INTRO TO BASH SHELL SCRIPTING

JOHN KENNEDY

SKEBI69@GMAIL.COM

Presentation: https://gitpitch.com/skebi69/IntroBash
Source: https://github.com/skebi69/IntroBash

DISCLAIMER

- As in most of Linux, there is more than one way to do things. My way may not always be the best.
- Some of my techniques and methods may be older/outdated. I am old.
- There are going to be somethings that I miss out on (arrays) either on purpose or by accident.

NOTES

- I would like to have built a usable script but did not have time so:
- Hopefully my examples will be easily modified to fit your needs

TOPICS

- Why script?
- She-Bang - Not Ricky Martin (see - I told you I was old)
- Variables
- Special variables
- User input

TOPICS (CONT)

- Conditional handling
- Select/case text menu
- Loops
- Functions

WHY SCRIPT?

- Run series of commands
- Build a long, complicated command
- Run commands that depend on specific conditions/results
- Run repetitive commands
- A lazy sysadmin is a good sysadmin - Let your scripts do the work for you

SHE BANG (#!)

- Start every script to set interpreter
- #!/bin/bash

VARIABLES

- A temporary store for small pieces of information
- No need to declare
- Can contain any type of data - String, integer, float, etc
- Good practice to surround with curly braces "{}"
- Like file names, try and avoid spaces in variables where possible

OTHER TYPES OF VARIABLES

- Special variables
- Arrays (not covered today)

SPECIAL VARIABLES

- $? - Exit code of last command
- $0 - Script name
- $1 - $N - Option 1 to whatever
- $$ - PID of script
- $IFS - Internal field separator - Normally set to whitespace

USER INPUT

SYNTAX

```
read <options> variable
```

EXAMPLES OF P AND S OPTIONS

-p - Prompt

```
read -p "Enter a number: " NUM
echo ${NUM}
```

-s - Supress output

```
echo "Enter password: "
read -s PASSW
echo "${PASSW}"
```

A READ EXAMPLE

```
echo "Enter your new password: "
read -s PASSWD1
echo "Enter your password again: "
read -s PASSWD2
if [[ ${PASSWD1} == ${PASSWD2} ]]
then
  PASSWD=$PASSWD
  echo "Your password has been saved"
else
  echo "The two passwords do not match, start over..."
fi
```

CONDITIONAL HANDLING

- if statement
- if else
- if elif
- case

IF STATEMENT

- Uses a test with a binary output to determine path to take
- Test operators - http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html
- Arithmetic operators - -eq, -ne, -lt, -le, -gt or -ge

IF STATEMENT SYNTAX

```
if [[ test condition ]]
then
  Do lots of stuff
fi
```

EXAMPLE OF USE

```
grep Hello *
ANSW=$?
if [[ ${ANSW} -lt 1 ]]
then
  echo "The command worked"
fi
```

IF ELSE STATEMENT

- Uses a test with a binary output to determine path to take
- Provides what to do if the condition is false
- Syntax

```
if [[ test condition ]]
then
Do lots of stuff
else
Do different stuff
fi
```

EXAMPLE OF USE

```bash
grep Hello *
ANSW=$?
if [[ ${ANSW} -lt 1 ]]
then
  echo "The command worked"
else
  echo "I can't find Hello anywhere"
fi
```

IF ELIF STATEMENT

- Uses a test with a binary output to determine path to take
- Uses further tests to provide different paths
- Syntax

```
if [[ test condition ]]
then
Do lots of stuff
elif [[ Second test condition ]]
Do different stuff
else
Do different stuff
fi
```

EXAMPLE OF USE

```
ANSW=6
if [[ ${ANSW} -lt 1 ]]
then
  echo "The answer is less than 1"
elif [[ ${ANSW} -lt 5 ]]
  echo "The answer is less than 5"
else
  echo "The answer is less than 10"
fi
```

CASE STATEMENT

- Gives several different paths depending on the results of a single condition

```
case <expression> in
case1)
 break
 ;;
case2)
 break
 ;;
*)
 break
 ;;
esac
```

EXAMPLE OF USE

```
disk=`df -h | grep -v snap | awk '{print $5}' | grep % | grep
case ${disk} in
[1-6]*)
  echo="No disk issues here, move along"
  break
  ;;
[7-8]*)
  echo="There is a partition with ${disk}% space. Check it out
  break
  ;;
9[0-8])
  echo="Seriously...  One partition is ${disk}% full."
  break
  ;;
99)
```

SELECT CASE TO CREATE MENUS

- Allows menus in your script

```
select <variable> in <list>
do
  case $<variable in
    case1)
      break
      ;;
    case2)
      break
      ;;
    *)
      break
      ;;
  esac
done
```

EXAMPLE

```
select NUM in $(cat file1) Exit
do
case ${NUM} in
  Exit)
    exit
    ;;
  *)
    echo $NUM
    ;;
esac
done
```

LOOPS

- for loop
- while loop
- until loop

FOR

- Iterates through a list
- Syntax

```
for i in <list>
do
All the things
done
```

FOR LOOP EXAMPLE

```
for i in {1..5}
do
  echo ${i}
done
```

WHILE

- Performs actions while a condition is true

```
while [[ condtion ]]
do
Do stuff
done
```

EXAMPLE

```
x=0
while [[ ${x} -lt 10 ]]
do
  echo ${x}
  let x=${x}+1
  sleep 1
done
```

UNTIL

-Performs actions until a condition is true

```
while [[ condtion ]]
do
  Do stuff
done
```

EXAMPLE

```bash
x=0
until [[ ${x} -ge 10 ]]
do
  echo ${x}
  let x=${x}+1
  sleep 1
done
```

FUNCTIONS

- Functions are reusable bits of code
- They cut down on file size by reducing repetition
- Must be defined in a script before they are called
    - Commonly all functions are defined at the very beginning of a script

SYNTAX

```
function <name> {
  Do all the things to $1
}

<name>
<name> foobar
```

ANOTHER USE FOR FUNCTIONS

- Often times, if you have a series of commands you run often you can create a function file that gets called by .bashrc. This would be like alias command on steroids.

TOPICS

- Why script?
- She-Bang
- Variables
- Special variables
- User input

TOPICS (CONT)

- Conditional handling
- Select/case text menu
- Loops
- Functions

THANKS.

REMEMBER THE NEW VENUE NEXT MONTH

QUESTIONS