Session 4 - Shell Scripting

Bash Scripting

What is a shell script? In the simplest terms a shell script is a text file which contains a sequence of shells commands to be run by the computer. The commands in a shell script are the same kind of commands you would type in while interacting with the shell. In our case we will be learning to write Bash scripts.
Why write shell scripts? There are several reasons to write shell scripts:

- they allow you to create tools which you can use over and over
- they are useful for automation of repetitive tasks
- they are useful in rapid prototyping of program designs which will be rewritten later in a more suitable programming language
- your Linux system uses shell scripts extensively while it is booting so being able to write and modify shell scripts will be useful if you wish to become proficient at system administration

Simple Hello World shell script

```
vi simple1.sh
```

```
printf "Hello World\n"
```

```
bash ./simple1.sh
```

Modified Hello World script

```
vi simple2.sh
```

```
#!/bin/bash

# Author: John Doe
# Purpose: This script will greet the user
# Version: 1.0

read -p "Please enter your name: " myname
printf "Hello %s, pleased to meet you.\n" $myname
```

```
chmod +x simple2.sh
./simple2.sh
```

Simple2 script explanation

1. The first line in the script is the “interpreter directive” and effectively tells the system what shell (or other interpreter) to use when this script is run. The first two characters `#!` (sometimes called 'shebang') are a *magic number*. Magic numbers are a series of a few characters at the beginning of a file which the operating system uses to determine what type of file it is. There are many different magic numbers corresponding to the many file types on a Linux system.
2. The lines beginning with a pound sign `#` are comments. The comment which starts after the `#` are ignored by the shell. The comment extends to the end of the line.
3. The read command prints a prompt for the user using the -p parameter and stores the input value into a variable called myname. You should give your parameters meaningful names. Parameter can be uppercase, lowercase, mixed, and contain numbers, but they may not start with a number. Some good examples are: firstname, lastname, name1, filename2, currenttime, userinput, first_name, last_name, file_name, curr_time, user_input, etc…
4. The printf command prints a string with a format specifier (`%s`) to standard output (stdout). When the string is printed the format specifier is replaced with the value stored in the second parameter.

sysinfo script

`vi sysinfo.sh`

```
#!/bin/bash

# Author: John Doe
# Purpose: This script will print out information about the system
# Version: 1.0

clear
printf "Hello, $USER\n"
printf "\n"
printf "Today's date is $(date), this is week $(date +"%V").\n"
printf "\n"
printf "These users are currently connected:\n$(who)\n"
printf "\n"
printf "This is $(sed -n '/PRETTY_NAME/s/.*=//p' /etc/os-release) "
printf "running on an $(uname -m) processor.\n"
printf "\n"
printf "This is the uptime information:\n"
uptime
printf "\n"
printf "This is the physical RAM information:\n"
cat /proc/meminfo | grep MemTotal
printf "\n"
exit 0
```

```
chmod +x sysinfo.sh
./sysinfo.sh
```

sysinfo script explanation

1. the $USER variable is an environment variable. Environment variables are available for use within shell scripts.
2. Commands between $(...) run in a child process and their output is inserted into the string.
3. sed is a stream editor we will only mention briefly
4. `/proc/meminfo` is a virtual file which contains system memory information
5. `exit 0` is an exit status returned to the calling environment indicating success

In-class activity 1

Work through the vipractice.txt file provided by your instructor. The instructor will guide the class together through this exercise.

In-class activity 2

Using `simple2.sh` as an example, write your own Bash script named `greeting.sh` which

- prompts the user to enter their first name
- then prompts the user to enter their last name
- prints a greeting similar to "Hello John Doe, pleased to meet you."

In-class activity 3

Follow along as the instructor steps through a slightly more complex script and ways that you can debug a script.

Variables

Variables in a Bash script are used to hold or contain values. Let us carefully distinguish between the *name* of a variable and its *value*. If **variable1** is the name of a variable, then **$variable1** is a reference to its *value*, the data item it contains.

Example of variables

```
joe@joe-VirtualBox:~$ my_name=Joe
joe@joe-VirtualBox:~$ echo my_name
my_name
joe@joe-VirtualBox:~$ echo $my_name
Joe
joe@joe-VirtualBox:~$ printf "%s\n" my_name
my_name
joe@joe-VirtualBox:~$ printf "%s\n" $my_name
Joe
```

http://tldp.org/LDP/abs/html/varsubn.html

Flow Control Structures

if .. elif .. else

```
if [ condition ]; then
  commands
elif [ condition ]; then
  commands
else
  commands
fi
```

Example of if .. elif .. else structure

```
read -p "Please enter a number between 1 and 100: " number
if [ $number -lt 0 ]; then
  printf "The number you entered is too low.\n"
elif [ $number -gt 100 ]; then
  printf "The number you entered is too high.\n"
else
  printf "You entered a good number: %d\n" $number
fi
```

http://tldp.org/LDP/abs/html/testconstructs.html

case statements

```
case $variable in
  "value1")
    commands
    ;;
  "value2")
    commands
    ;;
  "value3")
    commands
    ;;
  *)
    commands
    ;;
esac
```

Example of a case statement

```
read -p "Please enter the kind of pie you want: " pie_choice
case $pie_choice in
  "Apple")
    printf "Your apple pie is coming right up.\n"
    ;;
  "Peach")
    printf "Your peach pie is coming right up.\n"
    ;;
  "Cherry")
    printf "Your cherry pie is coming right up.\n"
    ;;
  *)
    printf "Sorry, we don't have %s, how about chocolate?\n" $pie_choice
    ;;
esac
```

http://tldp.org/LDP/abs/html/testbranch.html

for .. do loops

```
for variable in $somelist
do
  commands
done
```

Example for .. do loop

```
for planet in Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto
do
  if [ "$planet" = "Pluto" ];
  then
    printf "Sorry, Pluto is no longer a planet.\n"
  else
    printf "%s is a planet.\n" $planet
  fi
done
```

Another type of for .. do loop

For the example below LCV means loop control variable

```
for (( initialize LCV; test LCV; update LCV ))
do
  commands
done
```

Another example of a for .. do loop

```
j=0
for (( i=0; i<10; i++ ))
do
  j += i
  printf "%d\n" $j
done
```

while .. do

```
while [ evaluation ]
do
  commands
done
```

Example while .. do loop

```
user_input="n"
while [ "$user_input" = "n" ]
do
  read -p "Are you done yet? [y/n] " user_input
done
```

until ... do

```
until [ evaluation ]
do
  commands
done
```

Example until .. do loop

```
user_input="n"
until [ $user_input = "y" ]
do
  read -p "Are you done yet? [y/n] " user_input
done
```

http://tldp.org/LDP/abs/html/loops1.html

Functions

Like other programming languages, Bash has functions, though in a somewhat limited implementation. A function is a subroutine; a code block that implements a set of operations, a "black box" that performs a specified task. Wherever there is repetitive code, when a task repeats with only slight variations in procedure, then consider using a function.

Simple Function (no parameters)

```
function_name(){
  command...
}
```

Example of function without parameters

```
greeting(){
  read -n "Please enter your name: " name
  printf "Hello %s\n" $name
}
```

Function with parameters

```
function_name(){
  commands using $1, $2, $3 etc...
}
```

Example of function with parameters

```
print_fibonacci(){
  f1=0
  f2=2

  for (( i=0; i<=$1; i++ ))
  do
    printf "%d " $f1
    fn=$((f1+f2))
    f1=$f2
    f2=$fn
  done
}
```

http://tldp.org/LDP/abs/html/functions.html

Secure Shell (SSH)

Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. The best known example application is for remote login to computer systems by users.

SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application with an SSH server. Common applications include remote command-line login and remote command execution, but any network service can be secured with SSH.

The most visible application of the protocol is for access to shell accounts on Unix-like operating systems
.
SSH was designed as a replacement for telnet and for unsecured remote shell protocols such as the Berkeley rlogin, rsh, and rexec protocols. Those protocols send information, notably passwords, in plaintext, rendering them susceptible to interception and disclosure using packet analysis. The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet, although files leaked by Edward Snowden indicate that the National Security Agency can sometimes decrypt SSH, allowing them to read the contents of SSH sessions.

https://en.wikipedia.org/wiki/Secure_Shell

To install sshd type the following in a terminal window

```
sudo apt-get install ssh
Do you want to continue? [Y/n] y
```

Session 4 Homework

Install an SSH server on your local machine and try to login to it (from itself) using SSH. Use netstat to make sure port 22 is open.

References:

https://en.wikipedia.org
http://www.webopedia.com/quick_ref/computer-architecture-study-guide.html
http://tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf
http://www.tldp.org/LDP/abs/abs-guide.pdf