Experiment No. 2

2. Shell Programming.

INTRODUCTION:
Shell programming is a grouping of commands 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.

01. SHELL KEYWORDS:
echo, read, if fi, else, case, esac, for , while , do , done, until , set, unset, readonly, shift, export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.

02. General things SHELL

The shbang line The "shbang" line is the very first line of the script and lets the kernel know what shell will be interpreting the lines in the script. The shbang line consists of a #! followed by the full pathname to the shell, and can be followed by options to control the behavior of the shell.
E*XAMPLE*

```
#!/bin/sh
```

Comments Comments are descriptive material preceded by a # sign. They are in effect until the end of a line and can be started anywhere on the line.
E*XAMPLE*

```
# this text is not
# interpreted by the shell
```

Wildcards There are some characters that are evaluated by the shell in a special way. They are called shell metacharacters or "wildcards." These characters are neither numbers nor letters. For example, the *, ?, and [ ] are used for filename expansion. The <, >, 2>, >>, and | symbols are used for standard I/O redirection and pipes. To prevent these characters from being interpreted by the shell they must be quoted.

E*XAMPLE*

```
Filename expansion:
rm *; ls ??; cat file[1-3];
Quotes protect metacharacter:
echo "How are you?"
```

03. SHELL VARIABLES:

Shell variables change during the execution of the program .The C Shell offers a command "Set" to assign a value to a variable.
For example:

```
% set myname= Fred
% set myname = "Fred Bloggs"
% set age=20
```

A $ sign operator is used to recall the variable values.
For example:

```
% echo $myname will display Fred Bloggs on the screen
```

A @ sign can be used to assign the integer constant values.
For example:

```
%@myage=20
%@age1=10
%@age2=20
%@age=$age1+$age2
%echo $age
```

List variables

```
% set programming_languages= (C LISP)
% echo $programming _languages
C LISP
% set files=*.*
% set colors=(red blue green)
% echo $colors[2]
blue
% set colors=($colors yellow)/add to list
```

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.

04. EXPRESSION Command:
To perform all arithmetic operations .
Syntax:

```
Var = „expr$value1" + $ value2"
```

Arithmetic The Bourne shell does not support arithmetic. UNIX/Linux commands must be used to perform calculations.
E*XAMPLE*

```
n=`expr 5 + 5` echo $n
```

Operators The Bourne shell uses the built-in test command operators to test numbers and strings.
E*XAMPLE*

Equality:

=	*string*
!=	*string*
-eq	*number*
-ne	*number*

Logical:

-a	*and*
-o	*or*
!	*not*

Logical:

AND	&&
OR	||

Relational:

-gt	*greater than*
-ge	*greater than, equal to*
-lt	*less than*
-le	*less than, equal to*

Arithmetic:

+, -, *, /, %

Arguments (positional parameters) Arguments can be passed to a script from the command line. Positional parameters are used to receive their values from within the script.
E*XAMPLE*

```
At the command line:
$ scriptname arg1 arg2 arg3 ...
In a script:
echo $1 $2 $3          Positional parameters
echo $*                All the positional paramters
echo $#                The number of positional parameters
```

05. READ Statement:
To get the input from the user.
Syntax:

```
read x y
(no need of commas between variables)
```

06. ECHO Statement:
Similar to the output statement. To print output to the screen, the echo command is used. Wildcards must be escaped with either a backslash or matching quotes.
Syntax:

Echo “String” (or) echo $ b(for variable).

E*XAMPLE*

```
echo "What is your name?"
```

Reading user input: The read command takes a line of input from the user and assigns it to a variable(s) on the right-hand side. The read command can accept multiple variable names. Each variable will be assigned a word.
E*XAMPLE*

```
echo "What is your name?"
read name
read name1 name2 ...
```

6. 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.
Syntax:

1. 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.

2. if…else

    Syntax:

```
If cp $ source $ target
Then
Echo File copied successfully
Else
Echo Failed to copy the file.
```

3. nested if

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

Syntax:

```
if condition
then
        command
        if condition
        then
                command
        else
                command
        fi
fi
```

4.case ….. esac

This construct helps in execution of the shell script based on Choice.

EXAMPLE

```
case variable_name in
pattern1)
statements
;;
pattern2)
statements
;;
pattern3)
;;
*) default value
;;
Esac
```

07. LOOPS

There are three types of loops: while, until and for. The while loop is followed by a command or an expression enclosed in square brackets, a do keyword, a block of statements, and terminated with the done keyword. As long as the expression is true, the body of statements between do and done will be executed.

The until loop is just like the while loop, except the body of the loop will be executed as long as the expression is false.

The for loop used to iterate through a list of words, processing a word and then shifting it off, to process the next word. When all words have been shifted from the list, it ends. The for loop is followed by a variable name, the in keyword, and a list of words then a block of statements, and terminates with the done keyword.

The loop control commands are break and continue.

E*XAMPLE*

```
while command
do
        block of statements
done
------------

while [ expression ]
do
        block of statements
done
until command                   for variable in word1 word2 word3 ...
do                              do
        block of statements block of statements
done done
------------

until [ expression ]
do
        block of statements
done
------------

until control command
do
        commands
done
```

08. Break Statement:

This command is used to jump out of the loop instantly, without waiting to get the control command.

09. ARRAYS

(Positional parameters)

The Bourne shell does support an array, but a word list can be created by using positional parameters. A list of words follows the built-in set command, and the words are accessed by position. Up to nine positions are allowed.The built-in shift command shifts off the first word on the left-hand side of the list. The individual words are accessed by position values starting at 1.

E*XAMPLE*

```
set word1 word2 word3
echo $1 $2 $3                   Displays word1, word2, and word3
set apples peaches plums
shift                           Shifts off apples
echo $1                         Displays first element of the list
echo $2                         Displays second element of the list
echo $*                         Displays all elements of the list
```