

CS-344 - Unix Operating System Fundamentals

Lecture 10 Shell Programming

Based on slides created by
Dr. Bangalore for the
Spring 2005 offering of
the course

Shell: Turn Interpretation Off

- Shell interprets characters entered at the command prompt
- To turn off this interpretation we must use
 - Backslash - \
 - Double quotes - ""
 - Single quotes - ''
- What happens when the following commands are used
 - echo '\$HOME' '\$HOME' \ \$HOME
 - echo My files are: "*"
 - echo "My files are: *"
 - echo 'My files are: *'

“echo” command

- Using " " and ' ' with *echo*

```
$ echo 'Hello $USER'
```

```
Hello $USER
```

```
$ echo "Hello $USER"
```

```
Hello puri
```
- When the shell interprets ``` it stops interpretation until a matching ``` is found
- To eliminate new line with *echo* use

```
echo -n <text>
```

```
echo -n "Enter your choice: "
```

11/4/2005

7

Shell Programming



Passing Arguments to a Script

- Similar to passing command-line arguments in other programming languages, we can pass arguments to a shell script
Example: `./myscript arg1 arg2`
- Shell interprets the arguments are follows:
 - \$1 – argument 1
 - \$2 – argument 2 and so on
 - \$0 – the name of the current script
 - \$* – all arguments
 - \$# – no. of arguments
 - \$\$ – process id (PID) of process running the script

11/4/2005

9

Example: script with complex arguments

```
$ cat > cmdscript.sh
echo "The name of the script is: '$0'"
echo "No. of command-line arguments = '$#' "
echo "First three arguments are: '$1 $2 $3'"
echo "Complete argument list is: '$*' "
echo "PID of this process = '$$'"

$ chmod +x cmdscript.sh

$ ./cmdscript.sh "Hello World" "$USER" "$USER"
The name of the script is: ./cmdscript.sh
No. of command-line arguments = 3
First three arguments are: Hello World puri $USER*
Complete argument list is: Hello World puri $USER*
PID of this process = 5828

$ ./cmdscript.sh a b c
The name of the script is: ./cmdscript.sh
No. of command-line arguments = 3
First three arguments are: a b c
Complete argument list is: a b c
PID of this process = 5831
```

11/4/2005

10

Simple Arithmetic

- Using "bc"
 - `a=8; echo "$a * 9" | bc` (result is 8*9 = 72)
- Using "let" (assign values to variables and to perform arithmetic calculations)
 - `a=8`
 - `echo $a`
 - `let a=$a*9` [could also use `((a=$a*9))`]
 - `echo $a` [could use `echo $((a=$a*9))` instead]

11/4/2005

11

Obtaining Exit Status of Processes

- When we type a command and press ENTER
 - the shell interprets the command
 - creates child process (or processes)
 - completes input/output redirection and passes arguments
 - instructs child process to execute appropriate program
 - when program completes, shell receives the exit status code from the child process (every program returns an exit code on completion)
- To obtain this exit status, type "echo \$?" after the command is entered
- An exist code "0" indicates program completed successfully, otherwise something went wrong

11/4/2005

12

“test” or “[” command

- “test” or “[” command used for making decision
 - [*statement*] - returns 0 to the parent process if *statement* is true, otherwise 1
 - [-f *xyz*] - tests if *xyz* is a file, returns 0 if *xyz* is a file
 - [*string*] - true if a single string is present
 - [*x -eq y*] - true if *x* is equal to *y*
 - [*x -ne y*] - true if *x* is not equal to *y*
 - [*x -gt y*] - true if *x* is greater than *y*
 - [*x -lt y*] - true if *x* is less than *y*
 - [*x -ge y*] - true if *x* is greater than or equal to *y*
 - [*x -le y*] - true if *x* is less than or equal to *y*

NOTE: space after [and before] required

```
[$ [ -f /etc/passwd ]  
$ echo $?  
0
```

```
[$ [ 6 -eq 7 ]  
$ echo $?  
1
```

```
[$ [ 6 -lt 7 ]  
$ echo $?  
0
```

```
[$ [ $USER ]  
$ echo $?  
0
```

11/4/2005

13
