Lecture 5

Introduction to Shell Programming

COP 3344 Introduction to UNIX

Fall 2007

Acknowledgment: These slides are modified versions of Prof. Sudhir Aggarwal's slides

What Is a Shell Script?

- An executable file containing
  - Unix shell commands
  - Programming control constructs (`if`, `then`, `while`, …)
  - Basic programming capabilities (assignments, variables, arguments, expressions, …)
- The file entries are the *script*
- The file is interpreted rather than compiled and executed
  - The first line of the script indicates which shell is used to interpret the script

A Simple Script

```
#!/bin/sh
#this is the script in file my700
chmod 700 .
chmod go-rwx *
ls -l
ls -ld
```

- The `#!` is used to indicate that what follows is the shell used to interpret the script

Executing Shell Scripts

- Make the file executable and then run it

```
$ chmod 700 my700
$ ./my700
total 24
-rw-------   ... 14:31 file1
-rw-------   ... 14:32 file2
-rwx------   ... 14:35 my700
drwx------   ... 14:42 .
```


Another Shell Script

- List only a few fields in ls -l

myls

```
#!/bin/sh
ls -l | cut -c 1-10,38-
```

```
$ ./myls
total 32
-rw------- Sep 28 14:31 file1
-rw------- Sep 28 14:32 file2
-rwx------ Sep 28 14:42 my700
-rwx------ Sep 28 15:01 myls
```


Command Line Arguments

- Arguments on the command line can be passed to a shell script
  - $1, $2, …, $9 refer to up to the command line arguments
    - Note that $0 contains the name of the script
    - $* contains all the arguments
    - $# contains the number of arguments

echoargs

```
#!/bin/sh
echo $0: You entered the following $# arguments -- $*
The first argument was: $1
```

```
$ ./echoargs hello world!
./echoargs: You entered the following 2 arguments --
hello world!
The first argument was: hello
```

Another Example

mydiff
```
#!/bin/sh
diff -w $1 $1~
```

```
$ ./mydiff my700
5d4
< ls -ld
```

my700
```
#!/bin/sh
chmod 700 .
chmod go-rwx *
ls -l
ls -ld
```

my700~
```
#!/bin/sh
chmod 700 .
chmod go-rwx *
ls -l
```


User Defined Variables

- You can define variables
  - Set with the `set` command in *csh*
    ```
    set mypath=/home/special/public_html
    ```
  - Use = in *sh*
    ```
    beta = 3
    ```
  - Its value can be dereferenced with $
    ```
    echo $mypath
    ```

```
$ MyName="Ashok Srinivasan"
$ echo $MyName
Ashok Srinivasan
```

myadd
```
#!/bin/sh
sum=`expr $1 + $2`
echo $sum
echo $1 + $2
```

```
$ ./myadd 45 32
77
45 + 32
```

- Note
  - The command between ` and ` is run, and its output assigned to the variable `sum`
  - The command `expr` can evaluate an expression

Using *set* in Bourne Shell

- You can reset the command line arguments using `set`
  - Note that the third line of *mywc* uses \` to remove the special meaning of `

```
$ wc file1
   3  5  21  file1
```

mywc
```
#!/bin/sh
set `wc $1`
echo File \`$4\`:
echo Lines = $1
echo Words = $2
```

```
$ ./mywc file1
File `file1`:
Lines = 3
Words = 5
```


Quoting Rules

- Using single quotes
  - 'xyz' disables all special characters in xyz
- Using double quotes
  - "xyz" disables all special characters in xyz except $, `, and \
- Using the backslash
  - \x disables the special meaning of character x

Quoting Examples

`var1=alpha`	sets the variable
`echo $var1`	prints: `alpha`
`echo "$var1"`	prints: `alpha`
`echo '$var1'`	prints: `$var1`
`cost=2000`	sets the variable
`echo 'cost:$cost'`	prints: `cost:$cost`
`echo "cost:$cost"`	prints: `cost:2000`
`echo "cost:\$cost"`	prints: `cost:$cost`
`echo "cost:\$$cost"`	prints: `cost:$2000`

Reading Values into Variables

- Use `read` to store input into variables

echoargs2
```
#!/bin/sh
echo Enter your name.
read FirstName Lastname
echo $FirstName, you typed the following
command line arguments: $*
```

```
$ ./echoargs2 Hello World!
Enter your name.
Ashok Srinivasan
Ashok, you typed the following command line
arguments: Hello World!
```