# IAT-2 QUESTION PAPER AND SOLUTION

**SUBJECT CODE: 17CS35**
**SUBJECT NAME: UNIX AND SHELL PROGRAMMING**
**SEMESTER: III**
**DATE: 17OCT2018 (ODD SEM 2018)**

**1 (a) What are the three modes of vi editor? Explain.** **[06] CO1 L1**

The three modes of vi editor are

1. Command Mode
2. Input Mode or Text Mode or Insert Mode
3. ex Mode or Last Line Mode

**1. Command Mode:**

1. This is the default mode of the editor where any key that is pressed by the user is considered as a command and is interpreted to run on text.
2. Pressing a key doesn't show it on screen but may perform a function like moving the cursor, deleting or changing part of the text or perform many other operations.
3. As soon as the command is entered, it is executed – pressing the Return key is not required.
4. In command mode the user can't enter or replace text.

Command mode commands are:

1. To Move the cursor vertically
   - k Moves cursor up
   - j Moves cursor down
2. To Move the cursor horizontally
   - h Moves cursor left
   - l Moves cursor right
3. For Word Navigation
   - b Moves back to beginning of word
   - e Moves forward to end of word
   - w Moves forward to beginning of word
4. Moving to line extremes
   - 0 or | Moves to the first character of current line
   - $ Moves to the end of line
   - 30| Moves cursor to column 30 of current line
5. Scrolling
   - [ctrl-f] Scrolls forward
   - [ctrl-b] Scrolls backward
   - [ctrl-d] Scrolls half page forward
   - [ctro-u] Scrolls half page backward
6. Absolute movement
   - 40G Goes to line number 40
   - 1G Goes to line number 1
   - G Goes to last line of the file

**2. Input Mode or Text Mode or Insert Mode:**

When the vi is in Text Mode, every key pressed by the user is considered as text and is echoed on the screen. The keyboard acts as a typewriter. This mode is invoked by pressing any one of the following keys – i, a, I, A, o, O, r followed by a character, R, s, S.

Input mode commands are:

a) i Inserts text to left of cursor
b) a Appends text to right of cursor
c) I Inserts text at beginning of line
d) A Appends text at end of line
e) o Opens line below
f) O Opens line above
g) r Replaces single character
h) s Replaces single character with any number of characters
i) R Replaces text from cursor to right
j) S Replaces entire line

**3. ex Mode (or Last Line Mode):**

This mode is used to handle files like saving and performing substitution. Pressing a : (colon) in the command mode invokes this mode. Being in ex mode we can enter an **ex mode command** followed by [Enter]. After the command is executed, the user will be back to the default command mode.

ex Mode commands are

| | |
|---|---|
| a) :w | Saves file |
| b) :x | Saves file and quits vi |
| c) :wq | Saves file and quits vi |
| d) :w file1.txt | Saves file as file1.txt |
| e) :q | Quits vi when no changes are made to file |
| f) :q! | Quits vi even though changes are made to file |
| f) :sh | Escapes to UNIX shell (temporarily) |
| g) :!cmd | Executes command cmd and returns to command mode |

**Pictorial representation of the three modes of vi:**

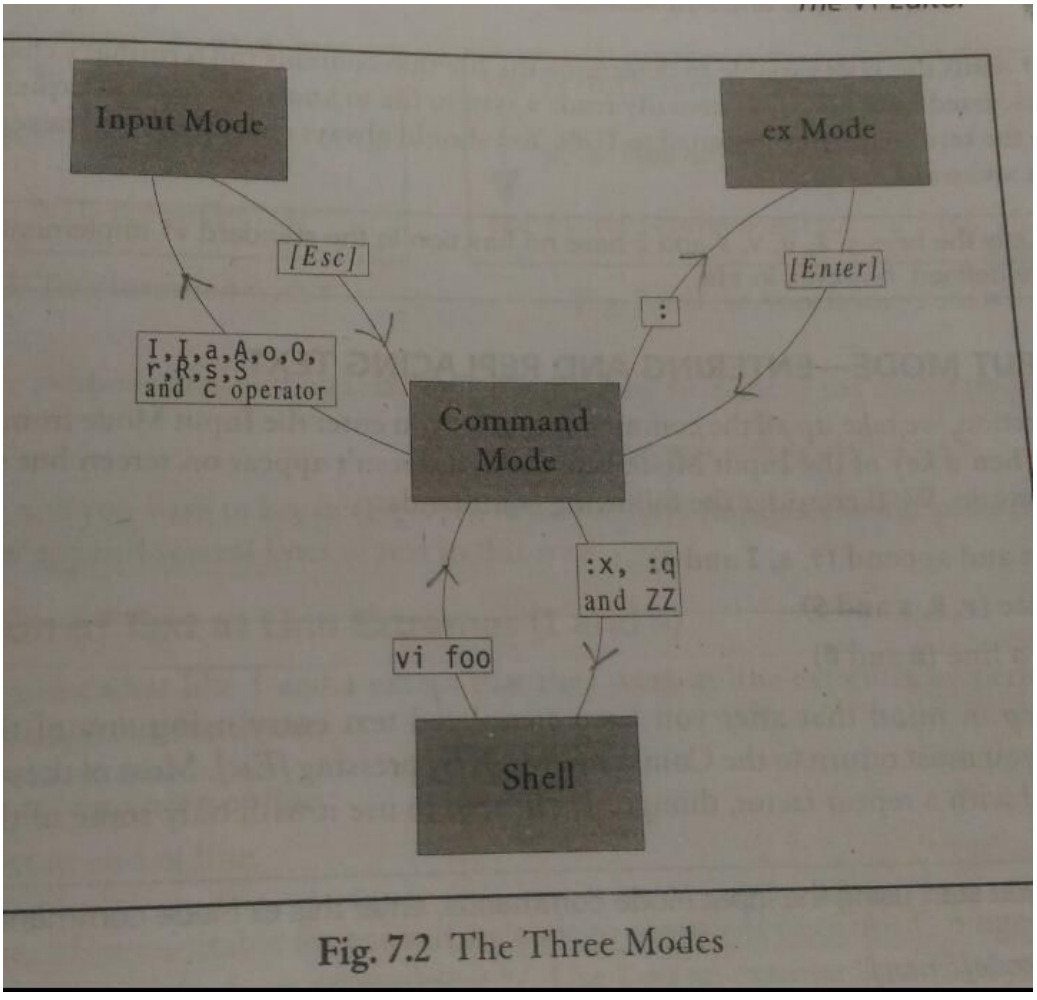



Fig. 7.2 The Three Modes

**1 (b) What is navigation? What are the commands for navigation in vi editor? [04] CO1 L1**

Navigation commands help in movement of cursor in vi editor. Navigation commands operate in command mode and doesn't show up on screen but simply performs a function.

Following are the different categories of navigation commands available in vi editor

1. Moving the cursor vertically
   - k Moves cursor up
   - j Moves cursor down
2. Moving the cursor horizontally
   - h Moves cursor left
   - l Moves cursor right
3. Word Navigation
   - b Moves back to beginning of word
   - e Moves forward to end of word
   - w Moves forward to beginning of word
4. Moving to line extremes
   - 0 or | (pipe) Moves to the first character of current line
   - $ Moves to the end of line
   - 30| Moves cursor to column 30 of current line
5. Scrolling
   - [ctrl-f] Scrolls forward
   - [ctrl-b] Scrolls backward
   - [ctrl-d] Scrolls half page forward
   - [ctro-u] Scrolls half page backward
6. Absolute movement
   - 40G Goes to line number 40
   - 1G Goes to line number 1
   - G Goes to last line of the file

**2(a). Devise wild-card patterns to match the following filenames : [05] CO1 & CO2 L1**

i) foo1, foo2 and foo5

Ans: foo[125] or foo?

ii) quit.c, quit.o and quit.h

Ans: quit.[coh] or quit.?

iii) watch.html, watch.HTML and Watch.html

Ans: [wW]atch.{html,HTML}

or

[wW]atch.[hH][tT][mM][lL]

iv) all filenames that begin with a dot and end with .swp

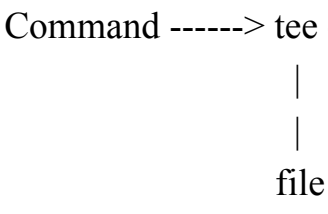
Ans: .*.swp

**2(b) What is tee and pipe command? Give examples.**

**[05] CO1 L1**

**tee command:**

tee is an external command and not a feature of the shell. It handles a character stream by duplicating its input. It saves one copy in a file and writes the other to standard output.
The tee command being a filter (which uses standard input and standard output) can be placed anywhere in a pipeline. tee will not perform any filtering action on its input, it gives out exactly what it takes.




Examples:
1.
$ who | tee user.txt
cmrit  tty7  2018-10-05 08:20 (:0)
$ cat user.txt
cmrit  tty7  2018-10-05 08:20 (:0)
In the above command the output of who is displayed to the terminal and as well as the same output is written to the file user.txt.
2.
$ who | tee /dev/tty | wc -l
cmrit  tty7  2018-10-05 08:20 (:0)
Here, using tee command we display both the list of users and their count on the terminal.

**Pipe command:**
The standard input and standard output are two separate streams and are individually manipulated by the shell. So it is possible for the shell to connect these streams so that one command takes input from the other.  The shell connects these streams using a special operator '|' called **pipe.**

When two commands are connected with a pipe, the output of first command is passed directly as input to the second command. It's the shell that sets up this connection and the commands have no knowledge of it.  There is no restriction on the number of commands that can be used in  pipeline.

Examples:

1.
$who | wc -l #who is said to be piped to wc.
1
$

We will get the count of how many users have logged into the system currently.

2.
$ls | wc -l
36

We get the count of number of files in the current working directory.

**3(a) Explain grep command with all options.**

**[06] CO1 & CO2 L4**

**Command grep:**

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

Syntax:
grep options pattern filename(s)

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

Examples:
**1. To search for the pattern "sales" in the file staff.txt**
$ grep "sales" staff.txt
1006|chanchal singhvi|gm |sales |7600
1234|jai lalit |director|sales |5000
grep command displays all lines in staff.txt containing the pattern "sales" as output.

**2. To search for the pattern president in staff.txt**
$ grep president staff.txt
$

grep command doesn't display any lines as the pattern 'president' is not found in file staff.txt. Note the pattern is not enclosed in double quotes.

**3. To search for a pattern in more than one file**

```
$ grep director staff.txt emp.txt
staff.txt:9876|Jai Shama       |director|production|7000
staff.txt:1234|jai lalit       |director|sales     |5000
staff.txt:4532|lalit chowdury  |director|marketing |4000
emp.txt:9876|anitha|director|production|7000
emp.txt:1234|kalpana|director|sales     |5000
emp.txt:4532|lalit chowdury  |director|marketing |4000
```

grep command displays only those lines in file staff.txt and emp.txt which have the pattern 'director'.

If the pattern is more than one word, we need to use double quotes for the pattern as shown below.

```
$ grep "jai lalit" staff.txt
1234|jai lalit       |director|sales     |5000
```

The options of grep are :

| Sl. No. | Options | Descriptions |
|---|---|---|
| 1 | -i | Ignores case for matching |
| 2 | -v | Doesn't display lines matching expression |
| 3 | -n | Displays line numbers along with lines |
| 4 | -c | Displays count of number of occurrences |
| 5 | -l | Displays list of file names only |
| 6 | -E | Treats pattern as an extended regular expression (ERE) |
| 7 | -f file | Takes pattern from file, one per line |

Examples:

$grep -i ‘agarwal’ emp.lst
(-i ignores case, displays all having agarwal, Agarwal, AGARWAL AgarWal etc.)

$grep -v ‘director’ staff.txt
(displays all lines except the line containing the pattern - director)

$ grep -n ‘marketing’ staff.txt
(displays line numbers along with the lines matching the pattern)

$grep -c ‘director’ staff.txt
(displays count of lines containing pattern)

$grep -l ‘manager’ *.txt
(displays only filenames containing the pattern)

**3(b) Explain egrep with examples**

**[04] CO1 & CO2 L4**

**grep -E is same as egrep**

**Extended regular expression (**ERE**)** make it possible to match dissimilar patterns with a single expression.

+ - Matches one or more occurrences of the previous character
? - Matches zero or one occurrences of the previous character
exp1 | exp2 – Matches exp1 or exp2
GIF|JPEG – Matches GIF or JPEG
(x1|x2)x3 – Matches x1x3 or x2x3
(lock|ver)wood - Matches lockwood or verwood

grep -E “[aA]gg?arwal” staff.txt
or
egrep “[aA]gg?arwal” staff.txt

Matches Agarwal, aggarwal, aggarwal and Aggarwal

grep -E ‘sengupta|dasgupta’ staff.txt
or
egrep ‘sengupta|dasgupta’ staff.txt
Matches sengupta or dasgupta

grep -E ‘(sen|das)gupta’ staff.txt
or

egrep ‘(sen|das)gupta’ staff.txt
matches sengupta or dasgupta

**4(a). Briefly explain the wild-card characters used in shell and grep command with examples. [5+5] CO1 & CO2 L4**

**The wild-card characters used in Shell and their examples are -**

| Sl. No. | WILD CARD | MEANING |
|---|---|---|
| 1 | * | Matches **zero or more characters** except a leading dot |
| eg | chap* | Matches **chap**, chap01, charpter1, chapxxx i.e., all filenames beginning with chap |
| 2 | ? | Matches **one character** except a leading dot |
| eg | chap? | The pattern ‘chap?’ matches all five character filenames beginning with chap. Eg. chapx, chapy, chapz |
| 3 | [ ] | The character class: Matches a single character enclosed in the brackets (- for range, ! to exclude) |
| eg. | chap0[124] | Matches file names chap01 chap02 chap04 |
| eg. | chap0[1-4] | Matches file names chap01 chap02 chap03 chap04 |
| eg. | *.[!co] | Matches all filenames with a single character extension but not the .c and .o files. |
| eg. | [!a-zA-Z]* | Matches all filenames that don’t begin with an alphabetic character. |
| 4 | {pat1, pat2,...} | Pat1, pat2, etc. (Not in Bourne Shell) |
| eg | ***.{c,java,txt}** | Matches all files with extension .c, .java and .txt |

**The wild-card characters used in grep and their examples are -**

| Sl. No. | Symbols or Expression | Matches |
|---|---|---|
| 1 | * | Zero or more occurrences of previous character |
| eg | g* | g, gg, ggg, etc. |
| 2 | . | A single character |
| eg | 2... | Matches a four character pattern beginning with 2 |
| eg | .* | Nothing or any number of characters |
| 3 | [ ] | Matches a single character enclosed in the brackets |
| eg | [pqr] | A single character either p or q or r |
| eg | [1-3] | A digit between 1 and 3 |
| 4 | [^] | A ^ (caret) with in the character class does not match the character |
| eg | [^pqr] | A single character other than p, q or r |
| eg | [^a-zA-Z] | A non alphabetic character |
| 5 | ^ | For matching at the beginning of a line |
| eg | ^printf | Matches pattern printf at the beginning of line |
| 6 | $ | For matching at the end of the line |
| eg | base$ | Pattern base at end of line |
| eg | ^base$ | Base as the only word in line |
| eg | ^$ | Lines containing nothing (empty lines) |

**5(a) What is shell programming? Write a shell program to create a menu and execute a given option based on user's choice. Option include i) current month calendar ii) process status iii) list of files iv) current date v) content of a given file vi) Display current logged in users. [10] CO3 & CO4 L4**

**Shell Programming:**

In UNIX, to perform a specific function a group of commands have to be executed regularly, instead, they can be stored in a file, and the file itself

can be executed. This method of programming is called **shell programming** and the file containing the set of commands is called **shell script or shell program**.

All shell scripts use the .sh extension though it is not mandatory.

Shell scripts are executed in a separate child shell process. By default, the child and the parent shells belong to the same type. But, the user can use a different sub-shell by providing a special interpreter line in the first line of the shell script to specify a different shell for the script.

```
#!/bin/sh
#menu.sh: Uses case to offer 6 item menu
#
echo   "                MENU  \n
1.  Current month calendar   \n
2.  Process Status  \n
3.  List of files  \n
4.  Current Date  \n
5.  Content of a given file  \n
6.  Display Current logged in users  \n
7.  Exit  \n
Enter your choice :\c"
read choice
case "$choice" in
     1)   cal     ;;
     2)   ps   -f  ;;
     3)   ls       ;;
     4)   date    ;;
     5)   echo "Enter the file name to display contents:  \c"
          read fname
          cat  $fname  ;;
     6)   who    ;;
     7)   exit    ;;
     *)   echo "Invalid option"
esac
```

**6(a) Discuss the following command with respect to vi editor** **[10] CO1 & CO2 L4**

| | |
|---|---|
| 1) b | Moves cursor back to beginning of word |
| 2) w | Moves cursor forward to beginning of word |
| 3) \|(pipe) | Moves cursor to first character of the current line |
| 4) :1,5w ab.txt | Writes line 1 to 5 to the file watch.txt |
| 5) G | Moves cursor to the first character of last line |
| 6) h | Moves cursor left by one character |
| 7) e | Moves cursor forward to end of word |
| 8) J | Joins the current line with the line following |

9) 1,$s/director/manager/gc

In the entire file substitutes all occurrences of director with manager but **selectively, that is only after accepting confirmation from the user.**

The cursor is positioned at each occurrence of **director** and the message ‘replace with manager (y/n/a/q/l/^E/^Y)?’ is displayed at

the last line. If the user presses 'y' the pattern – **director** is replaced with manager. If the user presses 'n' the pattern is not replaced.

10) yy yanks or copies the current line

**7(a) Name and explain the three standard files used by UNIX commands? Explain with example.**

**[06] CO1 & CO2 L4**

The three standard files used by UNIX commands are

1. Standard input
2. Standard output
3. Standard error

**1. Standard input:**

Standard input is file (or stream) representing input, which is by default connected to the keyboard. Standard input file can represent three input sources. They are

**a. The keyboard, the default source**

eg: $ wc [Enter]
wc taking input from
the keyboard - the default
source [ctrl-d]
3 10 54

**b. A file using redirection with the < symbol (a metacharacter)**

eg: $ wc < sample.txt
9 43 253 sample.txt

**c. Another program using a pipeline**

eg: $ cat sample.txt | wc -l

**2. Standard output:**

Standard output is the file (or stream) representing output, which is by default connected to the display. When the command

displays output on the terminal, it actually writes to the **standard output** file as a stream of characters and not directly to the terminal.

There are three possible destinations for standard output. They are

**1. The terminal, the default destination**

eg: $ cat file.txt

**2. A file using the redirection symbols > and >>**

eg:$ cat file.txt > newfile

$ cat file1.txt >> newfile

**3. As input to another program using a pipeline**

eg: $ ls -l | wc -l

**3. Standard error:**

Standard error is the file (or stream) representing error messages that emanate from the command or shell. This is also by default connected to the display.

Each of the three standard files is represented by a number, called a **file descriptor.** The kernel maintains a table of file descriptors for every process running in the system. In the table, the first three are allocated to the three standard streams

0 – standard input
1 – standard output
2 – standard error

Eg:

$ cat nofile.txt
cat: nofile.txt: No such file or directory

1. $ cat nofile.txt > sample.txt
cat: nofile.txt: No such file or directory

The diagnostic output has not been sent to errorfile. The error stream can't be captured with >. The standard error can't be

redirected in the same way standard output (> and >>) can be done.

**To redirect standard error we have to use 2> symbol**

2. $ cat nofile.txt 2> errorfile

$ cat errorfile
cat: nofile.txt: No such file or directory

We can also append diagnostic output as shown below

3. $ cat secondnofile 2>> errorfile

$ cat errorfile
cat: nofile.txt: No such file or directory
cat: secondnofile: No such file or directory

4. $find.sh > find.out 2> find.error

The output of the find.sh script is redirected to find.out file and the error messages are redirected to the file find.error.

**7(b) Explain the following commands with examples.**
**a) set b) map c) ab [04] CO1 & CO2 L4**

**set commands:**

Set command is used to modify the configurations of vi session.

**:set all** command displays a list of the options and settings

**:set command displays all the current settings**

The below table shows some of the common options of set:

| "set" Command | Short form | Description |
|---|---|---|
| :set tabstop=8 | :set ts | Tab key displays 8 spaces |
| :set ignorecase<br>:set noignorecase | :set ic<br>:set noic | Case sensitive searches |
| :set number<br>:set nonumber | :set nu<br>:set nonu | Display line numbers |

| | | |
|---|---|---|
| :set showmode<br>:set noshowmode | | Editor mode is displayed on bottom of screen |

**ab command:**

We can define abbreviations that vi will automatically expand into the full text whenever its typed during text-input mode.

To define abbreviation, use the ex command

**:ab abbr phrase**

**abbr** is an abbreviation for the specified **phrase**. The sequence of characters that make up the abbreviation will be expanded during text-input mode only if you type it as a full word, abbr will not be expanded within a word.

Examples:

:ab os Operating System

:ab #d #define

:ab #i #include

:ab teh the

:ab ibm International Business Machines

:ab cmrit  CMR Institute of Technology

The above command abbreviates 'CMR Institute of Technology' to the initials cmrit. Whenever the user type cmrit as a separate word during text-input mode, cmrit expands to the full text.

Abbreviation expand as soon as the the user types a nonalphanumeric character (eg. a punctuation), a carriage return, or ESC (returning to command mode).

An abbreviation won't expand when you type an underscore (_), as it is treated as part of the abbreviation.

**To disable abbreviations:**

We can disable the abbreviation by typing

**:unab abbr**

Example:

:unab cmrit

**To list currently defined abbreviations:**

To list currently defined abbreviations, type

:ab

**map command:**

vi allows user to map keys for any of the vi commands. After mapping a key, when the user presses the mapped key, the function assigned to the key is performed.

**To map basic keys:**

Example:
**1)**
**:map – x**

x in command mode deletes the character at the current cursor position. We are mapping x to '-'. Now pressing '-' deletes the character at the current cursor position.

**2)**
**:map – dd**

dd in command mode deletes the current line. The '-' key is mapped to dd. So by pressing the '-' key, deletes current line.

**To map special characters:**

:map <keyname> command

Example:
:map ^D dd

The above command maps the key <ctrl-d> to dd. Pressing <ctrl-d> will delete the current line.

**To unmap:**

:unmap dd
:unmap ^D
:unmap -

:unmap h

**To list the current mappings:**

**:map**

**8(a) Write UNIX commands for the following:**

**[04] CO1 & CO2 L1**

1. Writing the first 50 lines to another file. (vi editor command)
   Ans :1,50w newfile
2. Searching for a pattern in backward direction and to repeat the same pattern search in opposite direction.
   Ans: ?pattern[Enter] followed by 'N'
3. Inserting a text at the beginning of the line. (vi editor command)
   Ans: I
4. To escape to UNIX shell (vi editor command)
   Ans: :sh
5. List all the files in PWD which are having exactly five characters in their filename and any number of characters in their extension.
   Ans: ls ?????.* or ls $PWD/?????.*

**8(b) Explain the use of test and [ ] to evaluate an expression in shell with an example. [06] CO3 & CO4 L4**

The command test uses certain operators to evaluate the condition on its right and returns either a true or false exit status, which is then used by if for making decisions.

The command test works in 3 ways:
**1. Compares two numbers**
**2. Compares two strings or a single one for a null value**
**3. Checks a file's attributes**

**1. Numeric Comparison operators used by test:**

| Sl. No. | Operator | Meaning |
|---|---|---|
| 1 | -eq | Equal to |

| | | |
|---|---|---|
| 2 | -ne | Not equal to |
| 3 | -gt | Greater than |
| 4 | -ge | Greater than or equal to |
| 5 | -lt | Less than |
| 6 | -le | Less than or equal to |

Examples:

$ x=5; y=7; z=7.2

a) $ test $x -eq $y ; echo $? # so returns 1 (false)
1

b) $ test $x -lt $y ; echo $? # returns true (0)
0

c) $ test $z -gt $y ; echo $? # numeric comparison is restricted
to
1 # integers. Therefore, returns false

d) $ test $z -eq $y ; echo $? # returns true, 7.2 is treated as 7
0

Examples c and d shows that test uses only integer comparisons

**2. String comparisons used by test**

| Sl. No. | Test | True if |
|---|---|---|
| 1 | S1 = s2 | String s1 = s2 |
| 2 | S1 != s2 | String s1 is not equal to s2 |
| 3 | -n str | String str is not a null string |
| 4 | -z str | String str is a null string |
| 5 | str | String str is assigned and not null |
| 6 | S1 == s2 | String s1 = s2 (Korn and Bash only) |

**3. FILE TESTS**

| Sl. No. | Test | True if file |
|---|---|---|
| 1 | -f file | File exists and is a regular file |
| 2 | -r file | File exists and is readable |
| 3 | -w file | File exists and is writable |
| 4 | -x file | File exists and is executable |
| 5 | -d file | File exists and is a directory |
| 6 | -s file | File exists and has a size greater than zero |
| 7 | -e file | File exists (Korn and Bash only) |
| 8 | -u file | File exists and has SUID bit set |
| 9 | -k file | File exists and has sticky bit set |
| 10 | -L file | File exists and is a symbolic link (Korn and Bash only) |
| 11 | f1 -nt f1 | F1 is newer than f2 (Korn and Bash only) |
| 12 | f1 -ot f2 | F1 is older than f2 (Korn and Bash only) |
| 13 | f1 -ef f2 | F1 is linked to f2 (Korn and Bash only) |

**SHORTHAND FOR test:**
As test is widely used, there exists a shorthand method of executing it. **A pair of rectangular brackets enclosing the expression can replace it.**

**Thus the following two forms of test are available**
**1. test $x -eq $y**
**2. [ $x -eq $y ]**

Note that we must provide whitespace around the operators (around -eq), their operands (around $x and $y) and inside the [ and ]. The second form is easier to use but be liberal to use whitespace.

```
$ x=3; y=3
$ [  $x  -eq  $y  ] ; echo $?
0                # successful as space is provided as required

$ [$x-eq$y] ; echo $?
[3-eq3]: command not found
127              # fails: as no space is provided inside [, ], and around
-eq
```

**Example:**

```
#!/bin/sh
# filetest.sh: Tests file attributes
#
if [ ! -e $1 ] ; then
   echo “$1 does not exist”
elif [ ! -r $1 ] ; then
   echo “$1 is not readable”
elif [ ! -w $1 ] ; then
   echo “$1 is not writable”
else
   echo “$1 is both readable and writable”
fi
```