UNIT – III
SHELL PROGRAMMING

Dr.T.Logeswari

Shell

- Interface between user and the kernel

- Prompt for executing Unix program is called ***shell prompt (*** $, %, #)

Types of Shells

1. Bourne shell (sh) : $
2. C shell (csh) : %
3. Korn shell (ksh) : $
4. Bourne-Again shell (bash) : $

- # indicates any shell

Bourne shell

- Symbol : **$**
- Executable filename is **sh**
- Default shell for Unix
- Developed by Stephen Bourne

C shell

- Symbol : **%**
- Executable filename is **csh**
- Similar to C program
- Developed by Bill Joy

- **Advantage** over Bourne Shell
  - C shell can execute processes in background

- **TC shell (tcsh) :** Free version of C shell under Linux

Korn shell

- Symbol : **$**
- Executable filename is **ksh**
- Developed by David Korn

- Korn shell = Bourne shell + C shell

Bourne-Again shell

- Symbol : **$**
- Executable filename is **bash**
- Developed by Fox and C Ramey

- Bash is a freeware shell
- Default for Linux

Shell as command line processor : Steps

1. First it **parses** the command line, identifies and removes extra spaces, tabs etc
2. Evaluates variables prefixed with $ sign
3. Executes commands with back quotes
4. Redirection is checked if any and connects the concerned files
5. Substitutes meta-characters like *
6. Looks for required files and commands, retrieves them and transfers them to the kernel for execution
7. PATH variable helps to find the path for the commands
8. Semicolon involves multiple commands
9. Logical operators are evaluated

Editors

- Software package used to enter and execute any program

- Unix editors : Vi, Vim, emacs, pico

Editor

1. ed:
   - First editor on Unix
   - Developed by Ken Thompson
2. ex:
   - Extended editor
   - Developed by Bill Joy
3. vi:
   - **Vi**sual editor
   - Screen-oriented version of ex
4. vim:
   - **V**i **Im**proved
   - Improved version of vi

Modes of vi editor

1. Command mode
   - Here every character typed is a command
   - <Esc> : used to return to command mode
2. Insert mode
   - Every character typed is added to the text in the file
   - Insert or i
   - <Esc> : used to exit insert mode
3. Ex mode
   - Last-line command
   - Makes user to enter commands at the bottom of vi screen
   - Colon (:) is used to enter ex mode

Invoking vi

- vi filename. extension
- Filename occurs at the bottom of screen

- Options:
  - vi filename : edits filename starting at line 1
  - vi –r filename : recovers filename that was being edited when system crashed
  - vi +n filename : edits filename and places cursor at line n

Quiting vi

- While quiting vi, new or modified file is automatically saved

- Options:
  - :x / :wq / :q - quits vi ,writing modified file
  - :w – writes modified file and remains in command mode
  - :q! – quits vi, without saving the latest changes
  - ZZ – saves and exits; this is known as last command

Moving the cursor

- Mouse does not work in Unix
- Arrows and keys has to be used

- Options:
  - h : moves cursor left one character
  - l : moves cursor right one character
  - j : moves cursor down one line
  - k : moves cursor up one line

Arrow keys

- Slow on lengthy files
- But sometimes they produce strange effects
- Options:
  - 0(zero) – moves cursor to start of current line
  - $ - moves cursor to end of current line
  - W - moves cursor to beginning of next word
  - b - moves cursor to beginning of preceding word
  - :0 or 1G - moves cursor to first line in file
  - :n or nG - moves cursor to line n
  - :$ or G - moves cursor to last line in file

Screen manipulation

- Caps lock (^) before a letter indicates ctrl key

- Options:
  - ^f : moves forward one full screen
  - ^b : moves backward one full screen
  - ^d : moves forward one half screen
  - ^u : moves backward one half screen

Undo and Redo

- u : undo , single toggle
- Ctrl r : redo

Inserting text

- Options:
  - i : inserts text before cursor until you press <Esc>
  - a : inserts text after cursor until you press <Esc>
  - o : opens and puts text in a new line below current line, until you press <Esc>
  - O : opens and puts text in a new line above current line, until you press <Esc>

Changing text

- Options:
  - r : replaces single character under cursor
  - R : replaces characters, starting with current position, until you press <Esc>

Deleting text

- Options:
  - x : deletes single character under cursor
  - dd : deletes entire current line

Cutting and pasting text

- Options:
  - yy : copies current line into buffer
  - p : puts or pastes the line in the buffer into the text after current line

Saving and reading files

- Options:

  - : r filename – reads file named ‘filename’ and inserts after current line
  - :w – writes current contents to a file
  - :w newfile - writes current contents to a new file named ‘newname’
  - :w! prevfile - writes current contents over a pre-existing file named ‘prevfile’

Searching text

- Options:

  - /string : searches forward for occurrence of string in text
  - ?string : searches backward for occurrence of string in text
  - n : moves to next occurrence of searched string
  - N : moves to next occurrence of searched string in opposite direction

Regular expressions

- . (dot) – any single character except newline
- * - zero or more occurrence of any character

- [….] – any single character specified in the set
- [!.] - any single character not specified in the set

- ^ - beginning of line
- $ - end of line

[….] – Set examples

- [A_Z]
- [a-z]
- [0-9]
- [0-9 A-Z]
- [A-Z][a-zA-Z][0-9]

Regular expression examples

- /Hello/
- /^[a-zA-Z]/
- /^[a-z]*/
- /2134$/
- /[0-9]*/
- /^[!#]/
- /^TEST$/

Determining line numbers

- Options:
  - **:.=** - returns line number of current line at the bottom of screen
  - **:=** - returns total number of lines at the bottom of screen
  - **^g** - provides current line number with the total number of lines, in the file at the bottom of screen

    Ex: “hello” Line 3 of 6 – 50% -- Col 1

Vi settings

- Options here are default
- :set option – turn on
- :set nooption – turn off

- Options:
  - :set ai – turns on auto indentation
  - :set all – prints all options to the screen
  - :set eb – precedes error messages with bell
  - :set nu – shows line numbers
  - :set prompt – prompts for command input with :
  - :set showmode – indicates input or replace mode at bottom

Shell variables

- They have the ability to store and manipulate information within a shell program
- Rules for naming variables:
  - Can contain alphanumeric character and underscore(_)
  - Must start with alphabet or underscore
  - Case sensitive
  - No limit on length of variable name
  - Ex: No_of_std, NAME

Types of shell variables

- System variables
- Local or user defined variables
- Read-only variables

System variables

- Also known as environment variables
- Set either during booting or after logging in
- Written in uppercase only

- Ex: PATH, HOME, IFS, SHELL, LOGNAME, OSTYPE, PS1,PS2

System variables

- PATH : list of directories separated by colon(:)
- HOME : path of home directory
- SHELL : absolute pathname of user's shell program
- LOGNAME : stores user name
- OSTYPE : type of OS
- PS1 : holds primary prompt value ($)
- PS2 :holds secondary prompt value (>)

- IFS :
  - Internal Field Separator
  - Holds token used to separate a string into sub-strings
  - 3 default tokens : space, tab, newline
  - **od** : used to display non-printable characters
  - -b : displays octal value
  - -c : displays character itself

  - Ex: $echo $IFS | od –bc

    011 012 012

    \t \n \n

User-defined variables

- Variables are defined and used by users
- Exist only during execution of shell program
- Local to the user's shell
- Not accessible to other users
- Ex: read x, y, z

  x=10

  y=20

  z=30

Read-only variables

- Values are fixed
- **'readonly'** function is used to convert any variable to read-only variable

- Ex: echo Enter a number :

```
read n                  # Enter a number : 5
echo N=$n               # N=5
readonly n
n=n+10
echo N=$n               #N=5
```

Defining and evaluating shell variable(Storing)

- Assignment statement i.e., equal to (=) operator without space on either side of it
- **Syntax** : variable=value
- Ex: no=10

- ‘$’ is used to display variable values
- Ex: echo $no # display value 10

- To define NULL values
- Ex: num=“ ” or num=

Data type of shell variables

- All shell variables are **string** type
- Content are stored in ASCII format
- By default, shell variables are initialized to null string; hence it is not required to declare or initialize them
- Ex:

```
Num=10
echo $Num
```

Script

- A script is a file that contains set of shell commands

FEATURES OF SHELL SCRIPT

- It is a complete programming language
- It consists of sequence of commands for selective execution, I/O operations and looping
- It runs in an interpretive mode, executing one statement at a time
- They are named just like any other files
- When a shell script is created for the first time, it will have only read and write permissions. Execution permission must be granted

EXECUTING SHELL SCRIPT

- 2 ways:
  1. sh filename.sh
  2. chmod +x filename.sh

     ./ filename.sh

Eg $ chmod u +x prog1.sh

$./prog1/.sh

welcome

COMMENTS

- Comments are non-executable statements
- They explain the purpose of the program, login and complex commands used
- Comments are written by using ‘ # ‘ character as the first character

- Ex: #program to display date

```
date            # shows current date
```

KEYWORDS

- Words with predefined meaning
- They cannot be used as ordinary shell variables

- Ex: echo, read, while, if, for, case, break

USER INPUT STATEMENT (Read Command)

- Shell allows to prompt for user input
- read statement is used to get input from user and store data into a variable
- Syntax:

```
read varname1, varname2…..
```

USER INPUT EXAMPLE

```
echo “Enter 3 numbers”
read a b c
echo “Entered numbers are $a $b $c”

read -p "enter your name: " first last
echo "First name: $first"
echo "Last name: $last"
```

SHELL ARITHMETIC USING expr

- All variables in Unix are string variables
- **expr** command is used to convert string variables to numeric format

- Arithmetic operators:
  - +, -, *, /, %

- **bc** is used for floating point calculations

EXAMPLES

- a=5 ; b=10
- expr $a + $b
- expr $a - $b
- expr $a * $b
- expr $a / $b
- expr $a % $b

EXAMPLES

- expr is used with command substitution using back quotes (` `) to assign values to variables

- a=5; b=10
- a=`expr $a + 1` or a=$(expr $a + 1)
- echo $a

RULES FOR ARITHMETIC OPERATIONS

- Multiple assignments can be done in a single line

  Ex: a=5 ; b=10

- Operators in expr command must be followed and preceded by at least one blank space

$ expr $a + $b + $c

- Hierarchy of operators:
  - 1: / * %
  - 2 : + -

BASIC CALCULATOR (bc)

- For real arithmetic's, basic calculator (bc) is used
- Output of arithmetic calculation are piped to bc
- Scale function is used to set number of decimal places after decimal point

- Ex: a=5.5; b=10.5

```
scale=2
c=`echo $a + $b | bc`
d=`echo $a \* $b | bc`
echo $c $d
```

COMMAND SUBSTITUTION

- Two commands can be connected either by using pipeline or by command substitution

- Back quote (` `) works only within double quotes and doesn’t work within single quotes

- echo “ Current date is `date` ”
  - Current date is Tuesday Feb 18 11:55:59 IST 2014
- echo ‘ Current date is `date` ’
  - Current date is `date`

ESCAPE SEQUENCE

- Two character String beginning with \ (backslash)
- \c : places cursor in the same line that displays the output

- Ex: echo “Enter name: \c “
  - Enter name: $_

- \t : tab of 8 character position
- \n : newline character equivalent to pressing ENTER

Positional Parameter

- Information can be conveyed to a shell script through command line argument or shell script arguments
- These argument submitted with a shell script are known as positional parameters
- Bourne shell stores the first nine command line arguments in the variable $1, $2….$9

POSITIONAL PARAMETERS

- Command line arguments submitted with a shell script
- Positional parameters automatically store values of command line arguments
- **set** command is used to assign values

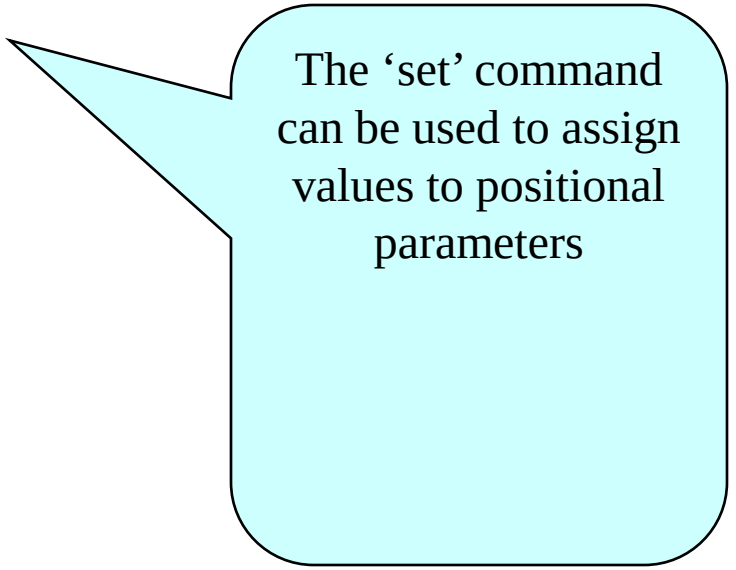
POSITIONAL PARAMETERS

Parameter	Meaning
$0	Name of the current shell script
$1-$9	Positional parameters 1 through 9
$#	The number of positional parameters
$*	All positional parameters, “$*” is one string
$@	All positional parameters, “$@” is a set of strings
$?	Return status of most recently executed command
$$	Process id of current process

EXAMPLES:

```
$ set tim bill ann fred
      $1  $2   $3  $4

$ echo $*
tim bill ann fred
$ echo $#
4
$ echo $1
tim
$ echo $3 $4
ann fred
```



EXAMPLES:

$ set `date`

Tuesday Feb 18 11:55:59 IST 2014

$1 $2 $3 $4 $5 $6

$ echo $*

Tuesday Feb 18 11:55:59 IST 2014

$ echo $# : 6

$ echo $1 : Tuesday

$ echo $4 : 11:55:59

Exit command

- Terminates the execution of shell scripts
- If program is executed successfully, it returns non-zero; otherwise zero value is returned
- $? : variable that stores the status of exited command

TEST COMMAND

- test expression

Or

- [expression]

- Ex: a=5; b=10

```
test $a –eq $b ; echo $?

[ $a –eq $b] ; echo $?
```

TEST COMMAND

- 3 types:
  - Numeric test
  - String test
  - File test

NUMERIC COMPARISON

OPERATORS	MEANING	USAGE
-eq	Equal to	if [5 –eq 6]
-ne	Not equal to	if [5 –ne 6]
-lt	Less than	if [5 –lt 6]
-le	Less than or equal to	if [5 –le 6]
-gt	Greater than	if [5 –gt 6]
-ge	Greater than or equal to	if [5 –ge 6]

Output: True : $?=0; False : $?=1

STRING COMPARISON

OPERATORS	MEANING
str1 = str2	str1 is equal to str2
str1 != str2	str1 is not equal to str2
str1	str1 is not null or not defined
-n str1	str1 is not null and exists
-z str1	str1 is null and exists

STRING COMPARISON

- = sign must be preceded and followed by at least one blank space
- If string contains more than one word separated by white space, then they must be enclosed in double quotes
- Ex: str1="New Horizon College"

- While comparing such strings they must be enclosed in quotes
- Ex: ["str1" = "str2"]

FILE COMPARISON

TEST	MEANING
-e file	True if file exists
-f file	True if file exists and a regular file
-r file	True if file exists and is read-only
-w file	True if file exists and is writable
-x file	True if file exists and is executable
-s file	True if file exists and is non-empty
-d file	True if file exists and is a directory

LOGICAL OPERATORS

OPERATORS	TYPE
! expression	Logical NOT
expression1 –a expression2	Logical AND
expression1 –o expression2	Logical OR

HIERARCHY OF OPERATORS

OPERATORS	TYPE
!	Logical NOT
-lt, -gt, -le, -ge, -eq, -ne	Relational
-a	Logical AND
-o	Logical OR