Lecture 28: SHELL PROGRAMMING

Features of the Shell

- Command Interpreter
- Input / Output Redirection
- Filters and Pipes
- Wildcards
- Background Processing
- Shell as a Programming Language

- The Shell is the command interpreter of any UNIX system. It interprets the commands that the user gives at the prompt and sends them for execution to the kernel.
- The Shell is essential for interactive computing where the user desires instant output of -His/her commands.
- The Shell has the features of re-directing the standard input, output and error files to devices other than the standard devices.
- Using the pipe feature different commands can be combined to solve a particular problem, which is not possible through a single command. The Shell creates temporary files to hold the intermediate results and erases them once the command execution is over.
- The Shell has the capability of file name expansion, using Metacharacters or Wildcards, discussed earlier.
- More than one command can be given at the same line using the command terminator ";".
- The Multitasking feature of UNIX is supported by the shell using the background processing method, where more than one process can be started in the background.

Besides the features that are already discussed, the shell has many more facilities like defining and manipulating variables, command substitution, error handling, etc. All these together can be

Gomilitary.in

used as a programming language.

Different types of shells available in the UNIX system are the Bourne shell, the 'e' shell, and the Korn shell.

In this book we shall be concentrating on the programming features of the Bourne shell only.

Shell as a Programming Language

- Manipulation of variables
- Decision making
- Looping
- Parameter handling
- Handling interrupts

The Shell's capabilities do not end with it being a command interpreter. It is also a programming language that offers standard programming structures like loops, conditional branching of control, defining and manipulating variables, file creation and parameter passing.

This is possible by writing a **Shell script,** which is essentially a program file containing **UNIX** commands that are executed one after the other.

The Shell script is similar to the **batch files in DOS** but it is much more powerful and complete.

The main features of the shell programming language are:

- **Structured language construct** - implement programming language features like looping and decisions making.

- ***I/O* interaction** in the form of accepting values from the user and displaying the results.
- **Subroutines construct** to facilitate a modular approach to programming.
- **Variables**
- **Arguments** to control the execution on different values or files that are passed as arguments.
- **Interrupt handling** to receive signals and carry out alternate courses of action.

Creating and executing a shell scripts

Open a file in vi editor.

Write any UNIX command.

Save the file under a given name.

At the shell prompt give the command **sh** followed by the file name.

The command written in that file will be executed.

Gomilitary.in

Example: Creating and executing a shell script.

Open a file called firs_script using vi;

$ vi first_script

Enter the command Is -1 and save the file.

Execute the script by issuing the following command:

$ sh first_script

will produce the output of the command Is -I

Question:

Write the steps to execute the following commands as a shell script.

Is -1

cat emp.dat
Is -1 I grep "emp.dat"

Shell Variables

User defined variables : created by the user
Environmental variables : created by the shell
Pre-defined variables : created by the shell and commands
Using the variables
Accepting values to the variables

A variable is a name associated with a data value and it offers a symbolic way to represent and manipulate data. The most important function of shell variable is to customize the operations of the shell.

For example, using variables the user can establish a different shell prompt, specify a new home directory, assign different search paths for the commands or it can be used for shorthand notations for large command lines.

The variables in the **Bourne Shell** are classified as:

User defined variables : defined by the user for his/her use

Environmental variables : defined by the shell for its own operations

Pre-defined variables : reserved variables used by the shell and UN1X commands for specifying the exit status of commands, arguments to shell scripts, the formal parameters, etc.

The user defined variables are created by specifying the name of the variable followed by the assignment operator and the value of the variable at the prompt. No 'white space is allowed before or after the assignment operator.

$ variable=value

Example: creating user defined variables.

$ name=mano will create a variable name containing the value 'mano'

$ age=56 will create a variable age and store the characters "5" and "6".

The shell, by default, treats all the value as strings of characters only.
Computations on numeric variables are done in a different manner, to be discussed later.

Gomilitary.in

Environmental variables

PATH	: contains the search path string
HOME	: specifies the full path names for the user login directory
TERM	: holds the terminal specification information
LOGNAME	: holds the user login name
PS1	: stores the primary prompt string
PS2	: specifies the secondary prompt string
SHELL	: stores the name of the shell (Bourne, Korn or C)

The Shell maintains its own set of variables that are made available to each process as it begins execution. These are also called the environmental variables.

Some of the standard environmental variables found in many systems are:

PATH

contains the search path string. The commands given by the user are searched in the directories specified in the path string. An error message is displayed on failure.

HOME

specifies the full path name for the user's login directory. The cd command without any argument will look for the contents of this variable and change the directory accordingly.

TERM

holds the terminal specification. Being a UNIX system it can have different type of terminals. Typical entries found are vt100, vt220, ansi etc.

LOGNAME

holds the user login name.

PS1

stores the primary prompt string, which is the dollar sign ($). To change the prompt, simply assign the new value to this variable.

For example, $ PS1="hello" will change the prompt to 'hello'

PS2

specifies the secondary prompt string, which is displayed for the continuation of commands into the next line. Usually a greater than symbol (» is assigned to it.

SHELL

stores the name of the shell (Bourne, Korn or C). In Bourne shell, the entry sh is found in this variable:

Using variables

Displaying the contents of variables

$echo mesg $echo $variable

Escape sequences with echo command: \\b, \\f, \\n, \\r, \c

Reading values into variables

$read var

The echo command simply echoes back its arguments on to the terminal screen.

Gomilitary.in

For example:

$ echo welcome to RU

will produce the output welcome to RU

When the echo command writes its arguments, all Metacharacters are expanded by the shell.

$ echo *

will produce all the filenames present in the current directory as the output. This is a crude version of the Is command.

The echo command can also be used to display the contents of the shell variables.

$ echo HOME will display the string 'HOME'

If the argument to the echo command is prefixed by a dollar sign '$', it treats the argument as a variable and displays the contents of that variable. "

If a variable by that name is not found, then a blank line is echoed.

Example: The **echo** command

$ echo $HOME will display the value of the variable **HOME,** say, **/usr/mano**

$ Name=David variable **Name** is assigned a value **David**

$ echo $Name
David the output produced by **echo**
$
$ echo Hello $Name !, welcome to $HOME

the output may be:

Hello David I welcome to /usr/mano

The output of echo can be redirected and can be sent to a pipeline.

$ echo "A quick brown fox jumps over the lazy little dog" | wc
1 10 48 the output will be the number of lines, words and characters.

The escape sequences used with echo command to control the output are:

\\b Back space

\\f Form feed

\\n New line

\\r Carriage return

The echo command will exit immediately without printing a new line, if it encounters the n\c"' escape sequence.

The echo command is used as the 'print' statement in a shell script.

Reading values into variables

We can execute the echo command given in the previous example through a shell script. By doing so, it will display the same output each time we execute the script. If we want to write an interactive script which will take the input from the user and display the output, this version of the script will not be helpful.

Gomilitary.in

The command **read** is used to read a value to a variable at runtime.

$ read <var> <Enter>

This will read a value to the variable from the standard input device (key board).

Example: Write a shell script which will accept the name and age from the user and display the same on the terminal screen.

$ vi script_2 <Enter> will open a file scripC2 in vi.

Enter the following lines and save it.

```
echo "Enter your name: \c"
read name
echo "Enter your age:\c"
read age
echo "hello $name, nice to meet you. You are $age years old"
```

```
$ sh script_2          <Enter>          Execute this script.
```

The '\c' escape sequence will place the cursor at the end of the output of echo, so that the read command will wait for its input at the same line.

Example: Accept two filenames from the user and copy the first file onto the second. Write a shell script kopy in vi and execute it with the sh kopy command.

```
# This script takes two file names and copies the first-named file into the second one

    echo "Enter source file name:\c"
    read source
    echo "Enter the target file name:\c"
    read target
    cp $source $target
    echo file $source is copied into file $target
```

The hash symbol (#) is used to mark a comment line in the shell script. The statements followed by the '#' sign are ignored.

NOTE:

A shell script can also be executed by assigning the execute permission to it using the chmod command.

Once the execute permission is assigned, the shell script can be executed like any other command in UNIX by giving its name at the prompt.

To convert the scripts described above, we can use:

$ chmod +x kopy <Enter> (and similarly for other scripts)

Classroom Exercise:

Write the other methods to assign execute permission to a script file.

Gomilitary.in