

CS 344 – UNIX OS

Fundamentals – Lecture #8

Purushotham Bangalore
Department of Computer and Information
Sciences
University of Alabama at Birmingham (UAB)

Instruct Shell to 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: *'

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

3/21/2005

3

Summary

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

3/21/2005

4

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
 - echo '\$USER'
 - echo "\$USER"
 - echo '\$USER' "\$USER"
 - echo "\$USER" `date` "\$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*
```

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

3/21/2005

7

Example 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
```

3/21/2005

8

Simple Arithmetic

- Using bc
a=8; echo "\$a * 9" | bc (result is 8*9 = 72)
- Using "let"
a=8
echo \$a
let a=\$a*9 [could also use ((a=\$a*9))]
echo \$a [could use echo \$((a=\$a*9)) instead]

3/21/2005

9

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

3/21/2005

10

“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
```

3/21/2005

11

if...then...else

- Syntax:
 - if *command* (run command and obtain exit status)
 - then *command* (if exit status is 0, execute this)
 - else *command* (if exit status is not 0, execute this)
 - fi
 - OR
 - if condition ; then action ; else action2 ; fi
- Example:

```
if [ $1 ]
then
  if [ -f $1 ]
  then echo File Exists
  else echo File does not exist
  fi
else
  echo "Usage: $0 <filename>"
fi
```

3/21/2005

12

for

- **Syntax:**
for *word* in *wordlist*
do
command
done
OR
for word [in wordlist ...] ; do actions ; done
- **Example:**
for file in *.java
do
echo \$file
wc -l \$file
done
OR
for file in *.java ; do echo \$file; wc -l \$file; done

3/21/2005

13

while

- **Syntax:**
while *command* (run command and obtain exit status)
do
command (if exit status is 0, execute command)
done
OR
while [conditions] ; do actions ; done
- **Example:**
a=0
while [\$a -le 10]
do
echo \$a
((a=\$a+1))
done
OR
a=0; while [\$a -le 10]; do echo \$a; ((a=\$a+1)); done

3/21/2005

14

case

- Syntax:
case word
a) *command* ;; (if word matches a, execute *command*)
b) *command* ;; (if word matches b, execute *command*)
esac
OR
case word in [pattern [| pattern]) actions ;; ...] esac
- Example:
a=\$1
case \$a in
A) date ;;
B) cal ;;
C) ls ;;
D|E) who | wc -l ;;
esac
OR
a=\$1; case \$a in A) date ;; B) cal ;; C) ls ;; D|E) who | wc -l ;; esac

3/21/2005

15

Reading User Input

- To read input from users in a shell use the “read” command
- The variable name is passed as an argument to the read command
- If a variable exists its value is modified, otherwise a new variable is created and the value entered is assigned
- Syntax: read *variable_name*

```
$ read a; echo $a  
1234  
1234
```

```
a=0  
while [ $a -le 5 ]  
do  
read b; echo $a $b; ((a=$a+1));  
done
```

3/21/2005

16

Complete Shell Script

```
# An example script with various shell constructs
clear
echo "1) grep $USER /etc/passwd"
echo "2) ypcat passwd | grep $USER"
echo "3) who | sort | awk '{print $1}' | uniq -c"
echo "What command do you want to run?"
echo Select 1, 2, or 3
read choice
echo You have selected option $choice
echo
echo The output for the selected command is:
case $choice in
1) grep $USER /etc/passwd ;;
2) ypcat passwd | grep $USER ;;
3) who | sort | awk '{print $1}' | uniq -c ;;
esac
a=$?
if [ $a -eq 0 ]
then
echo Command executed successfully
else
echo Command returned with the exit code $a
fi
```