UNIX PROGRAMMING (JNTUK-R16)
UNIT-5: SHELL PROGRAMMING

SYLLABUS:

- Shell Variables
- The export Command
- The .profile file a Script Run During Starting
- The First Shell Script
- The read command
- Positional Parameters
- The $? variable knowing the exit Status
- More about the set Command
- The exit Command
- Branching Control Structures
- Loop Control Structures
- The continue and break Statement
- The expr Command
- Performing Integer Arithmetic-Real Arithmetic in Shell Programs
- The here Document(<<)
- The sleep Command
- Debugging Scripts
- The script Command
- The eval Command
- The exec Command.

Text Books.

The Unix programming Environment by Brain W. Kernighan & Rob Pike, Pearson.
Introduction to Unix Shell Programming by M.G.Venkateshmurthy, Pearson.

1. Shell Variables:

Refer UNIT-3 TOPIC-7

2. The export command:

- With many shells, including bourne and korn shells, values of the variables set or changed in one program will not be available to other programs.
- It is possible to make the values available across all programs or processes by using the export command.

Example: (In Bourne Shell)

```
$TERM=vt100
$export TERM
$
```

Example: (In Korn Shell)
$export TERM=vt100
$

- After the example, check the TERM value by echo command

Example:
$echo $TERM
vt100
$

- Using the option -f, Names are exported as functions

Example:
$name () { echo "hello"; }
$export -f name
$name
hello

3. The .profile file (A script running during starting):

- Every user has a .profile of his or her own
- This file is a shell script that will be present in the home directory of the user
- As this file resides in the home directory, it gets executed as soon as the user logs in
- The system administrator provides each user with a profile that will be just sufficient to have a minimum working environment
- This file is automatically executed on login and called as AUTOEXEC.BAT file of Unix

Example contents of a typical .profile file

```
$cat .profile
# user $HOME/.profile – commands executed at login time
HOME=/home/mgv/programs
PATH=$PATH:$HOME/bin:/usr/bin/X11:/usr/hosts:.
MAIL=/usr/spool/mail/$LOGNAME                    #mailbox location
IFS=

PS1="$"
PS2=">"
echo "Today's date is `date` "
news
calendar
echo "You are now in the $HOME directory"
$
```

4. The First Shell Script:

- A Shell Script or Shell Program is a set of commands that are executed together as a single unit
- A Shell script also includes
    - Commands for selective execution (Control commands)
    - Commands for I/O operations like read and echo commands
    - Commands for repeated execution (loop control statements)
    - Shell Variables and so..on
- A Shell Script is named just like all other files.
- It uses .sh extension
- A shell program runs in the interpretive mode

Example:

```
$vi  first.sh
echo  "Hello World"
echo  "Welcome to UNIX Shell Programming"
echo  Today the date is 'date'
```

Executing a Shell Script:

- A Shell Script is executed in two ways
    - Using sh command (Like Internal Command)
    - Using chmod for execute permissions and filename directly for execution (Like External Command)

Execution1: (As an Internal Command)

```
$sh  first.sh
Hello World
Welcome to UNIX Shell Programming
Today the date is Mon  Aug 27  11:58:00  IST  2018
```

Execution2: (As an External Command)

```
$chmodu+x  first.sh
$./first.sh
Hello World
Welcome to UNIX Shell Programming
Today the date is Mon  Aug 27  11:58:00  IST  2018
```

Comments:

- Comments are used to explain
    - The purpose and logic of the program and
    - Commands used in the program
- The # symbol is used to represent the comment lines in the shell script

Example:

```
# This is a comment line
# This is my first shell program
```

5. The read command:

- The read command is used to give input to a shell program
- This command reads just one line and assigns this line to one or more shell variables

Example:

```
$vi  inputs.sh
echo "Enter Value for n"
read n
echo "The n value is $n"
echo "Enter a name"
read name
echo "The name is $name"
```

Execution:

```
Enter Value for n
10
The n value is 10
Enter a name
india
The name is india
```

Multiple Arguments:

- The read command can take multiple arguments.
- In other words, values for more than one variable can be input or assigned, using a single read command

Example:

```
$read  a  b  c
```

- Arguments are separated by space
- If number of input values are less than the number of arguments, then the arguments or variables to which values are not input will be initiated to null.
- If number of input values are more than the number of arguments, then first (n-1) values are assigned to the first (n-1) arguments and all the remaining input values are assigned to the n^{th} argument

The readonly function (Read-only variables):

- These variables uses readonly( ) function.
- The values of variables which can only be read but not to be manipulated are called read-only variables.

Example:

```
$cat example
echo Enter value for x
read x
echo value of x is $x
readonly  x
```

```
x='expr  $x+1'
echo The value of x now is $x
$
```

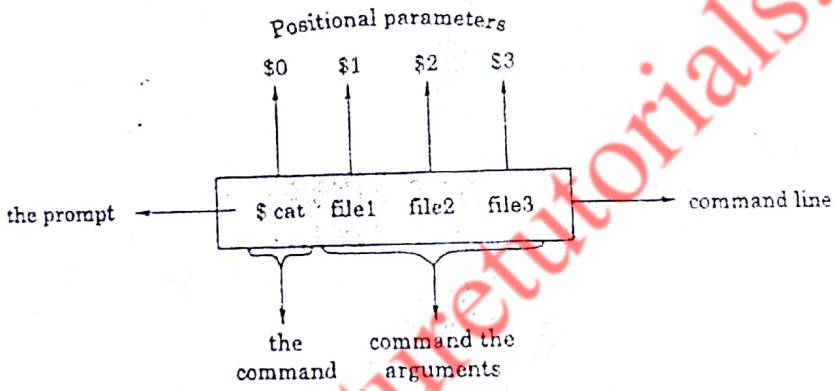
Execution:

```
$sh example
Enter value for x
4
value of x is 4
example: line 5 : x : readonly variable
$
```

6. Positional Parameters:

- Arguments submitted with a shell script are called positional parameters
- The first argument is parameter no 1 and second argument is parameter no 2 and so on.
- Actually these parameters are stored in certain special shell variables
- There are 9 such variables represented as $1, $2, $3, $4, $5, $6, $7, $8, $9
- Depending upon their physical positions in the command line they are called Positional Parameters

Example:



The S0 variable:

- It holds parameter number 0
- It is always representing a command or program to be executed

S#, S* and S@ variables:

- The $# variable holds a count of the total number of parameters, that is, arguments
- The $* variable holds the list of all the arguments
- It may be observed that the $# variable is similar to argc and the $* variable is similar to argv[ ] in C
- Like $*, the $@ variable also holds the list of all the arguments
- When $* and $@ are used within quotes, the contents of $* is considered as a single string whereas the contents of $@ are considered as independently quoted and considered as independent string arguments.

Difference between $* and $@:

- Let the files fa, fb and fc are present in the current working directory

Example:

```
$cat  sample.sh
ls  "$*"
ls  "$@"
$
```

Execution:

```
$sh  sample.shfafb  fc
ls:fafbfc:No such file or directory
fafb  fc
$
```

Understanding Positional Parameters using shell program:

Example:

```
$cat  positional.sh
echo Command or Program name is $0
echo The number of arguments are $#
echo  The arguments are $*
echo  The first argument is $1
echo  The second argument is $2 and so on
$
```

Execution:

```
$sh  positional.sh  a1  a2  a3
Command or Program name is positional.sh
echo The number of arguments are 3
echo  The arguments are a1  a2  a3
echo  The first argument is a1
echo  The second argument is a2 and so on
$
```

The set command (Assigning values to positional parameters):

- Positional parameters assigned using the set command

Example:

```
$set  India is my country
$
```

This line assigns

- India as $1
- is as $2
- my as $3
- country as $4

Example:
$echo $1 $4
india country

The shift command (Handling Excess Command Line Arguments):
- Only a maximum of nine arguments are given in a command line
- In case more than nine arguments are given in a command line, no error is indicated
- Such type of situations are handled by using shift command
- This command is used to shift the position of the positional parameters

Example:
$set hello good morning how do you do welcome to Unix programming
shift 3
echo $1 $2 $3 $4
how do you do

7. The $? variable (Knowing the exit status):
- Whenever a command is successfully executed the program returns a 0 (zero)
- If a command is not executed successfully a value other than 0 will be returned.
- Logically, a 0 is considered as true and a non-zero is considered as false.
- These returned values are called program exit status.
- This will be available with the shell's special variable called $?
- An exit status value available in $? is normally used in decision making in shell programs.

Example:
$cat sample.sh
cat: sample.sh: No such file or directory
$echo $?
1
$

8. More about the set Command:
i) The set command without arguments:
- When this command used without arguments displays the contents, the system variables that are either local or exported.

Example:
$set
CDPATH=:/users/mgv:/usr/spool
EDITOR=/bin/vi
HOME=/usr/mgv
IFS=
MAIL=/usr/spool/mail/$LOGNAME
PATH=/usr/local/sbin: /usr/sbin:/usr/bin/X11
.....
$

ii) The set command with options:

- Many options such as –x, -v, -- and others are allowed to be used with this command.
- The options –x and –v are used to debug shell scripts

iii) The set command and the – option:

- Under certain circumstances arguments to the set command are passed on through command substitution.
- Sometimes this method is error-prone.
- Such a situation is handled by using the special option – (double hyphen)

Example:

```
$ls  -l  myfile
 -rwxr-xr-x  myfile
$
$set  `ls  -l  myfile`
 -rwxr-xr-x : bad options
$
```

- In this situation, the – option is handled as,

```
$set  --  `ls  -l  myfile`
```

9. The exit command:

- This command is used to terminate the execution of a script.
- This is also used to exit from the shell currently using.
- It is not necessary to use this command at the end of every shell script
- The shell recognizes end of the script automatically
- This command can optionally use a numeric argument
- A 0 (zero) exit value indicates success and a non-zero value indicates failure
- If no arguments are used, this command returns a 0 (zero)

Example1:

```
$exit    #exit from the shell
```

Example2:

```
$cat  >  sample1
echo  "Success"
exit  0
```

Example3:

```
$cat  >  sample2
echo  "Failure"
exit  1
```

10. Operators in UNIX:

- An operator is a symbol that is used for arithmetic and logical manipulations
- Some of the operators are represented as Options instead Symbols
- The different operators in UNIX are,
  - i) Arithmetic operators
  - ii) Logical operators
  - iii) Relational operators

i) Arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	. /
Modulus	%

- The multiplication operator is used as an escape character because * is a wildcard character

ii) Logical operators

- Logical AND and OR operators are represented as options

AND	-a
OR	-o
NOT	!

iii) Relational operators:

- All relational operators are represented as options

>	-gt
>=	-ge
<	-lt
<=	-le
==	-eq
!=	-ne

11. Branching Control Structures:

- Program structures that are used to shift the point of execution are called Branching Control Structures
- These statements are also called as Selection Statements or Conditional statements or Decision-making Statements
- These are two types
  - i) Two-way Selection (if, if-then-else)
  - ii) Multi-way Selection (if-elif-else, case)

i) Two-way Selection:

- The two-way selection statements are
  - a) if statement
  - b) if-then-else statement

a) if statement:

Syntax:

```
if [ test_expression ]
then
true-block
fi
```

Example:

```
if [ $a -eq $b ]
then
echo "a and b are equal"
fi
```

b) if-then-else statement:

Syntax:

```
if [ test_expression ]
then
true-block
else
false-block
fi
```

Example:

```
if [ $a -eq $b ]
then
echo "a and b are equal"
else
echo "a and b are not equal"
fi
```

ii) Multi-way Selection:

- The Multi-way selection statements are
  - a) if-elif-else statement
  - b) case statement

a) if-elif-else statement:

Syntax:

```
if [ test_expression ]
then
command(s)
elif [test_expression ]
command(s)
else
command(s)
fi
```

Example:

```
if [ $a -eq $b ]
echo " a and b are equal"
elif [ $a -gt $b ]
echo " a is greater than b"
else
echo " a is less than b"
fi
```

b) The case statement:

Syntax:

```
case string-value in
pattern-1) command(s) ;;
pattern-2) command(s) ;;
.....
.....
pattern-n) command(s) ;;
*) echo "None of the patterns matched" ;;
esac
```

Example:

```
case $digit in
echo "ZERO" ;;
echo "ONE" ;;
echo "TWO" ;;
echo "THREE" ;;
echo "FOUR" ;;
echo "FIVE" ;;
echo "SIX" ;;
echo "SEVEN" ;;
echo "EIGHT" ;;
echo "NINE" ;;
*) echo "NOT A VALID NUMBER" ;;
esac
```

12. The test command:

- This is a built-in shell command that evaluates the expression given to it as an argument.
- It returns true if the evaluation of the expression returns a 0 (zero)
- It returns false if the evaluation of the expression returns a non-zero
- With this command, there are 3 types of tests carried out
    - i) Numeric Tests
    - ii) String Tests
    - iii) File Tests

i) Numeric Tests:

- In Numeric tests, two numbers are compared using relational operators

Example:

```
$x=5; y=7
$test $x -eq $y; echo $?
1                                   # because test failed
$[ $x -lt $y ]; echo $?
0
                                    # because test succeed
```

ii) String Tests:

- String tests are conducted for checking
  - equality of strings
  - non-equality of strings
  - zero or non-zero length of a string and so on.

Example:

```
$ans=y
$[ "$ans" = "y" ]; echo $?
0
$[ "$ans" != "y" ]; echo $?
1
$[ -n "$ans" ]; echo $?
0
$[ -z "$ans" ]; echo $?
1
```

iii) File Tests

- File Tests are conducted for checking the status of files and directories
- Using these tests one can find out the type of a file and permissions granted or not granted to it

Test	Exit status
-e file	True if file exists
-f file	True if file exists and is a regular file
-r file	True if file exists and is readable
-w file	True if file exists and is writable
-x file	True if file exists and is executable
-d file	True if file exists and is a directory
-c file	True if file exists and is a character special file
-b file	True if file exists and is a block special file
-h file	True if file exists and is a link file
-s file	True if file exists and has a size greater than zero (0)

Example:

```
$ls -l sample
-rw-r--r--  sample
$[ -f sample ]; echo $?
0                                   #regular file
```

```
$[ -x sample ]; echo $?
1                              #no execute permission
```

13. Loop Control Structures:

Refer UNIT-3 TOPIC-8

14. Unconditional statements:

- The unconditional statements in Unix are,
  - i) break statement
  - ii) continue statement

i) break statement:

- The break statement is used to terminate from loop or block

Syntax:

```
break
```

Example:

```
for a in 0 1 2 3 4
do
if [ $a -eq 2 ]
then
break
fi
echo $a
done
```

Output:

```
0
1
```

ii) continue statement:

- The continue statement is similar to the break statement, except it causes the current iteration of the loop to exit, rather than the entire loop

Syntax:

```
continue
```

Example:

```
for a in 0 1 2 3 4
do
if [ $a -eq 2 ]
then
continue
fi
echo $a
done
```

Output:
0
1
3
4

15. The expr command:

- This command is used to carry out basic arithmetic operations including modulo division on integers
- This command is used only when arithmetic operations are simple and are few.
- The expr command works only on integers.
- For complex arithmetic operations we can use Unix calculators like bc.
- The arithmetic operators are

Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%

- While using this command one must remember that
  - On either sides of the arithmetic operators a blank or tab must be present
  - The operator * must be escaped, otherwise shell treats it as a wildcard

Integer Arithmetic:

- The both operands in an arithmetic operation are integers then that operation is called Integer Arithmetic

Example:

```
$x=3; y=5
$expr 3 + 5
8
$expr $x - $y
-2
$expr 3 \* 5
15
$expr $y / $x
1
$expr 13 % 5
3
$
```

- The expr command is often used with command substitution to assign values to variables

Example:

```
$x=6; y=2; z=`expr $x + $y`
$x=`expr $x + 1`                    #implementation of x++
```

- The expr command also perform certain string manipulations such as,
  - To determine the length of a given string
  - To extract a sub-string from a given string
  - To locate the position of a character in a string

Example:

```
$expr "india" : `.*`
```

16. Real Arithmetic in Shell Programs:

- The both operands in an arithmetic operation are Real then that operation is called Real Arithmetic
- The expr command works only on integers.
- Real arithmetic can be managed using the bc (basic calculator) command along with the scale function and the echo command
- The output of the arithmetic expression is piped to the bc command

Example:

```
$c=`echo $a + $b | bc`
```

- Because of piping, echo does not display its output, rather it will redirect its output to the bc command

Shell Script for Area of Triangle:

```
$vi tarea.sh
echo "Enter base and height"
read base height
tarea=`echo "scale=2; 1/2*$base*$height" | bc`
echo "area of triangle=$tarea"
```

Execution:

```
$sh tarea.sh
Enter base and height
2
2
area of triangle=2.00
```

Shell Script for converting Fahrenheit to Celcius:

```
$vi f2c.sh
echo "Enter Fahrenheit value"
read f
c=`echo "scale=2; 5/9*($f-32)" | bc`
echo "Fahrenheit to Celsius=$c"
```

Execution:

```
$sh f2c.sh
Enter Fahrenheit value
100
Fahrenheit to Celsius=37.40
```

Shell Script for Gross Salary of an Employee:

```
$vi  salary.sh
echo  "Enter Basic Salary"
read  basic
if [ $basic -lt 1500 ]
then
 hra=`echo "scale=2;  $basic*10/100" | bc`
 da=`echo "scale=2;  $basic*90/100" | bc`
else
 hra=500
 da=`echo "scale=2;  $basic*98/100" | bc`
fi
gsalary=`echo "scale=2;  $basic+$hra+$da" | bc`
echo  "Gross Salary=$gsalary"
$
```

Execution:

```
$sh  salary.sh
Enter Basic Salary
1200
Gross Salary=2400.00
$sh  salary.sh
Enter Basic Salary
12250
Gross Salary=24755.00
```

17. The here document (<<):

- In Unix it is possible to include the document on which the system has to operate along with the command itself
- A here document is a way of getting text input into a script without having to feed it in from a separate file
- This type of redirection tells the shell to read input from the current source (HERE) until a line containing only word (HERE) is seen

Syntax:

```
command<< HERE
Text
Text
.....
HERE
```

Example1:

```
$ cat  << stop
> hi
```

```
> hello
> welcome
> stop
hi
hello
welcome
```

Example2:

```
$ wc -w << end
> welcome to the unix shell script programming
> end
7
```

18. The sleep command:

- Using this command, the user can make the system to sleep, that is, pause for some fixed period of time
- The sleep command suspends the calling process for at least the specified number of seconds (by default), minutes, hours or days.

Syntax:

```
$sleep number
```

Example1:

```
$sleep 60     #The system was sleeping for 60 seconds
$
```

Example2:

```
$sleep 3h     #The system was sleeping for 3 hours
```

19. Debugging Scripts:

- When a script does not work properly, we need to determine the location of the problem.
- The UNIX shells provide a debugging mode.
- Run the entire script in debug mode or just a portion of the script.
- Debug Statement Options are,

Option	Meaning
set -x	Prints the statements after interpreting metacharacters and variables
set +x	Stops the printing of statements
set -v	Prints the statements before interpreting metacharacters and variables
set -f	Disables file name generation (using metacharacters)

Example:

```
$cat > debug1.sh
 set -x
 echo "How many number of times the message is to be displayed?"
 read count
 until [ $count -eq 0 ]
 do
  echo $*
  count=$(expr $count - 1)
 done
 set +x
```

Execution:

```
 $ sh debug2.sh HELLO
 + echo 'How many number of times the message is to be displayed?'
 How many number of times the message is to be displayed?
 + read count
 2
 + '[' 2 -eq 0 ']'
 + echo HELLO
 HELLO
 ++ expr 2 - 1
 + count=1
 + '[' 1 -eq 0 ']'
 + echo HELLO
 HELLO
 ++ expr 1 - 1
 + count=0
 + '[' 0 -eq 0 ']'
 + set +x
```

20. The script command:

- This command is used to record an interactive session
- It is invoked either without any argument or with a filename as its argument

Syntax1: (without argument)

```
$script
 script started, file is typescript
```

- Here typescript is the filename by default.

Syntax2: (filename as argument)

```
$script record123
 Script started, filename is record123
```

- Once the script command is invoked, everything that is done at the terminal will be automatically recorded in the corresponding file
- Recording session is terminated using exit command

Syntax:

```
$exit  record123
$
```

- A record session may be displayed on the terminal using cat or more command
- A record session is printed using lp or lpr command
- A session can be appended to an existing file using the append option - a

Syntax:

```
$script  -a
$
```

21. The eval command:

- The use of the eval command makes the shell to scan the command line once more, i.e., second time and then actually executes the command line
- The eval command first evaluates the argument and then runs the command stored in the argument

Example1:

```
$ b=a
$ c=b
$ eval  echo  \$$c
  a
$
```

Example2:

```
$ a=10
$ b=a
$ eval echo '$'$b
  10
$
```

22. The exec command:

- This command has the following two abilities
  - Running a command without creating a new process
  - Redirecting standard input, output or error of a shell script from within the script
- The exec command replaces the current shell process with the specified command
- Normally, when you run a command a new process is spawned (forked).
- The exec command does not spawn a new process.
- Instead, the current process is overlaid with the new command.
- In other words the exec command is executed in place of the current shell without creating a new process.

- The command implements Unix exec system call.

Example1:

$exec date
Fri Aug 31 06:53:26 UTC 2018
login:

- The date program is overlaid into the process area of the current shell and then executed
- Finally the current process gets terminated and the control goes back into the login environment

Example2:

$exec vi myfile

- In this case we replaced the shell with the vi.
- The command places you in vi, so that you can edit myfile.
- When you exit vi you exit the system because vi has become your login interface to the operating system.
- This command can be used to close the standard input and reopen it with any file the user wants to read
- To change the standard input to a file called infile,

Example:

$exec < infile

- To change the standard output to a file called outfile,

Example:

$exec > outfile

Important Questions:

1. Explain about Positional Parameters
2. Explain about a) here document b) sleep command c) Script command
3. Explain about a) Debugging Scripts b) eval command c) exec command