Agenda

- *Shell Programming*
- *Shell scripts-Why?*
- *Invoking a shell script*
- *A Sample Script*
- *The "read "command*
- *Control Flow Constructs-if , case*
- *while loop*
- *for loop*
- *The “expr” command*

Shell Programming

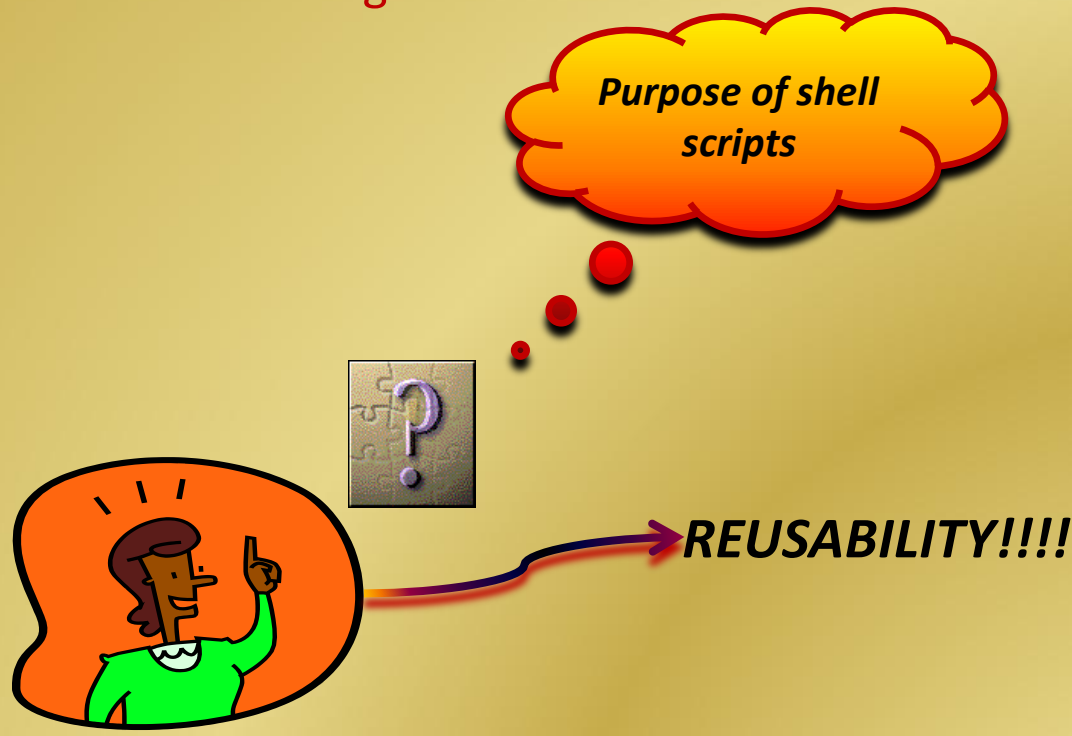
- Shell has a whole set of internal commands that can be put together as a language-i.e. its own variables, conditions and loops.
- Most of the constructs are borrowed from C.
- The external UNIX commands can blend easily with the shell's internal commands/constructs as well to create ***"Shell programs"***

The external UNIX commands blend easily with the shell's internal constructs

Shell scripts

-Why?-

- A shell script is simply a text (ASCII) file containing shell commands
- Created using the *"vi editor"*



Though it's not mandatory, we normally use the ".sh" extension for shell scripts

Execution

- Runs in an interpretive mode-i.e. not compiled to a separate executable file like in case of a C program.
- Here, each statement is loaded into the memory where it is executed.
- Because of this, generally shell programs may run slower than those written in high level languages.

Invoking
a
shell script

- To run the script, make it ***executable first*** and then invoke the ***script name***

```
debalina@bootcamp:~
[debalina@bootcamp ~]$ chmod +x 1.sh
[debalina@bootcamp ~]$
[debalina@bootcamp ~]$ 1.sh
Hello User....File is executable
[debalina@bootcamp ~]$
[debalina@bootcamp ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/debalina/bin:.
[debalina@bootcamp ~]$
```

- Run a ***separate shell*** to execute the script

```
debalina@bootcamp:~
[debalina@bootcamp ~]$ sh 1.sh
Hello User....File is executable
[debalina@bootcamp ~]$
```

Make sure that "." (current directory) is in PATH ; else use ***"./1.sh"*** to execute the script

A Sample Script

```
debalina@bootcamp:~
#!/bin/bash

#Script to display the current date and the shell name

echo -e  "Current date is `date`"

echo -e "\n My shell is $SHELL"

echo -e "\n Script ends"
~
```

Comment character

Shell script statements

Output

```
debalina@bootcamp:~
[debalina@bootcamp ~]$ sh demo.sh
Current date is Mon May 21 18:16:03 IST 2007

 My shell is /bin/bash

 Script ends
[debalina@bootcamp ~]$
```

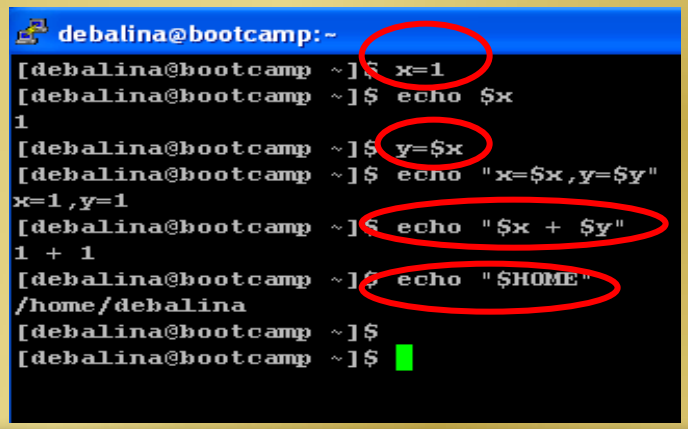
Multi-line commenting

- Wrap the lines within :<<MARKER.. MARKER like shown below
- Instead of MARKER you can use any string which is not used in your script. Preferably a unique string.
- Example Code:

```
:<<COMMENT
a line in the script.
another line in the script
<<yet another line in script>>
COMMENT
```

Shell Variables

- Shell supports variables that are useful to both in the command line and shell scripts
- Variables are assigned by using the "=" symbol
- Values of the variables are evaluated by using a "$" as prefix to the variable name
- Variables names are case senstitive
- All variables are of the string type and are initialized to "null" string by default.
- By convention UNIX system variables are in upper case like-***SHELL,HOME*** etc are all system variables



No whitespace on either side of the "= "symbol when assigning values to variables

The "read" command

- Is the shell's internal tool for taking input from the user.
- Used to make the script interactive
- Can be used with one or more variable like- read v1 v2
- Since this is a form of assignment no $ is used before the name.

Control Flow Constructs

- Every programming language needs repetition and selection constructs
- The shells provide the usual set:
  - if
  - while
  - for
- These work together with an extensive set of test functions

The "*if*" conditional

- The if construct is built in to the shell
- Makes a decision depending on a certain condition
- Usage:

```
if condition1
        then commands1
[elif condition2
        then commands2]

[else commands3]
fi
```

"if " also requires a ***"then"***

The *"test "* command

- The ***"test"*** command is used in conjunction with ***"if "*** to evaluate expressions
- ***"test"*** uses certain operators to evaluate the condition and returns either a true or false exit status
- ***"if"*** uses the returned exit status for decision making
- ***"test"*** can be used for :-
  - Comparing 2 numbers
  - Comparing 2 strings
  - Checking a file's attributes

"test" does not display any output but simply returns a value that sets the parameter " $? "

Numerical Comparison

Operator	Meaning
-eq	Equal to
-ne	Not equal to
-lt	Lesser than
-gt	Greater than
-ge	Greater equal to
-le	Lesser equal to

debalina@bootcamp:~

```
#Script to check whether 2 numbers are equal or not
echo -e "\nEnter the 2 numbers"
read num1 num2
if test $num1 -eq $num2
then
        echo -e "\n EQUAL"
else
        echo -e "\nUNEQUAL"
fi
~
~
~
```

Using the –eq operator

Shorthand for test:- ***if test $num1 –eq $num2*** can be written as ***if [$num1 -eq $num2]***

String Comparison

Test	*True if*
s1 = s2	String s1 = s2
s1 != s2	String s1 not equal to s2
-n s1	String s1 is not a null string
-z s1	String s1 is a null string
s1	String s1 is assigned and not null
s1 == s2	String s1 = s2 (Korn and Bash only)

File Tests

Test	*True if*
-f file	File exists and is a regular file
-r file	File exists and is readable
-w file	File exists and is writeable
-x file	File exists and is executable
-d file	File exists and is directory
-e file	File exists
-s file	File exists and has a size > 0

The **"case"** conditional

- The case statement is an expression with multiple alternatives
- Similar to the switch-case construct of C/C++
- The general form of the case construct is:

```
case "$variable" in
    condition 1)commands;;
    condition 2)commands;;
     condition 3)commands;;
    *)commands
esac
```

'' in **"case"** is equivalent to "**else**" in **"if "**construct*

"case" can be used to match values directly ,but cannot make any numeric checks(of the type $var –lt 2)

Sample code

```
# To extract the day of a week from the current date and check whether it is a weekday or a weekend

day=`date|cut -d " " -f 1`
case "$day" in
Fri|fri)echo "Weekend approaching";;
Sat|sat)echo "Weekend starts";;
Sun|sun)echo “Holiday!!!";;
*)echo "Sad :-( !";;
esac
```

Sample code

#Use wildcards in a case-esac construct

```
echo "enter answer yes or no"
read ans
case "$ans" in
[Yy][Ee]*)echo "Answer is YES";;
[Nn][oO])echo "Answer is NO";;
*)echo "Not a valid choice"
esac
```

while
loop

- For a number of repetitions
- Usage:

```
while condition
do
   commands
done
```

- The commands will be executed as long as the condition is true

Example

-Using ***while***-

```
#Generate numbers from 1-10
  n=1
while test $n -le 10
do
  echo "$n"
  n=`expr $n + 1`
done
```

for

loop

- For iterating over a ***list***
- Usage:

  for item [in list]
  do
  commands
  done

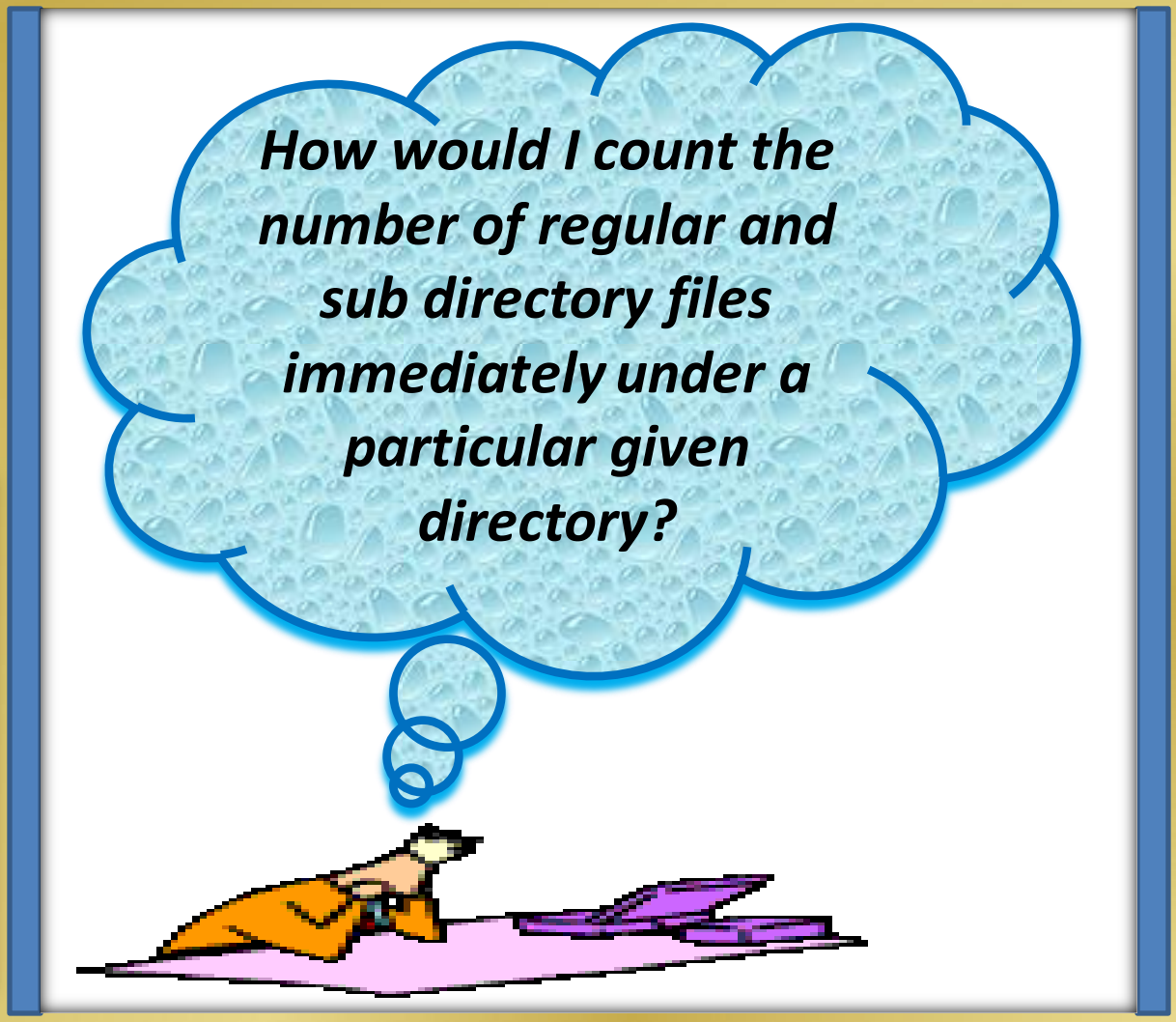
- Each pass through the loop assigns the next item in the list to $item

Example

-Using ***for*** -

```
#Using for loop
for var in $PATH $SHELL $HOME
do
    echo $var
done
```

Scenario


How would I count the number of regular and sub directory files immediately under a particular given directory?

Possible list types in for loop

```
#List types used in for loop
for var in $PATH $SHELL $HOME
do
   echo $var
done

for fname in s*.sh
do
    ls -l $fname
done
```

Possible list types in for loop

```
for fname in `ls c*.sh`
do
   cat $fname
done

for i in "$@“
do
  echo $i
done
```

The *"expr"* command

- The ***expr*** command is used in both arithmetic computation and string handling

```
debalina@bootcamp:~
[debalina@bootcamp ~]$ a=10
[debalina@bootcamp ~]$ b=9
[debalina@bootcamp ~]$ echo $a + $b
10 + 9
[debalina@bootcamp ~]$ echo "`expr $a + $b`"
19
[debalina@bootcamp ~]$
```

Values are strings by default

expr used for computations

- The ***expr*** command computes an expression and returns it's results
- A **space** on either side of the operators is essential in an expr statement.

The * is escaped using a "****" so that the shell does not interpret it as one of it's meta characters

Using *"expr"* for String handling

- ***expr*** can also be used in string manipulation- to calculate length, locate position of a character or extract a substring.
- Examples:-
  - *$expr "Are you free?" : '.*'*
    13
  - *$expr length "abcd"*
    4
  - *$expr substr "thbs" 2 2*
    hb

Note that space should be on either side of colon(:)


STOP