**0S LAB**

**4TH SEM IT**

**OPERATING SYSTEM**

An operating system is a software program designed to act as an interface between a user’s and a computer. It controls the computer hardware manages system resources and supervisors the interaction between the system and its users. The operating system also forms a base on which application software are developed and executed.

**HISTORY OF UNIX:**

UNIX was the first operating system that was made portable. This portability was consequence of using c, a system programming language developed at Bell Laboratories.

As UNIX developed, it became widely used within Bell Laboratories. AT & T could not market Unix due to certain government policies, and hence distributed if to academic and research institutes of a nominal fee.

University of California, Berkeley (USB) made a huge contribution to the growth of UNIX. A part from AT & T, most of the major features of UNIX. System is their contribution. They developed their own version of UNIX by the name BSD (Berkeley software distribution) UNIX. Their own version was called 4.3 BSD, it introduced features like

* Vi edition
* C shell
* More versatile mail features
* Symbolic links
* A networking protocol software TCP/IP along with their standard product, which made internet possible.

**MAJOR FEATURES OF UNIX:**

The major features of UNIX can thus be listed as follows:

* Multiuser, time sharing OS
* Multitasking OS
* Portability
* Modularity
* File structure and security
* Device independence
* Communication

**UNIX ARCHITECTURE:**

Unix has a layered architecture with two major separable parts the Kernel and the system program. Everything below the system call interface, and above the physical hardware is the kernel. figure depicts the most basic UNIX architecture.

Thus, the components of the UNIX as are:

* Kernel
* Shell
* Command and utilities
* Application

**KERNEL:**

It is the heart of the operating system. All the components request the kernel for the required services. when the system is boosted, the kernel is read into memory.it resides in the memory as long as the system is running. The function and service of the kernel are:

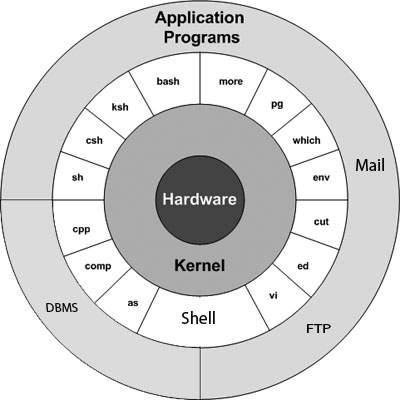
* File management and security
* I/O services
* Process scheduling and management
* System accounting
* Memory management
* Interrupt and error handling
* Date and time services

**SHELL:**

* It is the command interpreter, the interface between a user and the system.
* The kernel activities an interface programs called shell for each user logged into the system.
* Shell essentially provide a way to separate users or tasks from each other’s while the kernel maintains a unified overall control.
* Shell provides a prompt like $ or % depending on the variant of the shell being used.

**APPLICATION:**

* In addition to command & utilities, a number of UNIX based application programs are also available.
* Word processor, spread sheet, data managers, cap/cam programs are some examples of this application.
* For the kernel, application are just programs like commands & utilities.



**UNIX COMMAND:**

**SHELL COMMANDS SYNTAX:**

UNIX commands are grouped into 2-categories

1. Internal
2. External

**Internal Commands:**

There is a set of commands which are part of the shell doesn’t need to search the given path in the path variable to find the corresponding files. These are internal commands also called as shell. The example of 2 such commands are also: echo & pwd.

**External Commands:**

All other commands are available as independently compiled “C” program usually in ‘/ ‘/us directory the shell searches the path for these command files. They form the set of external commands.

**UNIX Commands:**

**1 .Date Command :**

This command is used to display the current data and time.

**Syntax :**

$date

$date +%ch

**Options : -**

a = Abbrevated weekday.

A = Full weekday.

b = Abbrevated month.

B = Full month.

c = Current day and time.

C = Display the century as a decimal number.

d = Day of the month.

D = Day in „mm/dd/yy‟ format

h = Abbrevated month day.

H = Display the hour.

L = Day of the year.

m = Month of the year.

M = Minute.

P = Display AM or PM

S = Seconds

T = HH:MM:SS format

u = Week of the year.

y = Display the year in 2 digit.

Y = Display the full year.

Z = Time zone .

To change the format :

Syntax :

$date „+%H-%M-%S‟

**2. Calender Command :**

This command is used to display the calendar of the year or the particular month of calendar year.

Syntax :

a.$cal <year>

b.$cal <month> <year>

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

**3. Echo Command :**

This command is used to print the arguments on the screen .

**Syntax :** $echo <text>

**Multi line echo command :**

To have the output in the same line , the following commands can be used.

**Syntax :** $echo <text\>text

To have the output in different line, the following command can be used.

**Syntax :** $echo “text

>line2

>line3”

**4. Banner Command :**

It is used to display the arguments in „#‟ symbol .

**Syntax :** $banner <arguments

**5.’who’ Command :**

It is used to display who are the users connected to our computer currently.

**Syntax :** $who – option‟s

**Options : -**

H–Display the output with headers.

b–Display the last booting date or time or when the system was lastely rebooted.

**6.’who am i’ Command :**

Display the details of the current working directory.

**Syntax :** $who am i

**7.’tty’ Command :**

It will display the terminal name.

**Syntax :** $tty

**8.’Binary’ Calculator Command :**

It will change the „$‟ mode and in the new mode, arithematic operations such as +,-,\*,/,%,n,sqrt(),length(),=, etc can be performed . This command is used to go to the binary calculus mode.

**Syntax :**

$bc operations

^d

$

1 base –inputbase

0 base – outputbase are used for base conversions.

Base :

Decimal = 1 Binary = 2 Octal = 8 Hexa = 16

**9.’CLEAR’ Command :**

It is used to clear the screen.

**Syntax :** $clear

**10.LIST Command :**

It is used to list all the contents in the current working directory.

**Syntax :** $ ls – options <arguments>

If the command does not contain any argument means it is working in the Current directory.

**Options :**

a– used to list all the files including the hidden files.

c– list all the files columnwise.

d- list all the directories.

m- list the files separated by commas.

p- list files include „/‟ to all the directories.

r- list the files in reverse alphabetical order.

f- list the files based on the list modification date.

x-list in column wise sorted order.

**DIRECTORY RELATED COMMANDS :**

**1.Present Working Directory Command :**

To print the complete path of the current working directory.

**Syntax :** $pwd

**2.MKDIR Command :**

To create or make a new directory in a current directory .

**Syntax :** $mkdir <directory name>

**3.CD Command :**

To change or move the directory to the mentioned directory .

**Syntax :** $cd <directory name.

**4.RMDIR Command :**

To remove a directory in the current directory & not the current directory itself.

**Syntax :** $rmdir <directory name>

**FILE RELATED COMMANDS :**

**1. CREATE A FILE :**

To create a new file in the current directory we use CAT command.

**Syntax:**

$cat > <filename

The > symbol is redirector we use cat command.

**2.DISPLAY A FILE :**

To display the content of file mentioned we use CAT command without „>‟ operator.

**Syntax :**

$cat <filename.

Options –s = to neglect the warning /error message.

**3.COPYING CONTENTS :**

To copy the content of one file to another. If file does not exist, a new file is created and if the file exists with some data then it is overwritten.

**Syntax :**

$ cat <filename source> >> <destination filename>

$ cat <source filename> >> <destination filename> it is avoid overwriting.

**Options : -**

-n content of file with numbers included with blank lines.

**Syntax :**

$cat –n <filename>

**4.SORTING A FILE :**

To sort the contents in alphabetical order in reverse order.

**Syntax :**

$sort <filename >

**Option :** $ sort –r <filename>

**5.COPYING CONTENTS FROM ONE FILE TO ANOTHER :**

To copy the contents from source to destination file . so that both contents are same.

**Syntax :**

$cp <source filename> <destination filename>

$cp <source filename path > <destination filename path>

**6.MOVE Command :**

To completely move the contents from source file to destination file and to remove the source file.

**Syntax :**

$ mv <source filename> <destination filename>

**7.REMOVE Command :**

To permanently remove the file we use this command .

**Syntax :**

$rm <filename>

**DIFFERENT TYPES OF FILE IN UNIX:**

In UNIX, all information is treated as file so besides a user program files. There are also special file such as file that contain information about directory contents of file that store information about various I/O devices connected to the system.

There are 3 categories of a file:  
**ORDINARY FILE:**

All file created by users’ ways under these categories. It includes all data files, programming file object executable file. The user can make changes to such file.  
  
**DIRECTORY FILE:**

For each directory there is a file by the same names which contains information about files under that directory. For example, for the directory / users / games. There will be a directory file called game in the directory / user which contains certain information on all the files and directory under the directory games. A directory files are automatically created by UNIX when directory is created.A directory file cannot be modified by user what is instead modified automatically by the system when a new file / a sub directory are added to the directory.  
  
**SPECIAL FILE:**

Most of the system files of UNIX are special files. Special files are typically associated with IO devices such files are found in the standard UNIX directories such as DEV / END etc special files can’t be allowed.  
  
**TYPES OF USER IN UNIX:**

**SYSTEM ADMINISTRATOR:**

The sys0tem administrator (SA) is primarily responsible for the smooth operation of the system. it's the job of SA to switch job on the system console. The SA also creates users and group of users for the system and takes backup of the data to present loss of data due to system breakdown.  
  
**FILE OWNER:**

The user who creates the file is said to the owner of that file. The owner of a file can perform any operation of data file copy, edit, delete.  
  
**GROUP OWNER:**

In UNIX it's applicable to define the users who will belong to a group. A group of users are also given a name, just as a user is given a name and can be a group owner of a file.

**OTHER USERS**

All users of the system who are not member of that group.

**RULES FOR FILE NAMING:**

In UNIX, file names

It can be up to 14 characters long.  
It can contain digit, dot, hyphen or underscore anywhere.  
It can contain both upper and lowercase alphabet this above two's are different to each other that is case sensitive.  
**EXAMPLE:** “bio.txt” and “bio text” are different to each other blank or tab are not allowed in between file name.  
**COMMAND:**  
**cat:** It is used to create a file   
**SYNTAX:** cat **>** filename

**EXAMPLE:** cat **>** BIODATA

NAME - GOURAV

BRANCH - IT

SEM - 4TH

ROLL NO. - 09

Save this file Press control + D  
**VI EDITOR:**

* An editor is the works space.Where all write and edit text program or script.txt,program or script UNIX provide a powerful editor VI.
* It is a full screen editor and it was developed by wailingvillying it display a contain of file on the screen and allows the user to insert,delete or change some part of the state.
* At the command type VI called the file name.
* Syntax: -VI<file name>
* Example: -$VI std.txt
* An empty VI screen will be present and each empty line is mark symbol() is shows 25lines are available for textediting and the last time i.e. 25 is reverse from some commands that can enter to act on the text.

**VI MODE:**

* The VI editor is having 3modes of execution the entire work of entering and editing the text can be done through a combination of command available of 3modes are: -
* Command Mode
* I/P Mode
* Ex Mode
* **COMMAND MODE:**
* This is the mode i.e. used when we initially placed in a blank file.Here use can pass command to act on the text using most of the key or the key board a lot of key function is equivalent command in ex mode.
* In this mode the text can’t be replaced on entered.To enter text, you have to other more called I/O mode by processing the following tress.
* I,I,a,A,o,O,r,R,s,S
* **INPUT MODE**
* In this mode only key i.e. processed in entered as that will have this mode by using ESC key.
* **EX MODE:**
* It uses explicit commands entered in the least of the screen-obtain as seen as we press (:)we can exit.
* Ex mode and return to the command and by processing entered key.

**BASIC VI COMMAND**:

* Input output mode command:

|  |  |
| --- | --- |
| COMMAND | PURPOSE |
| i | Insert the text to the left the cursor (existing text is shifted to left). |
| I | Insert the text to the beginning of line(existing text is shifted to right). |
| a | Append the text to the right cursor(existing text is shifted tp right). |
| A | TO append to end of the line. |
| o | Enter the line for input below the current line. |
| O | Enter the line for input above the line. |
| r | Replace a single character. |
| R | Replace text from cursor to right existing the entire line. |
| S | Replace the entire line |
| S | Replace single character on the cursor with any no. of character. |

* EX MODE COMMANDS: Ex mode save and quit command.

|  |  |
| --- | --- |
| COMMAND | PURPOSE |
| iw | To save the file and naming in entering and editing mode. |
| ix | To save the files and quiet from the editor. |
| :sh | Escape to UNIX shell. |
| :wr | To save the file and quiet editing mode. |
| :q | To quiet editing mode without saving the buffer. |
| :q! | Quiet without saving |
| :wq | Quit with saving. |

* COMMAND MODE COMMAND: Cursor movement on navigation command.

|  |  |
| --- | --- |
| COMMAND | PURPOSE |
| k | It moves the cursor up before one line. |
| j | It moves the cursor to down. |
| h | It moves the cursor left one space. |
| B | It moves the cursor backward one space. |
| l | It moves the cursor right one space. |
|  | It moves the cursor backward to the beginning of the next sentence. |
|  | It moves the cursor backward the beginning |
| w | It moves the forward to the beginning of a word. |
| 0 or 1 | It moves the beginning of the line. |
| $ | It moves the end of the line. |
| 1F | Scroll of the full page forward. |
| 1b | Scrollof the full page backward. |

* FILE HANDLING USING EX MODE:

|  |  |
| --- | --- |
| COMMAND | PURPOSE |
| :r file no 1 | It reads files 1 below current line. |
| :r !head-5data | It read first line of data one below current line. |
| :w>>5 | It append the buffer contain of file in file1. |
| :efile1 | Editing of current file in top editing file 1 start. |
| :e# | It returns to editing most recently editing file. |
| :n | It editing next file(if VI is invoked with more than one file name). |
| :rew | It rewinds the file list to start from first. |

* DELETION:

|  |  |
| --- | --- |
| COMMAND | PURPOSE |
| (ESC)\*(sm) | Delete one character on which cursor is present. |
| (ESC)\*(cap) | Delete a character to the left cursor. |
| dd | Delete the current line. |
| d/target | Delete text from the cursor to target. |
| dw | Delete a word |
| Ds$D | It delete from current cursor position to end of the line. |

**UNIX SHELL PROGRAMMING**

Shell programming is a group of commands grouped together under single filename. After logging onto the system a prompt for input appears which is generated by a Command String Interpreter program called the shell. The shell interprets the input, takes appropriate action, and finally prompts for more input. The shell can be used either

interactively - enter commands at the command prompt, or as an interpreter to execute a shell script. Shell scripts are dynamically interpreted, NOT compiled.

**Common Shells**.

**C-Shell - csh** : The default on teaching systems Good for interactive systems Inferior programmable features

**Bourne Shell - bsh or sh - also restricted shell - bsh** : Sophisticated pattern matching and file name substitution

**Korn Shell** : Backwards compatible with Bourne Shell Regular expression substitution emacs editing mode

**Thomas C-Shell - tcsh** : Based on C-Shell Additional ability to use emacs to edit the command line Word completion & spelling correction Identifying your shell.

**SHELL SCRIPT:**

It we have a sequence of UNIX command that is used frequently, so we have to store them in a file. It is than possible to have the shell read the file and execute the command , such a file called script file. Shell Script allows I/O and manipulate variable and also can have powerful flow of control and iteration construct that make program possible. A file can be executed by the “sh” command.

Syntax: sh<file name>

Example: sh<biodata>

**UNIX SHELL PROGRAMMING COMMANDS**

**ECHO COMMAND:**

It is used to display message on the screen.

Syntax: echo message

Example: echo this is my first program.

**VARIABLE:**

In UNIX variable do not have explicitly declare they can be create as any point of types by a simple assignment of values.

**Local variables**

Local variables are in scope for the current shell. When a script ends, they are no longer available; i.e., they go out of scope. Local variables are set and assigned values.

**E*XAMPLE***

variable\_name=value name="John Doe" x=5

**Global variables**

Global variables are called environment variables. They are set for the currently running shell and any process spawned from that shell. They go out of scope when the script ends.

**E*XAMPLE***

VARIABLE\_NAME=value

export VARIABLE\_NAME

PATH=/bin:/usr/bin:.

export PATH

**Extracting values from variables** To extract the value from variables, a dollar sign is used.

**E*XAMPLE***

echo $variable\_name echo $name echo $PATH

**Rules : -**

1. A variable name is any combination of alphabets, digits and an underscore („-„);

2 .No commas or blanks are allowed within a variable name.

3. The first character of a variable name must either be an alphabet or an underscore.

4. Variables names should be of any reasonable length.

5. Variables name are case sensitive . That is , Name, NAME, name, NAme, are all different variables.

**REFERENCING TO A VARIABLE:**

To $ symbol is used to refer to the contents of a variable.

Example: Echo $age

Echo your age is “$age

**READING A VALUE IN TO A VARIABLE:**

In UNIX the shell also allows a user to enter a value into variable during execution of a shell script. This is done by using read.

Example: read name

**EXPR COMMAND:**

It stands for expression. The shell do not support numeric variable. All the variable are printed as char string. In order to implement mathematically manipulation of variable expr command is necessary.

Exp1-$ expr 4+5

Output-9

Exp2-$ expr 4-2

Output-2

Exp3-$ expr 10/5

Output-2

Exp4-$ expr 7/\*2

Output-14

Exp5-$ expr $a + $b

Output

**Note:** There must be space an either side of operator variable can be used in expr command are:-

Example:$a=5

$b=4

$ expr $a $b

Example:the sum is expr $a + $b

The operator support by expression more +,-,\*,/.

**Notes:**

1. If the expression is c=10+20

It is to be typed as put space before & after of ‘+’ symbol.

Do not put space before & after of equal to ‘=’ symbol.

1. When we use numerical values for a comparison that time –eq to be used.

But in character or string we use (=) symbol for comparison

However when ‘\*’is used it. Should be produced by \ otherwise as shell interrupt.

1. It is a wild and character.

**CONDITIONAL STATEMENTS :**

The if construct is followed by a command. If an expression is to be tested, it is enclosed in square brackets. The then keyword is placed after the closing parenthesis. An if must end with a fi.

1. **Syntax of if :**

This is used to check a condition and if it satisfies the condition if then does the next action , if not it goes to the else part.

1. **Syntax of if…else :**

If cp $ source $ target

Then

Echo File copied successfully

Else

Echo Failed to copy the file.

1. **Syntax of nested if :**

here sequence of condition are checked and the corresponding performed accordingly.

if condition

then

command

if condition

then

command

else

command

fi

fi

**WHILE LOOP**

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

## Syntax

while command

do

Statement(s) to be executed if command is true

done

Here the Shell *command* is evaluated. If the resulting value is *true*, given *statement(s)* are executed. If *command* is *false* then no statement will be executed and the program will jump to the next line after the done statement.

## Example

Here is a simple example that uses the **while** loop to display the numbers zero to nine −

a=0

while [ $a -lt 10 ]

do

echo $a

a=`expr $a + 1`

done

Upon execution, you will receive the following result −

0

1

2

3

4

5

6

7

8

9

Each time this loop executes, the variable **a** is checked to see whether it has a value that is less than 10. If the value of **a** is less than 10, this test condition has an exit status of 0. In this case, the current value of **a** is displayed and later **a** is incremented by 1.

**FOR LOOP**

The **for** loop operates on lists of items. It repeats a set of commands for every item in a list.

## Syntax

for var in word1 word2 ... wordN

do

Statement(s) to be executed for every word.

done

Here *var* is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

## Example

Here is a simple example that uses the **for** loop to span through the given list of numbers −

#!/bin/sh

for var in 0 1 2 3 4 5 6 7 8 9

do

echo $var

done

Upon execution, you will receive the following result −

0

1

2

3

4

5

6

7

8

9

**OVERVIEW OF UNIX COMMAND LINE ARGUMENTS:**

The Unix shell is used to run commands, and it allows users to pass run time arguments to these commands.

While running a command, the user can pass a variable number of parameters in the command line.

Within the command script, the passed parameters are accessible using ‘positional parameters’.  These range from $0 to $9, where $0 refers to the name of the command itself, and $1 to $9 are the first through to the ninth parameter, depending on how many parameters were actually passed.

**Example:**

***$ sh hello how to do you do***

Here $0 would be assigned sh

$1 would be assigned hello

$2 would be assigned how

And so on …

We will now look at some additional commands to process these parameters.

**set**

This command can be used to set the values of the positional parameters on the command line.

**Example:**

$ set how do you do

$ echo $1 $2

how do

Here, “how” was assigned to $1 and “do” was assigned to $2 and so on.

**LAB WORK OF QNO-1**

**WRITE A SHELL SCRIPT TO PRINT THE COMMAND LINE ARGUMENT IN REVERSE ORDER.**

Let's define our arguments:

$ set -- one two three

Now, let's print them out in reverse order:

$ for ((i=$#;i>=1;i--)); do echo "${!i}"; done

three

two

one

How it works

for ((i=$#;i>=1;i--)) starts a loop in which i counts down from $# to 1. For each value of i, we print the corresponding positional parameter by ${!i}. The construct ${!i} uses *indirection*: instead of returning the value of i, ${!i} returns the value of the variable whose name is $i.

As a script

Or

if [ $# -eq 0 ]

then

        echo "$0 num1, num2, numN"

        exit 1

fi

x=""

echo -n "Numbers are : "

for n in $@

do

  echo -n $n

  echo -n " "

  x="$n $x"

done

echo ""

echo -n "Reverse order: "

echo $x

**LAB WORK OF QNO-2**

**Write a Shell script to check whether a given number is palindrome or not.**

clear

echo Enter the number

read n

number=$n

reverse=0

while [ $n -gt 0 ]

do

a=`expr $n % 10 `

n=`expr $n / 10 `

reverse=`expr $reverse \\* 10 + $a`

done

echo $reverse

if [ $number -eq $reverse ]

then

echo &quot;Number is palindrome&quot;

else

echo &quot;Number is not palindrome&quot;

fi

**Write a Shell script to Reverse a Number .**

echo &quot;Enter a Number:&quot;

read a

rev=0

sd=0

or=$a

while [ $a -gt 0 ]

do

        sd=`expr $a % 10`

        temp=`expr $rev \\* 10`

        rev=`expr $temp + $sd`

        a=`expr $a / 10`

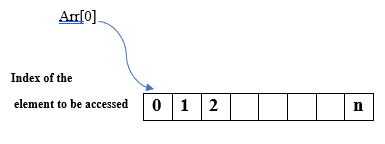
done

echo &quot;Reverse of $or is $rev&quot;

## Introduction to Array in Unix

Array is a Data structure that contains a group of elements that are typical of the same data type such as string, integer, float, etc. Array data structure is also available in Unix.

* **Array in Unix:**It is the collection of elements that may or may not be of the same datatype.
* **Index:**Array is zero-based, which means elements of the array are referenced by the index starting zero.
* **Size:**There is no maximum limit on the size of the array



tax of Array in Unix

There are different ways of forming an array in shell scripting. Let us go through each one of them in details:

**1. Indirect Declaration:**Here value is assigned for a particular index on the go. Example of which is mentioned below.

**Syntax:**

array\_name[index] = value

**2. Explicit Declaration:**First, the array is declared and then later the values are assigned to it. Declare is built-in keyword and -a is an option of reading built-in which allows reading and assigning values.

**Syntax:**

declare -a array\_name

**3. Compound Assignment:**Here, the array is declared with multiple values at a time.

**Syntax:**

array\_name = (value1 value2 value3 . . . valueN)

**Or**

array\_name = ([0]=value1, [1]=value2,[2]=value3..)

Here the index is optional if the index is not provided the value is assigned to the last index plus one

Arr = (1 2 3 4 5)  
for i in “${Arr[@]}”  
do  
echo $i  
done

**Output**:

Array in Unix eg9

**LAB WORK QNO-3**

**Write a shell script to sort the given array elements in ascending order using bubble sort**

# using Bubble sort

# Static input of Array

arr = (10 8 20 100 12)

echo "Array in original order"

echo ${arr[\*]}

# Performing Bubble sort

for ((i = 0; i<5; i++))

do

    for((j = i; j<5-i-1; j++))

    do

        if ((${arr[j]} > ${arr[$((j+1))]}))

        then

            # swap

            temp = ${arr[$j]}

            arr[$j] = ${arr[$((j+1))]}

            arr[$((j+1))] = $temp

        fi

    done

done

echo "Array in sorted order :"

echo ${arr[\*]}

Output :

Array in sorted order :

8 10 12 20 100

**LAB WORK QNO-4**

**WRITE A SHELL SCRIPT TO PERFORM SEQUENTIAL SEARCH ON AGIVEN ARRAY ELEMENT**

clear

echo enter number of elements

read n

k=0

echo enter array elements

for((i=1;i<=n;i++))

do

read arr[$i]

done

echo the array

for((i=1;i<=n;i++))

do

echo ${arr[$i]}

done

echo enter the element you want to search

read a

for((i=1;i<=n;i++))

do

if [ ${arr[$i]} -eq $a ]

then

k=1

break

fi

done

if [ $k -eq 1 ]

then

echo element found at position $i

else

echo element not found

fi

**OUTPUT:**

1)[admin@localhost ~]$ sh lsearch.sh

enter number of elements

4

enter array elements

67

90

23

45

the array

67

90

23

45

enter the element you want to search

23

element found at position 3

1)[admin@localhost ~]$ sh lsearch.sh

enter number of elements

4

enter array elements

23

78

56

44

the array

23

78

56

44

enter the element you want to search

98

element not found

**LAB WORK QNO-5**

**WRITE A SHELL SCRIPT TO PERFORM BINARY SEARCH ON AGIVEN ARRAY ELEMENTS**

**Write a shell script to do Binary Search**

**Shell Script:**

clear

echo -n "number of elements enter:"

read n

echo enter elements

for((i=1;i<=$n;i++))

do

echo -n "enter element $i :"

read arr[$i]

done

echo -n "enter the element you want to search:"

read item

lb=1

ub=$n

while [ $lb -le $ub ]

do

x=`expr $ub + $lb`

mid=`expr $x / 2`

if [ $item -lt ${arr[$mid]} ]

then

ub=`expr $mid - 1`

elif [ $item -gt ${arr[$mid]} ]

then

lb=`expr $mid + 1`

elif [ $item -eq ${arr[$mid]} ]

then

echo $item found at position $mid

break

fi

done

if [ $lb -gt $ub ]

then

echo not found

fi

**OUTPUT:**

[admin@localhost ~]$ sh binarysearch.sh

number of elements enter:5

enter elements

enter element 1 :23

enter element 2 :56

enter element 3 :78

enter element 4 :89

enter element 5 :90

enter the element you want to search:89

89 found at position 4

**CASE STAEMENT**

The basic syntax of the **case...esac** statement is to give an expression to evaluate and to execute several different statements based on the value of the expression.

The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

case word in

pattern1)

Statement(s) to be executed if pattern1 matches

;;

pattern2)

Statement(s) to be executed if pattern2 matches

;;

pattern3)

Statement(s) to be executed if pattern3 matches

;;

\*)

Default condition to be executed

;;

esac

Here the string word is compared against every pattern until a match is found. The statement(s) following the matching pattern executes. If no matches are found, the case statement exits without performing any action.

There is no maximum number of patterns, but the minimum is one.

When statement(s) part executes, the command ;; indicates that the program flow should jump to the end of the entire case statement. This is similar to break in the C programming language.

**LAB WORK-8**

**WRITE A SHELL SCRIPT TO ILLUSTRATE THE CASE STATEMENT**

FRUIT="kiwi"

case "$FRUIT" in

"apple") echo "Apple pie is quite tasty."

;;

"banana") echo "I like banana nut bread."

;;

"kiwi") echo "New Zealand is famous for kiwi."

;;

esac

OUT PUT

New Zealand is famous for kiwi.

**FILE PERMISSION**

File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

* **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
* **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
* **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

## The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows

$ls -l /home/amrood

-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile

drwxr-xr--- 1 amrood users 1024 Nov 2 00:10 mydir

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

* The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
* The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
* The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

## File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

### Read--Grants the capability to read, i.e., view the contents of the file.

### Write--Grants the capability to modify, or remove the content of the file.

### Execute--User with execute permissions can run a file as a program.

## Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

### Read--Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

### Write--Access means that the user can add or delete files from the directory.

### Execute--Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

## Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

### Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

|  |  |
| --- | --- |
| **Sr.No.** | **Chmod operator & Description** |
| 1 | **+**  Adds the designated permission(s) to a file or directory. |
| 2 | **-**  Removes the designated permission(s) from a file or directory. |
| 3 | **=**  Sets the designated permission(s). |

Here's an example using **testfile**. Running **ls -1** on the testfile shows that the file's permissions are as follows −

$ls -l testfile

-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

$chmod o+wx testfile

$ls -l testfile

-rwxrwxrwx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod u-x testfile

$ls -l testfile

-rw-rwxrwx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod g = rx testfile

$ls -l testfile

-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile

Here's how you can combine these commands on a single line −

$chmod o+wx,u-x,g = rx testfile

$ls -l testfile

-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile

## Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

|  |  |  |
| --- | --- | --- |
| **Number** | **Octal Permission Representation** | **Ref** |
| **0** | No permission | --- |
| **1** | Execute permission | --x |
| **2** | Write permission | -w- |
| **3** | Execute and write permission: 1 (execute) + 2 (write) = 3 | -wx |
| **4** | Read permission | r-- |
| **5** | Read and execute permission: 4 (read) + 1 (execute) = 5 | r-x |
| **6** | Read and write permission: 4 (read) + 2 (write) = 6 | rw- |
| **7** | All permissions: 4 (read) + 2 (write) + 1 (execute) = 7 | rwx |

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

$ls -l testfile

-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

$ chmod 755 testfile

$ls -l testfile

-rwxr-xr-x 1 amrood users 1024 Nov 2 00:10 testfile

$chmod 743 testfile

$ls -l testfile

-rwxr---wx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod 043 testfile

$ls -l testfile

----r---wx 1 amrood users 1024 Nov 2 00:10 testfile

LAB WORK-6

**Write a shell script to accept any two file name and check their permission**

echo -n "Enter file name : "

read file

# find out if file has write permission or not

[ -w $file ] && W="Write = yes" || W="Write = No"

# find out if file has excute permission or not

[ -x $file ] && X="Execute = yes" || X="Execute = No"

# find out if file has read permission or not

[ -r $file ] && R="Read = yes" || R="Read = No"

echo "$file permissions"

echo "$W"

echo "$R"

echo "$X"

echo -n "Enter file name : "

read file1

# find out if file has write permission or not

[ -w $file1 ] && W="Write = yes" || W="Write = No"

# find out if file has excute permission or not

[ -x $file1 ] && X="Execute = yes" || X="Execute = No"

# find out if file has read permission or not

[ -r $file1 ] && R="Read = yes" || R="Read = No"

echo "$file1 permissions"

echo "$W"

echo "$R"

echo "$X"

**Lab work -12**

**Write a Shell script to read a file name and change the existing permission**

#!/bin/bash

if [ "$1" = "" ]; then

echo "usage: $0 [filename]"

exit 1

fi

if ! test -f $1

then

echo "Invalid file name"

exit 1

fi

chmod u+x $1

**Labwork-10**

10. Write a Shell script to demonstrate Terminal locking.

echo "Enter a password"  
read pass1   
echo "Renter password"  
read pass2  
if [ $pass1 = $pass2 ]  
then  
    echo "Keyboard is locked"  
    trap " " 20 30 15 9 2 1 3  
    echo "Enter your password to unlock the screen"  
    read upass  
    while [ "$pass2" != "$upass" ]  
    do  
        echo "password is wrong !! "  
        echo "Renter password"  
        read upass  
    done  
else  
    echo "Password doesn't match"  
fi

**Labwork-11&13**

11. Write a Shell script to accept the valid login name, if the login name is valid then print its home directory else an appropriate message.

Clear  
if [ $# -eq 0 ]  
then  
echo "No command line argument passed"  
exit  
fi  
while [ $1 ]  
do  
cat /etc/passwd | cut -d ":" -f1 | grep "^$1" > temp  
ck=`cat temp`  
if [ "$ck" != "$1" ]  
then  
echo "ERROR:$1 is an invalid login name"  
else  
echo "Home Directory for $1 is"  
echo `cat /etc/passwd | grep "^$1" | cut -d ":" -f6`  
fi  
shift  
done

13. Write a shell script to display the calendar for current month with current date replaced by \* or \*\* depending on whether the date has one digit or two digits.

set `date`  
y=$3  
if [ $y -le 9 ]  
then  
cal |sed "s/$3/\*/"  
else  
cal |sed "s/$3/\*\*/"  
fi