Shell Programing

Shell Programming

- A shell programming is nothing but a series of Unix commands.
- Instead of doing one job at a time, u can give the shell a to-do list that carries out the entire procedure.
- Why learning Shell Programming?
- Shell programs can be used for variety of tasks from customizing your environment to automating your daily tasks.

  Eg your “.profile” file

Different types of Unix Shells

- Unix Shell programs are usually not compatible with each other . This means that a shell script which works with Bourne shell may not work with C shell & vice-versa.
- Bourne shell is a widely used shell & most shell programs are written using this language.
- C shell is also known as the programming shell. This shell allows the aliasing of commands . This proves very useful especially when using lengthy commands.

- **Apart from these two shells , Korn shell is also used . Since it is a superset of the Bourne shell , programs written in Bourne shell will be compatible with the Korn shell also .**

Characters With Special Meaning To The Shell

*?	Metacharacters to specify filenames by pattern matching.
;	Separates multiple commands entered at the command line
\	Hides the meaning of any special character that follows it from interpretation by the Shell.

“ ”	**Hides a space or any special characters that appear within, from interpretation by the Shell.**
‘’	**Single quotes also serve the same purpose as double quotes, except that they hide all special characters that appear within them.**

Characters With Special Meaning To The Shell

``	They are called “grave accents” and must not be confused with single quotes. Grave accents allow the output of any command to be used directly as arguments on the command line.
$	When prefixed to any variable, it instructs the Shell to substitute the variable by its value.

&	Places commands in the background for execution, leaving the terminal free for the other tasks.
|	Creates a pipe, that takes the output of one command & feeds it as input to another command.
<,>,>>	Input output redirection symbols.

Environment Variables

CDPATH	defines the path for a “cd” command
HOME	is the default argument for the “cd” command when we use the “cd” command all by itself, we are taken back to out HOME directory.

MAIL	**This variable defines the path & the name of the filein which the user' mail should be stored.**
MAILCHECK	**it specifies how often the Shell should check for arrival of mail in the file specified by the MAIL variable. (default 600 seconds)**

PS1	**Defines the primary prompt (default "$")**
PS2	**Defines the secondary prompt (">").**
TERM	**Identifies your terminal type. Several programs like vi cannot function normally without a clear understanding of your terminal type.**
LONGNAME	**Identifies your login name.**
IFS	**It defines the internal field separator (normally, the space, tab)**

Defining variables $ displaying on the Screen

- wish=“Hello Everybody”
- echo $wish
- Hello Everybody
- so to print the value of the variable use ‘$’
- echo `who` will print output of the ‘who’ comand.
- Here ` (back quote) is used to print output of the command

Local & Global variables : (use export)

- Ordinarily shell variables is a local variable, means known only to the shell that created it. In order to make the old shell's variable available we use the "export" command.

Eg

export ME

ME="cms"

echo $ME (prints cms)

sh (defines new child shell)

echo $ME (prints cms)

ME="cms institute"

echo $ME (prints cms institue)

press control D (to return to parent shell)

echo $ME (prints cms)

Understanding the .profile file

- When u log in to a ‘UNIX’ sytem, the system looks into ur directory for a file called “.profile”.
- This file contains startup instructions for ur account.
- This file contains definitions for several shell variables, and it exports them to make global. Since ur .profile file belongs to you, you modify it. }

example

cat .profile

HOME=/usr/cms

MAIL=/usr/mail/cms

PATH=.:/bin:/bin:/usr/bin

TERM=adm5

PS1=“C:”

EXPORT HOME MAIL TERM PATH PS1

Interactive shell scripts

- Interactive shell scripts are based on using Unix shell commands read & echo , & on using shell variables . For example
- echo Dear user,'what is your name\?
- read name
- echo Glad to meet you, $name
- In the above example we use the backslash to remove the special pattern-matching property of the question mark character .

Command Substitution

- What makes the set command useful in assigning positional parameter values is that its arguments need not be fixed in advance.when a command is enclosed in back-quotes , the command is replaced by the output it produces. This process is called command substitution.

Script,Pipes& Redirection

- A script can use pipes & redirection both internally & externally. That is pipes & redirection can be part of the commands within a script & the script as a whole can use pipes & redirection to control its input & output .
- For eg:- suppose we want a command that joins a list of files,passes the resulting file thro' pr to format the file, & sends that output to lpr .
- This can be done by – cat $* | pr | lpr

- **Here the $* gathers all the command-line arguments & passes them to cat. The cat command then concatenates the files & sends the combined output to the pr formatter. It in turn sends its output to the lpr line-printer command.**

Word separators

- One set of special characters consists of those characters used to separate one word from the next. These are termed internal field separators in Unix & they are defined by the shell variable IFS.
- Normally these characters are the space character, the tab character & the newline character

Background Process

- Terminating a command with an ampersand (&) sign causes it to run in background mode. The shell waits until the child process produced by a command dies. Thus you can't issue a new command until the preceding one finishes. When a job is run in background, however, the process is launched, & control returns to your shell immediately .

Command Terminators

- Newline,;, and &
- How does the computer know where one command ends & another begins? Normally the newline character, generated by the Return key acts as a signpost. But the semicolon & the ampersand also serve as command terminators.

Inserting Comments

- There is another metacharacter useful in shell scripts. It is the # symbol & it tells the shell to ignore what follows. For instance, $ # this computer is not playing with a full stack.
- The value of this metacharacter is that it lets you place explanatory comments in a shell script.

The for Loop

- A loop is a programming device that lets you cycle thro' the same steps several times. Of the three Unix shell loop forms, the for loop is the most used. It lets you apply the same sequence of operations to a series of subjects. The for loop is especially useful when you want to process several files in same manner.
- For eg:- to do a word count on several files, you can just type wc phys chem biol astr and the word count for each file are printed out.

A word-seeking program

- The grep command is designed to search thro' a file for a word. It even accepts multiple files as arguments. But it only accepts one word (or more generally, one search pattern) at a time. Hence to use several words with grep we need to use a loop.
- However the fgrep or egrep commands do accept a list of search words.

Choice making:The case statement

- The case statement lets a shell script choose from a list of alternatives . The general form of the case statement is case value in

  choice1) commands ; ;

  choice2) commands ; ; …

  esac

Conditional Looping:while AND until

- UNIX offers three loop structures: for, while, & until. The for structure takes a list of things & runs thro' a set of commands on the list members until the list is exhausted. The while & until structures, on the other hand, do not use a list.
- The general form of the while loop is while control command

  do

  commands

  done

The until loop

- The until loop is similar to a while loop, except it is executed until the test condition succeeds. In other words, a while loop runs until a control command fails & the until loop runs until a control command succeeds. The until structure looks like this : until control command

  do

  commands

  done

The if statement

- The syntax of the if command is

  if control command

  then

  commands

  fi

The test Command

- The working of a while, until or if construct depends upon whether or not a command has a successful status. The test command investigates the syntax, the arguments & translates the result into language the control statement can understand, that is, success or failure.

  Eg:- test string1 = string2

- **The test command has several options when checking the status of**
- **Files**
- **Strings**
- **Numerical comparisons**

The exit command

- The exit command terminates the script & it gives the script an exit status. The assigned status is simply the integer argument to the command.