

# CS-344 - Unix Operating System Fundamentals

## Lecture 10 Introduction to 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
  - Backslash - \
  - Double quotes - ""
  - Single quotes - ''
- 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: \*'

11/3/2006

3

### Effect of Quoting on Special Characters

Character	Inside ""	Inside ''	After \
*	NO	NO	NO
[ ]	NO	NO	NO
{ }	NO	NO	NO
<	NO	NO	NO
	NO	NO	NO
>	NO	NO	NO
;	NO	NO	NO
\$	YES	NO	NO
' (single quote)	NO	-	NO
" (double quote)	-	NO	NO
` (back quote)	YES	NO	NO
spaces	NO	NO	NO
newline	NO	NO	NO
!	YES	NO	NO

11/3/2006

4

### Summary

- For the bash shell
  - The backslash character and single quotes turn off interpretation for all special characters
  - The double quotes turn off interpretation of all special characters except \$, `, and \ (\ - only if the next character is interpreted)
- For the C shell
  - The backslash character turns off interpretation for all special characters
  - The single quotes turn off interpretation of all special characters except the !
  - The double quotes turn off interpretation of all special characters except \$, `, and !

11/3/2006

5

### Miscellaneous

- Passing Special Characters to Utilities
  - grep treats \$ character as end of line (e.g.,  
`grep '$' MyClass.java` - look for lines ending with ;)
- Single and double quotes can be mixed in commands (only outer quotes have any effect)
  - echo '\$USER'
  - echo "\$USER"
  - echo '\$USER' "\$USER"
  - echo '\$USER' "\$USER"
- Interpreting special characters in variable names

```
$ aa=t
$ echo $aa $aa "$aa"
t.c t1.c temp.java tmp typescript $aa t
```

```
$ aa=t
$ echo $aa $aa "$aa"
t.c t1.c temp.java tmp typescript $aa t
```

11/3/2006

6

## “echo” command

- Using “ ” and ‘ ’ with `echo`

```
$ echo 'Hello $USER'
Hello $USER
$ echo "Hello $USER"
Hello afgane
```
- 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: "; echo "1, 2, 3"
```

11/3/2006

7

# Shell Programming

## Introduction

- Used for storing commonly used sequences of commands
- So, a series of shell commands can be stored in a regular text file for later execution
- Before running a script, need to give it execute permissions with `chmod` utility

11/3/2006

9

## Determining shell for a Script

- When script is run, the system determines which shell the script was written for and then executes the shell using the shell as stdin
- The system knows which shell to use to execute the script by examining the first line of the script:
  - If the first line is just a #, the script is interpreted by the shell from which it is executed
  - If the first line is of the form `#! pathName`, then the executable program `pathName` is used to interpret the script
  - If neither of the two rules apply, then the script is interpreted by a Bourne shell.

```
$ cat myDefaultScript.sh
#
# Print today's date
echo -n The date today is
date
```

```
$ cat myBashScript.sh
#!/bin/bash
# Print today's date
echo -n The date today is
date
```

```
$ cat myScript.sh
# Print today's date
echo -n The date today is
date
```

11/3/2006

10

## 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 as 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

11/3/2006

11

## 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 afgane $USER
Complete argument list is: Hello World afgane $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
```

11/3/2006

12

## Simple Arithmetic (I)

- 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
echo $a
```

11/3/2006

13

## Simple Arithmetic (II)

- Using "expr"
 

```
x=1; x=`expr $x+1`; echo $x ... result will be 2
x=`expr length "mouse"`` ... find length of word mouse
```
- In bash, double set of parentheses will perform arithmetic operations: ((operation))
 

```
a=8
((a=$a*9))
echo $a ...result will be 72
```

Operation	Description
+, -	Addition, subtraction
++, --	Increment, decrement
*, /, %	Multiplication, division, remainder
**	Exponentiation

11/3/2006

14

## Arithmetic sense

- Integer arithmetic is faster than floating point arithmetic!
- If you know your variable will always be integer, use *declare* to declare it to be integer
 

```
declare -i name -> defines variable name as integer
declare -i x=2
x=3.2
((x=$x*5.1))
```

... syntax error in expression

11/3/2006

15

## 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

11/3/2006

16

## "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 ] is required**

<pre>[\$ [ -f /etc/passwd ] \$ echo \$? 0</pre>	<pre>[\$ [ 6 -eq 7 ] \$ echo \$? 1</pre>	<pre>[\$ [ 6 -lt 7 ] \$ echo \$? 0</pre>	<pre>[\$ [ \$USER ] \$ echo \$? 0</pre>	<pre>[\$ test -f foofoo \$ echo \$? 0</pre>
---	--	--	---	---

11/3/2006

17