      | ✓       |
| Dot            | ✓       |
| Class          | ✓       |
| Anchors        | ^ \$    |
| Back Reference | ✓       |

| Operators   | Allowed      |
|-------------|--------------|
| Sequence    | ✓            |
| Repetition  | * ? \{....\} |
| Alternation | ✓            |
| Group       |              |
| Save        | ✓            |

# Pattern Matches Address

- ▶ In some case, the address contains a regular expression that is the same as the pattern we want to match.
- ▶ In the case, we don't need to repeat the regular expression in the substitute command.
- ▶ We need to omitted it, by coding two slashes at the beginning of the pattern.
- ▶ So, if search pattern is the same as the address pattern, we don't need to repeat it.

# Replace String

- ▶ The replacement text is string. Only one atom and two metacharacters can be used in the replacement string.
- ▶ The allowed replacement atom is the back references .
- ▶ The two metacharacter tokens are allowed are the **&** and the back slash (**\**).
- ▶ **&** is used to place the pattern in the replacement string.
- ▶ **\** is used to escape an ampersand when it needs to be included in the substitute text.

# Replace String

- ▶ Consider the following example.

```
$ sed 's/UNIX/** & **/' file
```

- ▶ Here the replacement string becomes “ \*\*\* UNIX \*\*\*” . Means here “ & “ becomes UNIX.

```
$ sed '/now/s//now \& forever/' file
```

- ▶ Here “&” becomes escape here so replacement string becomes “ now & forever” when sed find “now” .



# Replace String

```
$ cat datafile
```

|                   |     |     |   |    |
|-------------------|-----|-----|---|----|
| Charles Main      | 3.0 | .98 | 3 | 34 |
| Sharon Gray       | 5.3 | .97 | 5 | 23 |
| Patricia Hemenway | 4.0 | .7  | 4 | 17 |
| TB Savage         | 4.4 | .84 | 5 | 20 |
| AM Main Jr.       | 5.1 | .94 | 3 | 13 |
| Margot Weber      | 4.5 | .89 | 5 | 9  |
| Ann Stephens      | 5.7 | .94 | 5 | 13 |

```
$ sed -e 's/[0-9][0-9]$/&.5/' datafile
```

|                   |     |     |   |      |
|-------------------|-----|-----|---|------|
| Charles Main      | 3.0 | .98 | 3 | 34.5 |
| Sharon Gray       | 5.3 | .97 | 5 | 23.5 |
| Patricia Hemenway | 4.0 | .7  | 4 | 17.5 |
| TB Savage         | 4.4 | .84 | 5 | 20.5 |
| AM Main Jr.       | 5.1 | .94 | 3 | 13.5 |
| Margot Weber      | 4.5 | .89 | 5 | 9    |
| Ann Stephens      | 5.7 | .94 | 5 | 13.5 |

# Delete Part Of a Line

- ▶ To delete part of line, we leave the replacement string or text empty.
- ▶ In other words, partial line deletes are a special substitution case in which the replacement is null.
- ▶ **For example:** Delete all digits in the input from standard input.

```
$ sed 's/[0-9]//g'
```

▶ **Input:** 123abc456

321cba654

▶ **Output:** abc

cba

# Delete Part Of a Line

- ▶ Sed command work only on the first occurrence of pattern in a line.
- ▶ Here we want to delete all digits.
- ▶ Therefore we used the global flag (g) at the end of the patterns.
- ▶ If we did not use it , only the first digit on each line would be deleted.

# Change Part of Line

- ▶ To change part of line, we create a pattern that matches the part to be changed and place the new text in the replacement expression.
- ▶ **For Example:** Replace each space with tab in file.

```
$ sed 's/ / /g' file
```

Now is the time

To take a break

Output:

Now        is        the        time

To        take        a        break

# Add to Part of Line


- ▶ To add text to a line requires both a pattern to locate the text and the text that is to be added.

# Back References

- ▶ To add text to a line requires both a pattern to locate the text and the text that is to be added.
- ▶ The sed utility uses two different back reference in the substitution replacement string: whole pattern ( `&` ) and numbered buffer ( `\d` ).
- ▶ The whole pattern substitutes the deleted text into the replacement string.
- ▶ In numbered buffer replacement, whenever a regular expression matches text, the text is placed sequentially in one of the nine buffers.
- ▶ Number buffer replacement (`\d`) in which the `d` is a number between 1 and 9, substitute the numbered buffer contents in the replacement string.

# Back References

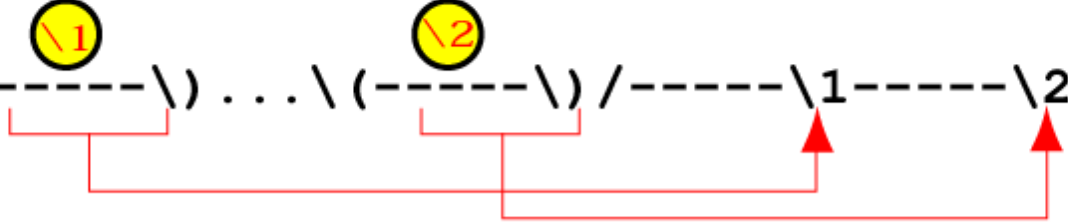
`s/-----/-----&-----/`



The diagram shows a red bracket under the first three dashes of the pattern, and a red arrow pointing from the end of the pattern to the backreference symbol (&).

(a) Whole Pattern Substitution

`s/\ (-----\1) . . . \ (-----\2) /-----\1-----\2/`



The diagram shows two yellow circles labeled \1 and \2 above the first and second subpatterns respectively. Red brackets are under each subpattern. Red arrows point from the end of the first subpattern to \1, and from the end of the second subpattern to \2.

(b) Numbered Buffer Substitution

# Whole Pattern Substitution

- ▶ When a pattern substitution command matches text in the pattern space, the matched text is automatically saved in a buffer (&)
- ▶ We can then retrieve its content and insert it anywhere and as many time as needed into the replacement string.
- ▶ Using the & buffer therefore allows us to match text, which automatically delete it and restore it so that it is not lost.



# Whole Pattern Substitution

- **Example 1:** Add space before each line of file and “—” at the end of each line.

```
$ cat 9bca
```

```
Hi
Hello
How r you
Ok...Good Bye
Take care
Have a nice day
```

```
$ sed 's/^.*$/ & --/' file
```

```
Hi--
Hello--
How r you--
Ok...Good Bye--
Take care--
Have a nice day--
```

- Here we find line any contain zero or more character using “^.\*\$” and replace it with space and the line with zero or more character, end with “--”.
- The meaning of “/ &--” means we add space, then write line as it is (denoted by &) and add “—” at end of line.

# Whole Pattern Substitution

- **Example 2:** Add “Rs.” before price of menu file using whole pattern substitution.

| \$ cat menu |    | \$ sed 's/[0-9]/Rs. &/' menu |        |
|-------------|----|------------------------------|--------|
| SAMOSA      | 20 | SAMOSA                       | Rs. 20 |
| COLD DRINKS | 10 | COLD DRINKS                  | Rs. 10 |
| DHOSA       | 50 | DHOSA                        | Rs. 50 |

- Here we look for digit in the file, whenever we find it is replace with “Rs. Follow by number”.

# Numbered Buffer Substitution

- ▶ It use one or more of the regular expression numbered buffer.
- ▶ We use it when the pattern matches part of the input text but not all of it.
- ▶ For example, In our Bike number it is divided into 3 parts. First part contain state info, second part contain code of division, and last 4 digit contain number of bike

\$ cat bike

|       |          |
|-------|----------|
| OM    | GJ51920  |
| SAI   | RJ100620 |
| DHOSA | MH096587 |

\$ sed 's/ \([a-zA-Z]\{2\}\)\([0-9]\{1,2\}\)\([0-9]\{1,4\}\)/\1-\2-\3/' bike

|       |            |
|-------|------------|
| OM    | GJ-5-1920  |
| SAI   | RJ-10-0620 |
| DHOSA | MH-09-6587 |

# Numbered Buffer Substitution

## ► Example:

```
$ sed 's/first-class/class-first/' stud.dat
```

Can be written as

```
$ sed 's/ \1 - \2 - \1/' stud.dat
```

# Substitute Flag

- ▶ There are four flag that can be added at the end of substitute command to modify its behavior.
  - ▶ Global substitution (g)
  - ▶ Specific occurrence substitution (digit)
  - ▶ Print (p)
  - ▶ Write file (w)

# Global Flag

- ▶ The substitution command only replaces the first occurrence of a pattern.
- ▶ If there are multiple occurrences, none after the first are changed.
- ▶ For example: To replace “cat” with “dog” in file

| Without Global Flag  | With Global Flag  |
|--|---|
| <pre>\$ cat file</pre><br>Ram had a black cat & white cat<br><br><pre>\$ sed 's/cat/dog/' file</pre><br>Ram had a black dog & white cat. | <pre>\$ cat file</pre><br>Ram had a black cat & white cat<br><br><pre>\$ sed 's/cat/dog/g' file</pre><br>Ram had a black dog & white dog. |

# Specific Occurance Flag

- ▶ Specific occurrence changes any single occurrence of text that matches the pattern.
- ▶ The digit indicates which one to change.
- ▶ Here in following example we want to change second occurrence of cat.

## Without Global Flag

```
$ cat file
```

```
Ram had a black cat & white cat and white cat.
```

```
$ sed 's/cat/dog/2' file
```

```
Ram had a black dog & white dog and white cat.
```

# Transform(y)

- ▶ If you wanted to change a word from lower case to upper case, you could write 26 character substitutions, converting "a" to "A," etc.
- ▶ Sed has a command that operates like the tr program.
- ▶ It is called the "y" command.
- ▶ For instance, to change the letters "a" through "f" into their upper case form, use:

```
sed 'y/abcdef/ABCDEF/' file
```



# Transform(y)

- If you wanted to convert a line that contained a hexadecimal number (e.g. 0x1aff) to upper case (0x1AFF), you could use:

```
sed '/0x[0-9a-zA-Z]*/ y/abcdef/ABCDEF' file
```

# Read(r)

- There is also a command for reading files. The command

```
sed '$r amit.txt' 9bca
```

will append the file "end" at the end of the file (address "\$)." The following will insert a file after the line with the word "INCLUDE:"

```
sed '/INCLUDE/ r file' <in >out
```

append file when

# Set of Line Address

- **For example:** Print only line 1 to 3

```
$ cat 9bca
```

Hi

Hello

How r you

Ok...Good Bye

Take care

Have a nice day

```
$ sed '1,3p' 9bca
```

Hi

Hi

Hello

Hello

How r you

How r you

Ok...Good Bye

Take care

Have a nice day

# Remembered Pattern ( / /)

- ▶ If the address specified pattern is the same as the string to be substitute, we can apply concept of Remembered pattern.
- ▶ Sed remembered the scanned pattern, and stores it in the ' / /'.
- ▶ The / / representing an empty regular expression is interpreted to mean that the search and substituted patterns are the same. We call it remembered pattern.

**For example:** In the following command

```
$ sed -n '/huge/s/huge/small/g' stud.dat
```

The address /huge/ appear to redundant , we can write it as follow

```
$ sed -n '/huge/s/ /small/g' stud.dat.
```

# Remembered Pattern ( / / )

- ▶ The two empty front slashes are for remembered pattern and effectively imply that there is a pattern in between them which is same as the one scanned i.e 'huge'
- ▶ Following statements are same and give same result.

Sed 's/director/manager/' emp.dat

Sed 'director/s/ /manager/' emp.dat

Sed 'director/s/director/manager/' emp.dat

The second form suggest that sed remembers the searching pattern 'director' and stored it in / /.

When we used it at target staring it means we are removing the pattern.

sed 's/|/ /g' emp.dat.

# Remembered Pattern ( / /)

**For example:** Find line which contain “now” and replace it with “Now & forever”.

```
$ sed '/now/s//now \& forever/' file
```

- ▶ Here “&” becomes escape here so replacement string becomes “ now & forever” when sed find “now” .

# Repeated Pattern ( & )

- ▶ When a pattern in the source string also occurs in target string, Repeated pattern is used.
- ▶ We can then use the special character “ & ” to represent it.
- ▶ For example:

```
$ sed '/huge/s/ /very huge/' stud.dat
```

- ▶ Can be written as

```
$ sed '/huge/s/ /very &/' stud.dat
```

- ▶ The only restriction is that ‘&’ can only be used to represent a complete source string.  
Not a part of it.

# Repeated pattern

**Example :** Replace the pattern “ Linux” in file with “Linux-Unix”.

- ▶ & is used to place the pattern in the replacement string.

```
$ sed -e 's/Linux/&-Unix/' thegeekstuff.txt
```

1. Linux-Unix Sysadmin, Linux Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux-Unix
5. Productivity (Too many technologies to explore, not much time available)
6. Windows- Sysadmin, reboot etc.



# Repeated Pattern

- **Example :** Add space after each line of file and “—” at the end of each line.

```
$ cat 9bca
```

```
Hi
Hello
How r you
Ok...Good Bye
Take care
Have a nice day
```

```
$ sed 's/^.*$/ & --/' file
```

```
Hi--
Hello--
How r you--
Ok...Good Bye--
Take care--
Have a nice day--
```

- Here we find line any contain zero or more character using “^.\*\$” and replace it with space and the line with zero or more character, end with “--”.
- The meaning of “/ &--” means we add space, then write line as it is (denoted by &) and add “—” at end of line.

# ENVIRONMENT VARIABLE

- ▶ UNIX operating system has defined these variables for its own use.
- ▶ When you log in on UNIX, your current shell (login shell) sets a unique working environment for you which is maintained until you log out.
- ▶ The “\$” prompt that we see is the default value of the UNIX defined variable “PS1” standing for system prompt 1.
- ▶ We can also assign different value to PS1 like

```
$ PS1 = "OM"
```

```
OM
```

- ▶ So every time the system prompt you, it display “OM” not “\$”.

# ENVIRONMENT VARIABLE ( SYSTEM VARIABLE)

- ▶ Following are most command examples of environment variables used under UNIX operating systems

| VARIABLE | MEANING  |
|----------|--|
| PS1      | Display Shell prompt in the bourne shell and variants.                   |
| PS2      | The system prompt 2, default value is “ > “                              |
| PATH     | Define the path Which shell must search in order to execute any command. |
| HOME     | Store the default working directory of the user.                         |
| LOGNAME  | Store the login name of the user.  |
| SHELL    | Define the name of your default working shell.                           |
| IFS      | Define the internal Field Separator, which is space , tab or new line.   |
| TERM     | Define the name of the terminal on which you are working.                |

# ENVIRONMENT VARIABLE ( SYSTEM VARIABLE)

| VARIABLE  | MEANING  |
|-----------|--|
| MAIL      | Defines the file where the mail of the user is stored.   |
| MAILCHECK | Define the duration after which the shell checks whether the user has received any mail. By default its values is 600 seconds. |
| TZ        | Define the name of the time zone in which we are working.  |

# NULL VARIABLE

- ▶ A variable which has been defined but has not been given any values is known as null variable.
- ▶ A null variable can be created in any way of the following.

```
$ d=""
```

```
$ d=' '
```

```
$ d=
```

- ▶ On echoing a null variable, only a blank line appears on the screen.
- ▶ If a null variable is used anywhere in a command the shell manage to ignore it .