CS242: Shell Programming

TA: Aparajita Dutta

Course Instructor: Ashish Anand

Indian Institute of Technology Guwahati

Introduction to Shell

- Program that enables the system to understand user commands
- Also called command interpreter (not a compiler)
- Different Unix shells available:
  - Korn Shell
  - Bourne shell
  - Bourne-Again shell (BASH, possibly the most popular shell today)
- Commands:
  - cat etc/shells from root directory (displays all supported shells)
  - which bash (displays the location of bash)

Shell Flavors

Program name	The corresponding shell
/bin/sh	Bourne shell
/bin/bash	The Bash shell
/bin/ksh	Korn shell

- /etc/passwd file determines which shell is effective in current session

Command line structure in shell

- Single word commands
  - who
  - date
- Multi word commands
  - echo “hi”
  - echo “hi” > filename
- Combining commands using semicolon or pipe
  - date; who
  - (date;who) | wc -l

Creating new commands

```
$ who | wc -l
```

```
$ echo ‘who | wc -l’ > number_users (why are quotes needed here?)
```

```
$ sh < number_users
```

What if the command contains variables?

Program output as arguments

```
$ echo the user is `who`
```

Output: the user is cse tty7 2019-08-30 14:57 (:0)

Shell variables

- Predefined shell variables (system variables)
  - Usually denoted in capital letters
    - $PATH (echo $PATH)
    - $PWD (echo $PWD)
- User defined variables
  - Usually in lower case

The first shell program: Hello World!

```
#! /bin/bash            #shebang

# this is a comment

:<<'END'

This is a

Multiline comment

END

echo "hello world!"    # this is also a comment
```

- Shebang specifies the absolute path to the Bash interpreter.
- Have to make the script file executable first using chmod

User defined variables

```
name=Mark

echo “My name is $name”
```

- No space before and after the equal sign
- Variable names should not start with numbers

Read user input

echo “Enter name:”

read name

echo “The name entered is $name”

Output: The name entered is ……

- How can you enter input after the question prompt without a new line? (-p flag)
- How to hide input on screen while reading? (-s flag)
- How to read array as input? (-a flag)

Reading and accessing input as array

```
echo "enter the array of numbers: "

read -a nums

echo "the numbers entered are: ${nums[0]} ${nums[2]}"
```

Pass arguments to a bash script

```
echo The arguments passed are: $0 $1 $2 $3
```

```
$ ./pass_args.sh hi hello bye
```

Output:
```
The arguments passed are: ./pass_args.sh hi hello bye
```

- How to pass arguments as array?
- How to print the number of arguments passed?

If-elif-else condition

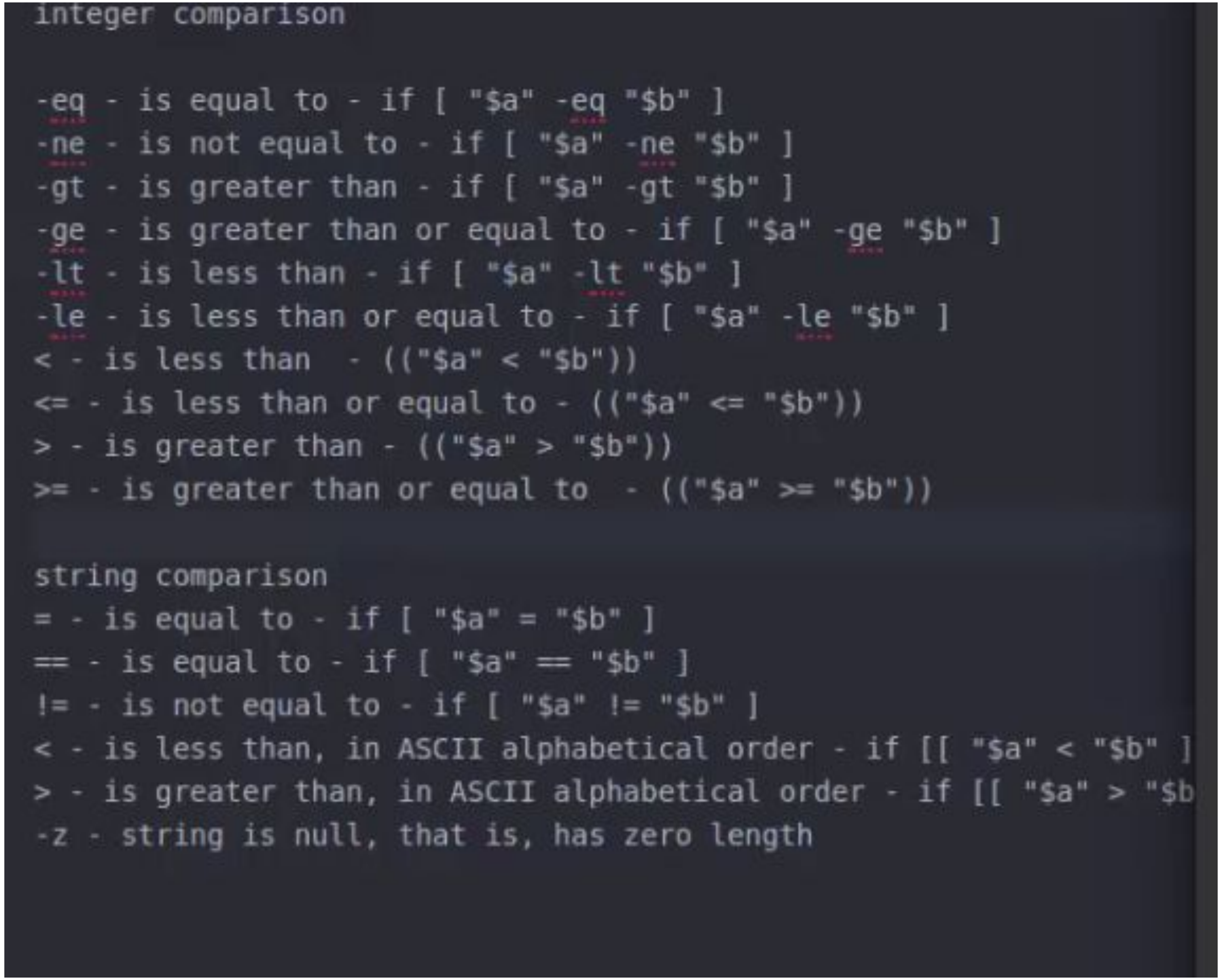
```
if [ condition ]
then
        statement
fi
```

```
if [ condition ]
then
        statement
else
        statement
fi
```

```
if [ condition ]
then
        statement
elif [ condition ]
then
        statement
else
        statement
fi
```

- **You can have nested if-else conditions**

If-elif-else condition



```
integer comparison

-eq - is equal to - if [ "$a" -eq "$b" ]
-ne - is not equal to - if [ "$a" -ne "$b" ]
-gt - is greater than - if [ "$a" -gt "$b" ]
-ge - is greater than or equal to - if [ "$a" -ge "$b" ]
-lt - is less than - if [ "$a" -lt "$b" ]
-le - is less than or equal to - if [ "$a" -le "$b" ]
< - is less than  - (("$a" < "$b"))
<= - is less than or equal to - (("$a" <= "$b"))
> - is greater than - (("$a" > "$b"))
>= - is greater than or equal to  - (("$a" >= "$b"))

string comparison
= - is equal to - if [ "$a" = "$b" ]
== - is equal to - if [ "$a" == "$b" ]
!= - is not equal to - if [ "$a" != "$b" ]
< - is less than, in ASCII alphabetical order - if [[ "$a" < "$b" ]
> - is greater than, in ASCII alphabetical order - if [[ "$a" > "$b
-z - string is null, that is, has zero length
```

Combining conditions

Logical AND (&& or -a)

Logical OR (|| or -o)

```
if [ $num -gt 0 ] || [ $num -lt 0 ]
then
    echo "number is non zero"
else
    echo "number is zero"
fi
```

```
if [ $num -gt 0 ] || [ $num -lt 0 ]
```

```
if [ $num -gt 0  -o  $num -lt 0 ]
```

```
if [[ $num -gt 0  ||  $num -lt 0 ]]
```

Arithmetic operations

```
num1=20
num2=5

echo $(( num1 + num2 ))
echo $(( num1 - num2 ))
echo $(( num1 * num2 ))
echo $(( num1 / num2 ))
echo $(( num1 % num2 ))
```

Output: $ ./arithmetic_ops.sh

```
25
15
100
4
0
```

```
num1=20
num2=5

echo $(expr $num1 + $num2 )
echo $(expr $num1 - $num2 )
echo $(expr $num1 \* $num2 )
echo $(expr $num1 / $num2 )
echo $(expr $num1 % $num2 )
```

Output: $ ./arithmetic_ops.sh

```
25
15
100
4
0
```

Floating point math operations

```
num3=20.5
num4=5

echo "$num3+$num4" | bc
echo "20.5-5" | bc
echo "20.5*5" | bc
echo "scale=3;20.5/5" | bc
echo "20.5%5" | bc
echo "scale=2;sqrt($num3)" | bc -l
```

Case-esac

```
case expression in
    pattern1 )
            statements ;;
    pattern2 )
            statements ;;
esac
```

```
vehicle=$1

case $vehicle in
    "car" )
            echo "Rent of $vehicle is 100 dollars" ;;
    "van" )
            echo "Rent of $vehicle is 80 dollars" ;;
    "bicycle" )
            echo "Rent of $vehicle is 5 dollars" ;;
    * )
            echo "unknown vehicle" ;;
esac
```

Array

```
os=('apple' 'banana' 'guava')

echo "${os[@]}"
echo "${os[2]}"
echo "${!os[@]}"
echo "${#os[@]}"
```

Output:

```
apple banana guava
guava
0 1 2
3
```

- You can add elements
- You can delete elements
- You can update elements

Loops: While

```
while [ condition ]
do
    command1
    command2
    command3
done
```

```
n=1

while [ $n -le 5 ]
do
    echo $n
    n=$(( n+1 ))
done
```

Output:

```
$ ./while.sh
1
2
3
4
5
```

Read file using while loop

```
while read p
do
    echo $p
done < hello.sh
```

```
cat hello.sh | while read p
do
    echo $p
done
```

Until loop

```
n=1

until [ $n -ge 10 ]
do
    echo $n
    n=$(( n+1 ))    (( n++ ))
done
```

Output:

```
$ ./until.sh
1
2
3
4
5
6
7
8
9
```

For loop

```
for (( i=0; i<5; i++ ))
do
    echo $i
done
```

Output:

```
$ ./forloop.sh
0
1
2
3
4
```

Select loop

```
select name in mark john mary kate
do
    echo "$name selected"
done
```

Output:

```
1) mark
2) john
3) mary
4) kate
#? 2
john selected
#?
```

- Read the usage of **break** and **continue** keywords

Functions

```
function name(){
    command
}


name () {
    commands
}
```

```
#! /bin/bash

function Hello(){
    echo "Hello world!"
}

quit () {
    exit
}

Hello
quit
```

Output ?

Pass arguments in function

```
function print(){
    echo $1 $2
}

quit () {
    exit
}

print Hello World
print World
quit
```

Output ?

- Keyword ‘**local**’ to assign local variable

Debugging a bash script

bash -x prog.sh

set -x (starts debugging)

set +x (ends debugging)

References

http://jatinga.iitg.ernet.in/~asahu/cs241/A3/reference_bash-cheatsheet.pdf

http://jatinga.iitg.ernet.in/~asahu/cs241/A3/AWK.and.shell_Questions.pdf

http://jatinga.iitg.ernet.in/~asahu/cs241/ShellM/ShellQuestionPartII.pdf

http://jatinga.iitg.ernet.in/~asahu/cs241/A5/Adv-Shell-Prob.pdf

https://www.youtube.com/watch?v=m4G3MLK8l4s&list=PLS1QulWo1RIYmaxcEqw5JhK3b-6rgdWO_&index=8

Books: The Unix Programming Environment

Unix in a Nutshell

http://jatinga.iitg.ernet.in/~asahu/cs241/A3/Linux.Shell.Scripting.Cookbook.pdf