CSCI 211
UNIX Lab

Shell Programming

Dr. Jiang Li

Why Shell Scripting

- Saves a lot of typing
  - A shell script can run many commands at once
  - A shell script can repeatedly run commands
- Help avoid mistakes
  - Once the script gets things right, it will always be right.
- Customize your job easily
  - Arguments, variables and input/files can be used for tailoring

A fast way to create some tools, utilities & apps

Jiang Li, Ph.D.
Department of Computer Science

Shell in Linux/UNIX

- What is a shell
  - A user program provided as the interface between the users and the OS
  - When a user input a command, the shell is responsible for interpreting and executing the command
- Different Shells
  - BASH (Bourne-Again SHell) - Most common shell in Linux.
  - CSH (C Shell)
  - KSH (Korn Shell)
  - TCSH

Jiang Li, Ph.D.
Department of Computer Science

Basic Ideas of Scripting

- Commands
  - Those you normally run in command line
  - A list of commands are executed one by one
- Control constructs
  - Specify the way to execute enclosed commands
    - Alternatively
    - Repeatedly

Get Started

- Type the following two commands

```
clear
echo "Hello, world!"
```

- Use vi to create a file "hewo.sh"
  - Write the two commands in the file
  - Save and exit
- Grant the permission to execute this file
  - Run `chmod +x hewo.sh`
- Run the script by typing `./hewo.sh`

First Line and Comment

- `#!/bin/bash`
  - As the first line, specify the shell that runs the script (`/bin/bash` is the path of shell command)
- Comments
  - A word or line beginning with # causes that word and all remaining characters on that line to be ignored.
  - A comment start with #

Variable

- Assignment format

  (assigning ‘value’ to variable ‘var’)

  ```
  var=value
  ```

  No blank space!

- To refer to the variable ‘var’ use the following:

  ```
  $var
  ```

- Example script:

```
str1="My"
str2="variable"
var1="$str1 first $str2"
echo $var1, $str1 second $str2
```


Jiang Li, Ph.D.
Department of Computer Science

Quotes

- Compare the following

```
str1="My"
str2="variable"
var1='$str1 first $str2'
var2="$str1 first $str2"
echo $var1
echo $var2
```

- Escape quote

```
echo  $var2\'s great.
```

Capture Command Output

- `var=`command``
  - Example: `var=`ls | wc -l``
- `var=$(command)`
  - Example: `var=$(ls | wc -l)`

Arithmetic

- `var=$((expression))`
  - Example: `i=$((i * 2 - 1))`

Read User Input

- `read var`
- `read -p "please enter: " var`
- `read -sp "please enter: " var`

- Try each of the above then `echo $var`

Command Line Arguments

- Values passed to the script from the invoking command
  - Example: `myscript "a string" 123`
- Represented by a dollar sign ($) followed by a number from 0 to 9
  - $0: the name of the script; $1: the first parameter; $2: the second parameter; and so on
  - Example: `echo $1, $2`

Jiang Li, Ph.D.
Department of Computer Science

Conditional Constructs

```
if [ condition1 ]
then
   command1
   command2
   command3
elif [ condition2 ]
then
   command4
   command5
else
   default-command
fi
```

```
if [ condition1 ];  then
   command1
   command2
   command3
elif [ condition2 ]; then
   command4
   command5
else
   default-command
fi
```

Example Conditions

- `[ -e FILE ]`: True if file exists
- `[ -f FILE ]`: True if FILE exists and is a regular file.
- `[ -s FILE ]`: True if FILE exists and has a size greater than zero.
- `[ -d FILE ]`: True if FILE exists and is a directory.
- `[ -w FILE ]`: True if FILE exists and is writable
- `[ -x FILE ]`: True if FILE exists and is executable.
- `[ STRING1 == STRING2 ]`: True if the two strings have the same values.
- `[ STRING1 != STRING2 ]`: True if the strings are not equal.
- `[ STRING1 < STRING2 ]`: True if "STRING1" sorts before "STRING2" with lexicographic order.
- `[ STRING1 > STRING2 ]`: True if "STRING1" sorts after "STRING2" with lexicographic order.
- `[ NUMBER1 -eq NUMBER2 ]`: True if Number1 is equal to Numbers
  - Also -gt, -ge, -lt, -le

Jiang Li, Ph.D.
Department of Computer Science

And, Or, and Not in conditions

- And: `-a`
  - Example:

```
if [ $a == "a" -a $n -eq 1 ]; then …
```

- Or: `-o`
  - Example:

```
if [ $a == "a" -o $n -eq 1 ]; then …
```

- Not: `!`
  - Example:

```
if [ ! $a == "a" ]; then …
```


Jiang Li, Ph.D.
Department of Computer Science

The `for` Loop

```
for var in item1 item2 … itemN; do
   command1
   command2
   ....
   commandN
done
```


Jiang Li, Ph.D.
Department of Computer Science

A `for` Example

- Create a script `hewo.sh` with the following content. Run it.

```
#!/bin/bash
for i in 1 2 3 4 5; do
    echo "Welcome $i times."
done
```

```
$ ./hewo.sh
Welcome 1 times.
Welcome 2 times.
Welcome 3 times.
Welcome 4 times.
Welcome 5 times.
```

A `for` Loop Handling Files

```
#!/bin/bash
for f in `ls`; do
  if [ -d $f ]; then
    echo "$f is a directory."
  elif [ -f $f ]; then
    echo "$f is a file."
  else
    echo "Don’t know what $f is.“
  fi
done
```


Jiang Li, Ph.D.
Department of Computer Science

`While` Loop

- `While` loop executes as long as its condition is true.

```
while [ condition ]
do
   commands
done
```


Jiang Li, Ph.D.
Department of Computer Science

While Loop Example

```
#!/bin/bash
n=1
while [ $n -le 5 ]; do
  echo "Welcome $n times."
   n=$(( n+1 ))
done
```


Jiang Li, Ph.D.
Department of Computer Science

Read from Files

- ```
  while read var; do
      echo var: $var
  done < filename
  ```
- ```
  while read var1 var2; do
      echo var1: $var1
      echo var2: $var2
  done < filename
  ```

Functions

- What is ***function***
  - A *function* is a part of a script that performs a specific sub-task and that can be called by its name
- Example

```
hello() {
    echo "Hello world!"
}

hello
```

Parameters

- Same as command line arguments
- Parameters are represented by a dollar sign ($) followed by a number from 0 to 9
  - $0: the name of the script; $1: the first parameter; $2: the second parameter; and so on
  - Example:

```
hello() {
    echo "Hello $1 $2, let us be friend."
}
hello John Smith
```


Jiang Li, Ph.D.
Department of Computer Science