Lecture 6

Bash scripting continued

1

1

Exit Status

- Every Linux command returns an integer code when it finishes, called its "**exit status**"
  - 0 usually* denotes success, or an OK exit status
  - Anything other than 0 (1 to 255) usually denotes an error
- You can return an exit status explicitly using the **exit** statement
- You can check the status of the last command executed in the variable **$?**

```
$ cat someFileThatDoesNotExist.txt
$ echo $?
1            # "Failure"
$ ls
$ echo $?
0            # "Success"
```

* One example exception: `diff` returns "0" for no differences, "1" if differences found, "2" for an error such as invalid filename argument

2

2

if/else

```
if [ condition ]; then          # basic if
    commands
fi

if [ condition ]; then          # if / else if / else
    commands1
elif [ condition ]; then
    commands2
else
    commands3
fi
```

- The `[ ]` syntax is actually shorthand for a shell command called "***test***" (Try: "man test")
- there ***MUST*** be spaces as shown:
  `if` space `[` space ***condition*** space `]`
- include the semi-colon after `]` (or put "`then`" on the next line)

3

3

The test command

```
$ test 10 -lt 5
$ echo $?
1            # "False", "Failure"
$ test 10 -gt 5
$ echo $?
0            # "True", "Success"
```

- Another syntax for the test command:
  Don't forget the space after [and before]

```
$ [ 10 -lt 5 ]
$ echo $?
1            # "False", "Failure"
$ [ 10 -gt 5 ]
$ echo $?
0            # "True", "Success"
```

4

4

test operators

comparison operator	description
=, !=, \<, \>	compares two **string** variables
-z, -n	tests if a string is empty (zero-length) or not empty (nonzero-length)
-lt, -le, -eq, -gt, -ge, -ne	compares **numbers**; equivalent to Java's <, <=, ==, >, >=, !=
-e, -f, -d	tests whether a given file or directory exists
-r, -w, -x	tests whether a file exists and is readable/writable/executable

```
if [ $USER = "husky14" ]; then
    echo 'Woof! Go Huskies!'
fi
```

```
LOGINS=$(w -h | wc -l)
if [ $LOGINS -gt 10 ]; then
    echo 'attu is very busy right now!'
fi
```

*Note: `man test` will show other operators.

5

5

More if testing

compound comparison operators	description
if [*expr1* -a *expr2*]; then ... if [*expr1*] && [*expr2*]; then ...	and
if [*expr1* -o *expr2*]; then ... if [*expr1*] || [*expr2*]; then ...	or
if [! *expr*]; then ...	not

```
# alert user if running >= 10 processes when
# attu is busy (>= 5 users logged in)
LOGINS=$(w -h | wc -l)
PROCESSES=$(ps -u $USER | wc -l)
if [ $LOGINS -ge 5 -a $PROCESSES -gt 10 ]; then
    echo "Quit hogging the server!"
fi
```

6

6

safecopy Exercise

- Write a script called `safecopy` that will mimic the behavior of `cp –i` where *from* is a filename and *to* is a filename:

```
$ cp –i from.txt to.txt
Do you want to overwrite to.txt? (yes/no)
```

```
$ ./safecopy.sh from.txt to.txt
Do you want to overwrite to.txt? (yes/no)
```

7

7

safecopy Exercise Solution

```
#!/bin/bash

FROM=$1
TO=$2

if [ -e $TO ]; then
    read -p "Do you want to overwrite $TO?" ANSWER
    if  [ $ANSWER = "yes" ]; then
        cp $FROM $TO
    fi
else
    cp $FROM $TO
fi
```

8

8

BMI Exercise

- Write a program that computes the user's body mass index (BMI) to the nearest integer, as well as the user's weight class:

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
≤ 18	underweight
18 - 24	normal
25 - 29	overweight
≥ 30	obese

```
$ ./bmi.sh
Usage: ./bmi.sh weight height

$ ./bmi.sh 112 72
Your Body Mass Index (BMI) is 15
Here is a sandwich; please eat.

$ ./bmi.sh 208 67
Your Body Mass Index (BMI) is 32
There is more of you to love.
```

9

9

BMI Exercise solution

```
#!/bin/bash
# Body Mass Index (BMI) calculator
if [ $# -lt 2 ]; then
    echo "Usage: $0 weight height"
    exit 1        # 1 indicates failure, 0 for success
fi

let H2="$2 * $2"
let BMI="703 * $1 / $H2"
echo "Your Body Mass Index (BMI) is $BMI"
if [ $BMI -le 18 ]; then
    echo "Here is a sandwich; please eat."
elif [ $BMI -le 24 ]; then
    echo "You're in normal weight range."
elif [ $BMI -le 29 ]; then
    echo "You could stand to lose a few."
else
    echo "There is more of you to love."
fi
```

10

10

Common errors

- `[: -eq: unary operator expected`
  - you used an undefined variable in an `if` test
- `[: too many arguments`
  - you tried to use a variable with a large, complex value (such as multi-line output from a program) as though it were a simple int or string
- `let: syntax error: operand expected (error token is " ")`
  - you used an undefined variable in a `let` mathematical expression

11

11

`while` and `until` loops

```
while [ condition ]; do   # go while condition is true
    commands
done


until [ condition ]; do   # go while condition is false
    commands
done
```

12

12

While exercise

- Prompt the user for what they would like to do. While their answer is "open the pod bay doors" tell them that you cannot do that and prompt for another action.

13

13

While Exercise solution

```
#!/bin/bash
# What would you like to do?
 read -p "What would you like me to do? " ACTION
 echo "You said: $ACTION"
 while [ "$ACTION" = "open the pod bay doors" ]; do
     echo "I'm sorry Dave, I'm afraid I can't do that."
     read -p "What would you like me to do? " ACTION
     echo "You said: $ACTION"
 done
 echo "Bye"
```

The quotes around "$ACTION" are important here, try removing them and see what happens.

14

14

select and case

- Bash Select statement:

```
PS3=prompt  # Special variable* for the select prompt
select choice in choices; do
       commands
       break # Break, otherwise endless loop
 done
```

- Bash Case statement:

```
case EXPRESSION in
   CASE1) COMMAND-LIST;;
   CASE2) COMMAND-LIST;;
   ...
   CASEN) COMMAND-LIST;;
esac
```

*see lecture 5

15

15

Select Example

```
PS3="What is your favorite food? " # Goes with the select stmt

echo "Welcome to the select example!"
echo "It prints out a list of choices"
echo "but does nothing interesting with the answer."

select CHOICE in "pizza" "sushi" "oatmeal" "broccoli"; do
    echo "You picked $CHOICE"
    break
done

echo "For the select statement, you pick a number as your choice."
```

16

16

Case Example

```
echo "Welcome to the case example!"
echo "Without a select statement, you must get the spelling/case exact.
read -p "What format do you prefer? (tape/cd/mp3/lp) " FORMAT
echo "You said $FORMAT"

case "$FORMAT" in
    "tape") echo "no random access!";;
    "cd") echo "old school";;
    "mp3") echo "how modern";;
    "lp") echo "total retro";;
esac
```

17

17

select/case Exercise

- Have the user select their favorite kind of music, and output a message based on their choice

18

18

select/case Exercise Solution

```
PS3="What is your favorite kind of music? "
select CHOICE in "rock" "pop" "dance" "reggae"; do
    case "$CHOICE" in
        "rock") echo "Rock on, dude.";;
        "pop") echo "Top 100 is called that for a reason.";;
        "dance") echo "Let's lay down the Persian!";;
        "reggae") echo "Takin' it easy...";;
        * ) echo "come on...you  gotta like something!";;
    esac
    break
done
```

19

19

Arrays

name`=(`***element1 element2 ... elementN***`)`

name`[`***index***`]=`***value***	`# set an element`
`$`***name***	`# get first element`
`${`***name***`[`***index***`]}`	`# get an element`
`${`***name***`[*]}`	`# elements sep.by spaces`
`${#`***name***`[*]}`	`# array's length`

- arrays don't have a fixed length; they can grow as necessary
- if you go out of bounds, shell will silently give you an empty string
  - you don't need to use arrays in assignments in this course

20

20

Functions

```
function name() {       # declaration
    commands            # ()'s are optional
}

name                    # call
```

- functions are called simply by writing their name (no parens)
- parameters can be passed and accessed as $1, $2, etc. (icky)
  - you don't need to use functions in assignments in this course

21

21