Module 3: vi editor and Shell

vi editor:

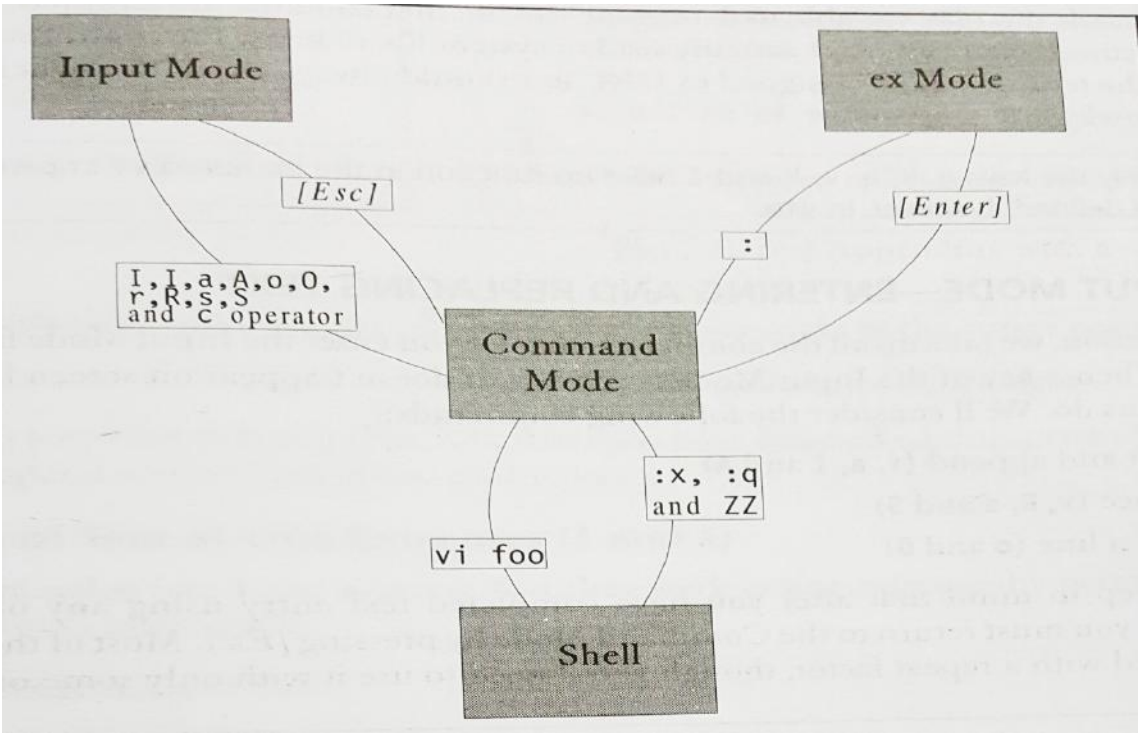
vi is an editor that uses a number of internal commands to navigate to any point in a text file and edit the text there.It also allows you to copy and move text from one file to another and within a file.

1. The .exrc file:

- vi reads the file $HOME/.exrc on startup.
- Many ex mode commands can be placed in this file, so they are available in every session.
- You can create abbreviations , redefine your keys to behave differently and also make variable settings.
- .exrc progressively develop into exclusive library containing all shortcuts, settings that you use regularly.

DIFFERENT WAYS OF INVOKING AND QUITTING VI EDITOR.

2.THREE MODES OF vi EDITOR



2.1. Command mode:

- The default mode of the editor where every key pressed is interpreted as a command to run on text.
- You have to be in this mode to copy and delete the text

2**.2 Input mode**:

- Every key pressed after switching to this mode actually shows up as text.
- This mode is invoked by pressing one of the input mode commands keys.

2.3. ex mode(last line mode):

- The mode used to handle files like saving and perform substitution. Pressing a **:** in the command mode invokes this mode.

- You then enter an ex mode command followed by [enter]. After the command is run, you are switched to default command mode.

3. Input mode: Entering and Replacing Text

Command	*Function*
i	Inserts text to left of cursor (Existing text shifted right)
a	Appends text to right of cursor (Existing text shifted right)
I	Inserts text at beginning of line (Existing text shifted right)
A	Appends text at end of line
o	Opens line below
O	Opens line above
r*ch*	Replaces single character under cursor with *ch* (No *[Esc]* required)
R	Replaces text from cursor to right (Existing text overwritten)
s	Replaces single character under cursor with any number of characters
S	Replaces entire line

- When a key of input mode is pressed, it doesnot appear on the screen.
- After you have completed text entry using any of these commands(except r), you must return to the command mode by pressing [Esc].
- Insert and Append(I,A,i,a)
- Replace(r,R,s,S)
- Open a line(o and O)

3.1 Insertion of text(i and a)

- The simplest type of input is insertion of text. Just press i.
- Pressing this key changes the mode from command to input.
- If the i command is invoked with the cursor positioned on existing text, text on its right will be shifted further without being overwritten.

3.2 Insertion of text at line extremes(I and A)

- I-inserts text at beginning of line
- A-appends text at end of line

  For example: Consider the line :

  The following command forks a process

  To convert the above line to comment line , we have to use /* at the beginning and */ at the end of the line.

  Ex 1: To insert the symbol /* at the beginning of line

  I/* [Esc]

  Ex 2:To insert the symbol */ at the end of the line

  A */ [Esc]

  Output: **/* The following command forks a process*/**

3.3 Opening a New Line(o and O)

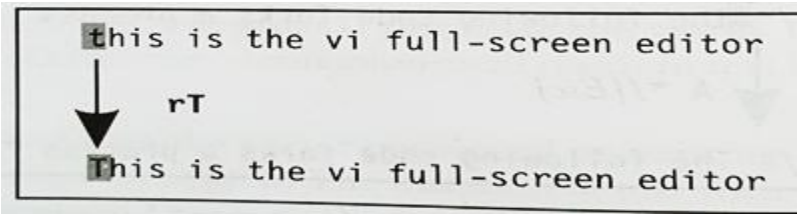
o- opens a new line below the current line

O-opens a new line above the current line

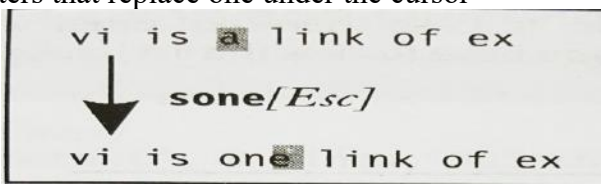
3.4 Replacing Text(r,R,s,S)

- To change existing text vi provides mainly four commands
- To replace a single character with another use

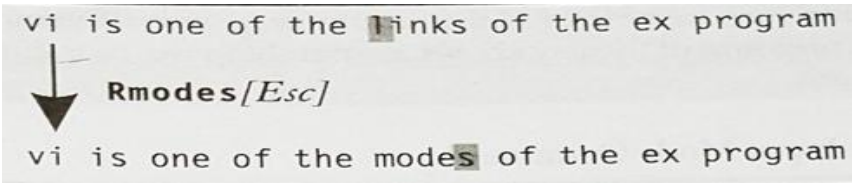
  r followed by the character that replaces the one under the cursor.



- To replace a single character by many
  s followed by the characters that replace one under the cursor



- To replace all text on the right of the cursor position.
  R



- To replaces the entire line irrespective of cursor position(Existing line disappears)
  S

4. ex Mode (saving text and quitting)

4.1 saving your work(:w)

- You must be able to save the buffer and remain in the editor. From time to time, you must use the **:w** command to write the buffer to disk.
- Enter a **:** (colon) which appears on the last line of the screen, then w and finally [Enter].

4.2 Saving and quitting(:x and :wq)

- To save and quit the editor use the **:x** command or **:wq** command .

4.3 Aborting editing(:q)

- Its possible to abort the editing process and quit the editing mode without saving the buffer. The **:q** command does the job.

4.4 Writing the selected lines:

The **: w command** is an abbreviated way of executing the ex mode instruction
w can be prefixed by one or two addresses separated by a comma.

Ex 1 **:10,50w unix** // saves lines from 10th to 50th of current file to the file unix
Ex 2 **:5w unix** //saves 5th line of current file to unix file
Ex 3 **:.w unix** //saves current line to unix file
Ex 4 **:$w unix** //saves last line to unix file
Ex 5 **: . , $ unix** //saves lines from current line to last line to unix

4.5 Escapes to the unix shell(:sh and [ctrl-z])

To make a temporary escape to the shell
Use **:sh or [ctrl-z]**

4.6 Recovering from a crash(:recover and –r)

- The power can go off leaving the work unsaved. But vi stores most of its buffer information in a hidden swap file. Even though vi removes this file on successful exit, a power glitch or improper shutdown lets this swap file remain on disk.
- vi will then complain the next time you invoke it with same file. You are advised to use **:recover** option to recover as much as possible.

SAVE and EXIT commands of the ex mode

Command	*Action*
`:w`	Saves file and remains in editing mode
`:x`	Saves file and quits editing mode
`:wq`	As above
`:w n2w.pl`	Like *Save As* ..... in Microsoft Windows
`:w! n2w.pl`	As above, but overwrites existing file
`:q`	Quits editing mode when no changes are made to file
`:q!`	Quits editing mode but after abandoning changes
`:`*n1*`,`*n2*`w build.sql`	Writes lines *n1* to *n2* to file `build.sql`
`:.w build.sql`	Writes current line to file `build.sql`
`:$w build.sql`	Writes last line to file `build.sql`
`:!`*cmd*	Runs *cmd* command and returns to Command Mode
`:sh`	Escapes to UNIX shell
`:recover`	Recovers file from a crash

5. NAVIGATION

5.1 Cursor Movement in the four directions(h,j,k,l)

h Moves Cursor left
j Moves Cursor down
k Moves cursor up
l Moves cursor right

5.2 Word Navigation:

There are 3 basic commands:

b Moves back to beginning of the word.
e Moves back to end of the word
w Moves forward to beginning of the word.

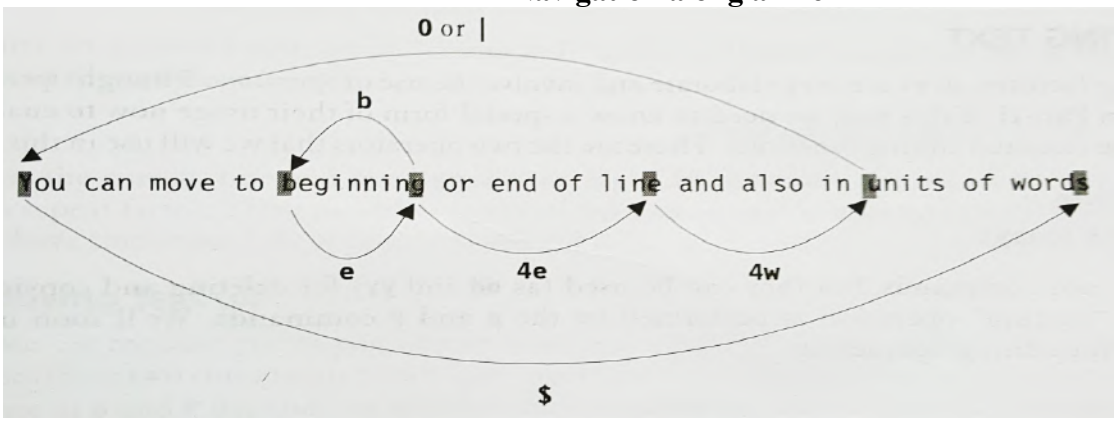
5.3 Moving to the line extremes:(0 ,|, $)

0 (zero) to move to first character of the line.
| (vertical bar) takes a repeat factor and using that you can position the cursor on a certain column.

$(dollar) Moves to end of the line

Ex 1: 30| Moves cursor to column 30

Navigation along a line



5.4 Scrolling(ctrl-f],[ctrl-d],ctrl-u])

Faster movement can be achieved by scrolling text in the window using the control keys.
The two commands for scrolling page at a time are:

[Ctrl-f]-→ Scrolls forward
[Ctrl-b]→ scrolls backward.

[Ctrl-d]→ scrolls half page forward
[Ctrl-u]→ scrolls half page backward.

5.5 Absolute Movement(G)

You need to use the G command with the line number as repeat factor to locate the offending lines.

To move to the 40th line **40G**
To move to the beginning of the line **1G**
To move to the end of the line **G**

6. EDITING TEXT:

There are two operators:

- **d delete**
- **y yank (Copy)**

6.1 DELETING TEXT(dd and x)

- The simplest text deletion is achieved by the **x command.** This command deletes the character under the cursor .Move the cursor to the character that needs to be deleted and press x. **x** deletes a single character
- Entire lines are removed with dd command.Move the cursor to any line and then press
  dd---→ deletes that line.
  7dd-→ deletes the current line and 6 lines below
  11dd-→ deletes the current line and 10 lines below.

6.2 Moving Text(p)

- The use of **p** command is to put operations that follow delete or copy operations.

6.3 Copying Text(y and p)

- vi uses the term yanking to coy the text.
- The use of yy command for yanking
  yy-→ copies current line
  4yy→ copies current line and three line below
- This yanked/copied text has to be placed in the new location. The put command **p** is used.

6.4 Joining Lines(J)

- To join the current line and the line following it use **J**
- To join following 3 lines with current line. **4J**

7. The Repeat command:

- Most editors don't have the facility to repeat the last editing instruction but vi has.
- The (.) command is used for repeating both input and command mode commands that perform editing tasks.
- The principle is this : Use the actual commandonly once and then repeat it at other places with dot command.
- The . dot command can be used to repeat only the most recent editing operation – be it insertion,deletion or any other action that modifies the buffer.
-

8. Searching for a Pattern (/ and ?)

Command	Function
/pat	Searches forward for the pattern pat
?pat	Searches backward for the pattern pat
n	Repeats search in same direction along which previous search was made
N	Repeats search in opposite direction along which previous search was made

- **v**i is extremely strong in search and replacement activities.
- Searching can be done in both forward and reverse directions.
- It is initiated from the command mode by pressing **/,** which shows up in the last line.
- Search forward:
  / (forward slash), followed by pattern to be searched
- Search backward:
  ? followed by pattern to be searched
  For example:
  /shell --→ Searches forward for the pattern **shell** in your file
  ?shell -→ searches backward for the pattern **shell** in your file.

8.1Repeating last pattern search(n and N)

The n and N commands repeat a search .
n repeats the search in same direction of original search.
N reverses the direction pursued by n

8.2 SUBSTITUTION: SEARCH and REPLACE(:s)

vi offers the powerful feature that of substitution which is achieved by the ex mode

s (substitute) command.

The syntax is

:address/source_patten/target_pattern/flags

- The soure_pattern is replaced with target_pattern in all lines specified by address.
- The address can be one or a pair of numbers separated by comma.
- The most commonly used flag is g, which carries out the substitution for all the occurrences of the pattern in a line.

Ex1.

:1,$s/director/member/g

Replaces the director by member globally throughout the file .

1,$ from 1st line to last line of the current file

Ex2:

:1,50s /director/member/g

Searches for the word director from line1 to line 50 and replaces it by word member.

Ex 3:

:1.30s/director//g

Here the target pattern is left out. Hence all the instances of director from line 1 to line 30 are deleted.

Ex4:

:.s/director/member/g

Searches for the word director and replaces it by word member in the current line

Ex 5:

:$s/director/member/g

Searches for the word director and replaces it by word member in the last line

8.2.1 Interactive Substitution:

- Sometimes you may likely to selectively replace a string .
- In this case use the **c (confirmation)** parameter as the flag at the end.

Ex:

:1,$s/director/member/gc

```
9876|jai sharma       |director |production|03/12/50|7000
                       ^^^^^^^y
2365|barun sengupta   |director |personnel |05/11/47|7800
                       ^^^^^^^n
1006|chanchal singhvi |director |sales     |09/03/38|6700
                       ^^^^^^^y
6521|lalit chowdury   |director |marketing |09/26/45|8200
                       ^^^^^^^n
```

Each line is selected in turn followed by a sequence of carets in the next line,just below the pattern that requires substitution. The cursor is positioned at the end of this caret sequence waiting for your response.

THE SHELL

1. The SHELL'S INTERPRETIVE CYCLE

When you log onto a UNIX machine , you see a prompt.This prompt remains until you key in something.Even though it may appear that the system is idling, a UNIX command is in fact running at the terminal.But this command is special its with you all the time and never terminate unless you log out. This command is **shell**.

If you provide the input in the form of ps command (that shows processes owned by you), you will see shell running

```
$ps
PID   TTY   TIME CMD
328   pts/2 0:00 bash
```

- The bash shell is running at the terminal /pts/2.
- When you key in the command it goes to shell as input.
- The shell scans the command line for metacharacters.These are the characters that mean nothing to the command but has special meaning to the shell.
- For example if the shell encounters metacharacters like *,| etc in the command line.
- If the metacharacter is *, then shell replaces it with all the filenames in the current directory.
- When all preprocessing is complete , the shell passes on the command to the kernel for ultimate execution.
- While the command is running, the shell has to wait for notice of its termination from the kernel.
- After the command is complete with its execution, then shell once again issues the prompt to take up your next command.

ACTIVITIES PERFORMED BY THE SHELL IN ITS INTERPRETIVE CYCLE.

1.The shell issues the prompt and waits for you to enter a command.

2.After a command is entered, the shell scans the command line for metacharacters and expands abbreviations (like * in rm *) to recereate the simplified command line.

3.It then passes on the command line to the kernel for execution

4.The shell waits for the command to complete and normallty cant do any work while the command is running.

5. After command execution is complete, the prompt reappers and shell returns to its waiting role to start the next cycle. Yo can now enter the next command.

2. WILD CARDS AND FILE NAME GENERATION.

2.1 WILD CARDS:

The metacharacters taht are used to construct the generalized pattern for matching filenames belong to the category called **wild cards.**

2.1.1 The * and ?

- **The metacharacter *** is one of the characters of the shell wild card set. It matches any number of characters including none.
- For example : to match filenames chap chap01 chap02 chap03 chap04
  $ ls chap*
  Output : // the * matches all strings along with none

chap
chap01
chap02
chap03
chap04

- **The metacharacter ?** matches a single character
  For example: to match filenames chapx chapy chapz
  $ ls chap?
  Output: //the ? replaces single character i,e ? is replaced by x, y,z
  chapx
  chapy
  chapz

2.1.2 Matching the dot(.)

- The **.** dot metachacter can be used to match all the hidden files in your directory.
- Example: To list all hidden files in your directory having atleast three characters after the dot.
  $ ls .???*
  Output:
  .bash_profile
  .exrc
  .netscape
  .profile

2.1.3 The character class [ ]

- This class comprises a set of charcters enclosed by the rectangular brackets [and],but it matches only a single character in the class.
- The **pattern [abcd]** is a character class and it matches a single character a, b, c or d
- For example to match chap01 chap02 chap03 chap04
  $ ls chap0[1234]
  output:
  chap01
  chap02
  chap03
  chap04

2.1.4 Negating the character class(!)

- It is used to reverse the matching criteria.
- For example:
  To match all the filenames with single character extensions but not .c or .o files
  $ ls *.[!co]
  *** to match any filename**
  . extension
  ! except
  [!co]---> .c extension or .o extension
- To match filenames that does not begin with a digit
  $ ls [!0-9]*

- To match filename with 3 character that does not begin with an Upper Case letter
  $ ls [!A-Z]??

2.2 REMOVING THE SPECIAL MEANINGS OF WILD CARDS

ESCAPING and QUOTING

Escaping: providing a \(backslash character) before the wild card to remove or escape its special meaning.

Quoting: enclosing the wild card or even the entire pattern within quotes ('chap* '). Anyting within the quotes are left alone by the shell and not interpreted.

2.2.1 ESCAPING

- Placing a \ immediately before a metacharacter turns off its special meaning.
- For instance * , matches * itself. Its special meaning of matching zero or more occurrencesof character is turned off.
- Ex 1:
  $rm chap*
  removes all the filenames starting with chap. Chap, chap01,chap02 and chap03 are removed.

  $rm chap* // * metacharacter meaning is turned off
  removes the filename with chap* /*name of the file itself is chap*.

- Ex 2:
- If there are files with names chap01, chap02,chap03.
- To list the filenames starting with chap0
  $ ls chap0[1-3]
  Output:
  chap01
  chap02
  chap03

- Ex 3:
  To match the file named as chap0[1-3]
  $ ls chap0\[1-3\]
  Output:
  chap0[1-3]

Escaping the space: To remove the file My document.doc, which has space embedded,
$rm My\ document.doc

Escaping the newline character.
$echo -e "The newline charcter is \n Enter the command"
Output:
The newline character /n – newline character is interpreted and cursor moves to next line
Enter the command

$ echo "the newline character is \\n enter the command"
Output:
the newline character is \\n enter the command /* here newline character interpretation is

turned off and the \n is printed as it is*/

Escaping the \ itself
**$echo **
Output
\

2.2.2 QUOTING:

- This is the another way of turning off the meaning of metacharacter.
- When a command argument is enclosed within quotes, the meaning of all enclosed special characters are turned off

$rm 'chap*' // * metacharacter meaning is turned off
removes the filename with chap* /*name of the file itself is chap*.

$rm "My\ document.doc" /* To remove the file My document.doc, which has space embedded.

$echo '\'
output
\

3. REDIRECTION: THE THREE STANDARD FILES

Redirection is the process by which we specify that a file is to be used in place of one of the standard files.

- With input files, we call it input redirection;
- With output file, we call it output redirection
- With the error file, we call it error redirection.

Standard input: The file representing the input, which is connected to the keyboard
Standard output: The file representing the output, which is connected to the display.
Standard error: the file representing the error messages that emanate from the command or shell. This is also connected to display.
Each of the three standard files are represented by a number called a **file descriptor.**
The first three slots are generally allocated to three standard streams in this manner
0: standard input
1: standard output
2: standard error

3.1 STANDARD INPUT

This file is indeed special

- The keyboard, the default source
- **a file using redirection with the < symbol**
- another program using the pipeline

- The input redirection operator is less than character (<).
- When you use wc without an argument , it prompts you to provide the input from standard input keyboard
  $ wc
  Unix is a multiuser multitasking OS
  [ctrl-d]
- When wc is used with argument. Filename is passed as an argument i,e wc takes the input from the filename we have specified

- For example: Create a file with name sample.txt
  $ vi sample.txt
  Unix is a multiuser multitasking OS

  :wq /*saving and quitting from the file

  $ wc < sample.txt /*wc command takes input from the file sample.txt
  output
  1 6 36 /* count of characters, words, lines of file sample.txt

3.2 STANDARD OUTPUT

- All commands displaying the output on the terminal actually write to the standard output file as a stream of characters and not directly to the terminal as such.
- There are three possible destinations of this stream
- The terminal, the default destination
- **A file using the redirection symbol > and >>**
- As input to another program using a pipeline
- There are two basic redirection operators for standard output.

  a. greater than character(>):

  If you want the file to contain only the output from this execution of the command, you can use greater than token.
  Ex: consider the file **sample.txt**
  $ cat sample.txt /* to display the content of sample.txt
  Unix is a multiuser multitasking OS

  $ wc sample.txt > newfile
  > symbol redirects the output of wc command to a file named newfile
  The output is now stored in newfile

  $ cat newfile /* To view the content of newfile

  1 6 37

  b Two greater than characters>> (append)

  If you want the output of the command to be appended without overwriting the existing content.

  $who >> newfile
  The output of who command is appended to newfile without overwriting the existing content.
  $cat newfile
  1 6 37 /* output of wc command executed previously

  root console aug 1 07:51 (:0) /* output of who command
  kumar pts/10 aug 1 02:51 (:0)
  sharma pts/6 aug 1 03:51 (:0)

3. 3.STANDARD ERROR

When you enter an incorrect command or try to open a non existent file, certain diagnostic messages show up on the screen. This is the standard error stream whose default destination is the terminal.

Standard output and error on monitor
Ex 1: Consider two files file1 and file2 , where file1 exist and file2 do not exist
$ ls –l file1 file2
-rwxr—r-- 1 gilberg staff 1234 oct file1
Cannot access file2: no such file or directory /*error because file2 do not exist

Ex 2: To redirect standard output to same file
$ls -l file1 file2 1>filelist 2>filelist
The 1st argument is file1 and 2nd argument is file2.
The output and errors are sent to file named filelist

$cat filelist
-rwxr—r-- 1 gilberg staff 1234 oct file1
Cannot access file2: no such file or directory

Ex 3: To redirect standard output to different files
$ls -l file1 file2 1>stdout 2>stderr
The 1st argument is file1 and 2nd argument is file2.
The output of 1st file is sent to filename stdout and errors are sent to file named stderr.

$cat stdout
-rwxr—r-- 1 gilberg staff 1234 oct file1

$cat stderr
Cannot access file2: no such file or directory

4.CONNECTING COMMANDS: PIPE.

- We often need to use a series of commands to complete a task. For example, if we need to see list of users logged into the system, we use who command.However if we need a hard copy of the list, we need two commands.
- First use **who** command to get the list and store the result in file using redirection
- We then use **lpr** command to print the file
- We can avoid the creation of intermediate file by using a pipe

Pipe is an operator that temporarily saves the output of one command in a buffer that is being used at the same time as the input of the next command.

- The first command must be able to send its output to standard output. The second command must be able to read its input from standard input.
- **The token for a pipe is vertical bar(|)**
  $ who | lpr
  We use the **who command** because it reads from the system and sends the list of users to standard output.The pipe command uses a buffer to send the piped data to next command.The recieving command must recieve its data from standard input.
  $who

root	console	aug	1	07:51	(:0)
kumar	pts/10	aug	1	02:51	(:0)
sharma	pts/6	aug	1	03:51	(:0)
rajath	pts/8	aug	1	06:51	(:0)

vikas pts/14 aug 1 09:51 (:0)

$who | wc -l
Output: 5 **/*** count of number of lines of who command
Here the output of **who** command has been passed directly as the input to **wc** command and **who** is said to be piped to **wc.**

5. SPLITTING THE OUTPUT: tee COMMAND

- The tee command is an external command and handles a character stream by duplicating its input.
- The tee command copies standard output and at the same time copies it to one or more files.
- The first copy goes to standard output i,e monitor and at the same time the output is sent to the optional files specified in the argument list.
- The tee command creates the output files, if they do not exist and overwrites them, if they already exist.

Ex:
The following command sequence uses **tee** to display the output of **who** and saves this output in a file as well.

$ who | tee user.txt

```
root    console   aug  1   07:51  (:0)
kumar   pts/10    aug  1   02:51  (:0)
sharma  pts/6     aug  1   03:51  (:0)
```

The output of who will be displayed on the monitor and at the same time it will be saved in a file **user.txt**

6.COMMAND SUBSTITUTION

- When a shell executes a command, the output is directed to standard output. Most of the time the standard output is associated with the monitor.
- There are times, however such as when we write complex commands or scripts that we need to change the output to a string that we can store in another string or variable.
- Command substitution provides the capability to convert the result of a command to a string.
- The command substitution operator that converts the output of a command to a string **is a dollar sign and a set of parentheses .**
- To invoke the command substitution, we enclose the command in a set of parentheses preceded by dollar sign($)
- When we use this command substitution, the command is executed and output is created and then converted to string of characters.
- Ex: simple demonstration of command substitution.
  $ echo " The date and time are:date"
  Output:
  The date and time are :date
- Using command substitution with date- using dollar sign along with the command enclosed in parenthesis
  $echo "The date and time are: $(date)"
  Output
  The date and time are : Mon Oct 3 07:09:48 GMT 2016

7. Searching for a pattern :

7.1 grep(globally search regular expression and print)

Unix has a special family of commands for handling search requirements and the principal member of the family is the **grep command**.

grep scans its input for a pattern and displays lines containing the pattern, the line numbers or filename where the pattern occurs.

Syntax:

grep options pattern filename(s)

grep searches for pattern in one or more filename or the standard input if no filename is specified.

The first argument(barring the options) is the pattern and the remaining arguments are filenames.

Consider three files emp.lst, emp1.lst, emp2.lst

$cat emp.lst

```
101|sharma|general manager|sales|10/09/61|6700
102|kumar|director|Sales|09/09/63|7700
103|aggarwal|manager|sales|03/05/70|5000
104|rajesh|manager|marketing|12/04/72|5800
105|adarsh|executive|sales|07/09/57|5300
```

$cat emp1.lst

```
201|anil|director|sales|05/01/59|5000
202|sunil|director|marketing|12/06/51|6000
203|gupta|director|production|09/08/55|7000
```

$cat emp2.lst

```
301|anil Agarwal|manager|sales
302|sudhir agarwal|director|production
303|rajath Agarwal|manager|sales
304|v.k.agrawal|director|marketing
305|v.s.agrawal|deputy manager|sales
306|agrawal|executive|sales
401|sumith|general manager|marketing
402|sudhir agarwal|director|production
```

Ex 1: Now to search the pattern **sales** in **emp.lst** file

```
$ grep "sales" emp.lst
101|sharma|general manager|sales|10/09/61|6700
103|aggarwal|manager|sales|03/05/70|5000
105|adarsh|executive|sales|07/09/57|5300
```

Ex 2: To search the pattern director from 2 files i,e emp.lst and emp2.lst

```
grep "director" emp.lst emp2.lst
emp.lst:102|kumar|director|Sales|09/09/63|7700
emp2.lst:302|sudhir agarwal|director|production
emp2.lst:304|v.k.agrawal|director|marketing
emp2.lst:402|sudhir agarwal|director|production
```

7.2 grep along with options

Table 13.1 Options Used by **grep**

Option	*Significance*
-i	Ignores case for matching
-v	Doesn't display lines matching expression
-n	Displays line numbers along with lines
-c	Displays count of number of occurrences
-l	Displays list of filenames only
-e *exp*	Specifies expression with this option. Can use multiple times. Also used for matching expression beginning with a hyphen.
-x	Matches pattern with entire line (doesn't match embedded patterns)
-f *file*	Takes patterns from *file*, one per line
-E	Treats pattern as an extended regular expression (ERE)
-F	Matches multiple fixed strings (in **fgrep**-style)

7.2.1 Ignoring case: when you look for a name but are not sure of the case, use the –i option to ignore case for pattern matching.

```
$ grep -i "sales" emp.lst
101|sharma|general manager|sales|10/09/61|6700
102|kumar|director|Sales|09/09/63|7700
103|aggarwal|manager|sales|03/05/70|5000
105|adarsh|executive|sales|07/09/57|5300
```

7.2.2 Deleting lines(-v): The –v option selects all lines except those containing the pattern

```
$ grep -v "director" emp2.lst
301|anil Agarwal|manager|sales
303|rajath Agarwal|manager|sales
305|v.s.agrawal|deputy manager|sales
306|agrawal|executive|sales
401|sumith|general manager|marketing
```

The lines containing the pattern **director** are deleted in the output.

7.2.3 Dislaying line numbers(-n).

The –n option displays the line numbers containing the pattern along with the line

```
$ grep -n "sales" emp.lst
1:101|sharma|general manager|sales|10/09/61|6700
3:103|aggarwal|manager|sales|03/05/70|5000
5:105|adarsh|executive|sales|07/09/57|5300
```

7.2.4 Counting lines containing patter(-c)

The –c option counts the number of lines containing the pattern.

```
$ grep -c "manager" emp2.lst
4
```

7.2.5 Dislaying filenames(-l)

The –l option displays only the names of the files containing the pattern.
Here the pattern **manager** is searched in all files ending with .lst (*.lst)

```
$ grep -l "manager" *.lst
emp2.lst
emp.lst
```

7.2.6 Matching multiple patterns(-e)

By using –e option, you can match multiple patterns

```
$ grep -e "agarwal" -e "Agarwal" -e "agrawal" emp2.lst
301|anil Agarwal|manager|sales
302|sudhir agarwal|director|production
303|rajath Agarwal|manager|sales
304|v.k.agrawal|director|marketing
305|v.s.agrawal|deputy manager|sales
306|agrawal|executive|sales
402|sudhir agarwal|director|production
```

7.3 BASIC REGULAR EXPRESSIONS

Table 13.2 The Basic Regular Expression (BRE) Character Subset

Symbols or Expression	*Matches*
*	Zero or more occurrences of the previous character
g*	Nothing or g, gg, ggg, etc.
.	A single character
.*	Nothing or any number of characters
[*pqr*]	A single character *p*, *q* or *r*
[*c1-c2*]	A single character within the ASCII range represented by *c1* and *c2*
[1-3]	A digit between 1 and 3
[^*pqr*]	A single character which is not a *p*, *q* or *r*
[^a-zA-Z]	A nonalphabetic 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

7.3.1 The character class []

A regular expression lets you specify a group of characters enclosed within a pair of rectangle brackets [ ], in which the match is performed for a single character in the group.

Thus the expression

[aA] Matches either a or A

```
$ grep "[aA]garwal" emp2.lst
301|anil Agarwal|manager|sales
302|sudhir agarwal|director|production
303|rajath Agarwal|manager|sales
402|sudhir agarwal|director|production
```

7.3.2 Negating a class(^)

Regular expressions use the **caret(^)** to negate the character class, while the shell uses **bang(!)**

Ex:

[^a-zA-Z] matches a non-alphabetic character

7.3.3 The *(asterisk)

The * refers to the immediately preceding character.Here it indicates that the previous character can occur many times or not at all.

The patttern g*

Matches none, g, gg, gg, ggg,.....

```
$ cat file.lst
ourrences of a character
ocurrences of a character
occurrences of a character
occcurrences of a character

$ grep "oc*urrences" file.lst
ourrences of a character
ocurrences of a character
occurrences of a character
occcurrences of a character
```

7.3.4 The dot (.)

A **.** matches a single character where as the shell uses **?** to indicate that.

```
$ grep "10." emp.lst
101|sharma|general manager|sales|10/09/61|6700
102|kumar|director|Sales|09/09/63|7700
103|aggarwal|manager|sales|03/05/70|5000
104|rajesh|manager|marketing|12/04/72|5800
105|adarsh|executive|sales|07/09/57|5300
```

Here the . matches single character. It list all files beginning with 10 followed by single character. It displays lines with id 101,102,103,104,105.

7.3.5 Specifying pattern locations(^ and $)

The two regular expressions characters that match pattern at the beginning or end of line .

^(caret) Matching at the beginning of the line

$(dollar) Matching at the end of the line

```
$ grep "^3" emp2.lst
301|anil Agarwal|manager|sales
302|sudhir agarwal|director|production
303|rajath Agarwal|manager|sales
304|v.k.agrawal|director|marketing
305|v.s.agrawal|deputy manager|sales
306|agrawal|executive|sales
```

^3 matches all the lines beginning with digit 3.

```
$ grep "5...$" emp.lst
103|aggarwal|manager|sales|03/05/70|5000
104|rajesh|manager|marketing|12/04/72|5800
105|adarsh|executive|sales|07/09/57|5300
```

5...$ matches all the lines ending with four digit number beginning with 5.

7.4 EXTENDED REGULAR EXPRESSIONS(ERE)

ERE make it possible to match dissimilar patterns with a single expression.
The ERE has to be used with **–E option.**

Table 13.3 The Extended Regular Expression (ERE) Set Used by **grep**, **egrep** and **awk**

Expression	*Significance*
ch+	Matches one or more occurrences of character *ch*
ch?	Matches zero or one occurrence of character *ch*
exp1|exp1	Matches *exp1* or *exp2*
GIF|JPEG	Matches GIF or JPEG
(x1|x2)x3	Matches *x1x3* or *x2x3*
(lock|ver)wood	Matches lockwood or verwood

7.4.1 The + and ?

+ Matches one or more occurrences of the previous character
? Matches zero or one occurrences of the previous character.

```
$ grep -E "oc+urrences" file.lst
ocurrences of a character
occurrences of a character
occcurrences of a character
```

The + symbol matches one or more occurrences of character c i,e **c, cc, ccc**
The occurrences of a character is not matched by +c , since there is no c in the occurrences.

```
$ grep -E "oc?urrences" file.lst
ourrences of a character
ocurrences of a character
```

The ? symbol matches zero or one occurrences of character c i,e **0 c , 1 c.**
The other two lines occurrences (2 c) and occurrences (3 c) are not matched .

7.4.2 Matching multiple patterns(|, (and))

- The | is the delimiter for the multiple patterns.
- Consider the file ere.lst with two lines of employee details.

```
$ cat ere.lst
235|barun sengupta|director|sales
279|s.n. dasgupta|manager|marketing
```

- To locate both sengupta and dasgupta from file ere.lst

```
$ grep -E "sengupta|dasgupta" ere.lst
235|barun sengupta|director|sales
279|s.n. dasgupta|manager|marketing
```

- The characters (and) lets you group patterns and use of | inside the parenthesis, you can frame more compact patterns.

```
$ grep -E "(sen|das)gupta" ere.lst
235|barun sengupta|director|sales
279|s.n. dasgupta|manager|marketing
```