Shell Scripting – Part II

Bash Script – Practice

- Write a bash script called *helpful.sh*
- On being run it should do the following automatically:
  - Make a directory ~/Documents/AutoDir
  - Fetch a file from the internet url home.manhattan.edu/~kqazi01/main.c into that directory
  - Create a symbolic link for that file called m.c
  - Display the number of lines in that file

Bash Script

- Shell scripting (including bash) has quite powerful features
    - Variables
    - Arithmetic operations
    - If-elif-else
    - Loops
    - File operations
- Almost like a full-fledged language

Bash Script – Variables

- Bash has variables to store data
- Store absolute values like strings or numbers
- Store the output of commands (which are in fact strings or numbers)
- Use the variables just like any other programming language

Bash Script – Variables

```
#!/bin/bash

greeting="Welcome"
user=$(whoami)
day=$(date)
echo "$greeting back $user! Today is $day, which is the best day of the entire week!"
echo "Your Bash shell version is: $BASH_VERSION."
```

Bash Script – Variables

- Notice the syntax of variables carefully
  - When declaring it is
    varname=value
  - When using it is
    $varname
- Notice, no need to append to string
- Try

```
v1="Hello"
v2="World"
v3=$v1$v2
echo $v3
```

Bash Script – Variables

- Variable types can be changed based on context
- Try this:

```
x=1
y=2
echo “$x $y”
```

Bash Script – Variables

- Outputs of commands can be stored in variables
- Try this:

```
var=$(ls)
echo “var”
```

Bash Script – Variables

- The variable $BASH_VERSION is one of a number of built-in variables
- Each of these holds some useful information
- These offer a lot of useful features in scripts

Bash Script – Input/Output

- We tried the command *echo* for output
- For input from the keyboard, the command *read* can be used

  read var1

- Wait for the user to enter keyboard input and store it in variable var1

Bash Script – Input Arguments

- Another way of taking input is through command line arguments
- For example

```
python myscript.py
ls -al
./somescript.sh arg1 arg2 arg2
```

- The values passed when running any program from a terminal, are command line arguments

Bash Script – Input Arguments

- To access the arguments within your script some built-in variables exist (remember those?)

  $1, $2, $3, etc. argument 1, 2, 3, etc.
  $0 the name of the script itself
  $# the number of arguments passed
  $@ values of all the arguments
  $* values of all the arguments (double quoted)
  $! Pid of last command
  $$ pid of current shell

- Let's try some examples

Bash Script – Arithmetic Operations

- Syntax to perform arithmetic ops is:

```
$(( expression ))
```

- Let's try some examples

```
ans=$(( 75 + 3 ))
echo "$ans"
v1=5
v2=4
echo $(( $v1 + $v2 ))
```

Bash Script – Arithmetic Operations

- Operations offered are:

++, -- (both pre and post)
-, +, *, /, %, **
~, >>, <<, &, |, ^

Bash Script – Conditional Statements

- General syntax

```
if boolean_expression
then
        statements
fi
```

- If-elif-else

```
if boolean_expr
then
        statements
elif boolean_expr
then
        statements
else
        statements
fi
```

Bash Script – Conditional Statements

- **The boolean expression can be any command that returns an exit status!!!**
  e.g. **if grep cobain Alice**
- To make comparisons, **three** *'test'* syntax exist
- Think of [ ], [[ ]], and (( )) as commands. **Notice the spacing**

  1) if [expr1 comparison expr2]
     e.g. if ["hello" = "world"]

  2) if [[expr1 comparison expr2]]
     More features, very useful when using regular expressions

  3) if ((expr1 comparison expr2))
     offers more C like syntax

- We will use syntax 1 most of the time (differences discussed later)

Bash Script – Conditional Statements

- To compare **strings** use =, !=, >, <,

```
read var1
read var2
if [ $var1 = $var2 ]
…
```

- To compare **numbers** however, use -eq, -ne, -gt, -lt, -ge, -le

```
var1=10
var2=20
if [ $var1 -eq $var2 ]
…
```

Try comparing numbers 1 and 001 using = and -eq to see the difference

Bash Script – Conditional Statements

- To test **files** use -e, -d, -L, -r, -w, -x, -s, etc.

  if [-e path/filename]

  checks if path/filename exists

- Summary explanation given on later slide

Bash Script – Conditional Statements

- Finally, combine conditions using && and ||

```
g=1
c=456
if [ "$g" -eq 1 ] && [ "$c" = "123" ]
then
        echo “First Match”

elif [ "$g" -eq 1 ] && [ "$c" = "456" ]
then
        echo “Second Match”

else
        echo “No Match”

fi
```

- What will print?

Bash Script – Conditional Statements

Strings

Str1 = Str2	Returns true if the strings are equal
Str1 != Str2	Returns true if the strings are not equal
-n Str1	Returns true if the string is not null
-z Str1	Returns true if the string is null

Number

expr1 -eq expr2	Returns true if the expressions are equal
expr1 -ne expr2	Returns true if the expressions are not equal
expr1 -gt expr2	Returns true if expr1 is greater than expr2
expr1 -ge expr2	Returns true if expr1 is greater than or equal to expr2
expr1 -lt expr2	Returns true if expr1 is less than expr2
expr1 -le expr2	Returns true if expr1 is less than or equal to expr2

Files

-d file	True if the file is a directory
-e file	True if the file exists
-L file	True if the file is a symbolic link
-r file	True if the file is readable
-w file	True if the file is writable
-x file	True if the file is an executable