UDEMY : Bash scripting and Shell Programming

Section : Shell scripting in a nutshell

Shell Scripting : Part One

Scripts :
- contain a series of commands
- an interpreter executes commands in the script
- anything you can type in command line, you can put in a script
- great for automating tasks

Eg of shell script using different programs as interpreter

```
#! /bin/csh
  echo "This script runs csh"

#! /bin/ksh
  echo "This script runs ksh"

#! /bin/zsh
  echo "This script runs zsh"
```

Shebang or Not to Shebang :
- if a script does not use a shebang it command can execute using your shell
- different shells have slightly varying syntax

Variable :
- Name-value pairs
- Syntax : VARIABLE_NAME : "Value"
- case sensitive
- by convention they are uppercase

Eg: #!/bin/bash

```
MY_SHELL = "bash"
echo "I like $MY_SHELL shell."
```

or

```
echo "I like ${MY_SHELL} shell."
```

- can also assign command output to a variable

Eg: #!/bin/bash

```
SERVER_NAME = $(hostname)
echo "You are running the script
on ${SERVER_NAME}."
```

Tests:

Syntax -> [condition-to-test-for]

Eg: [-e /etc/passwd]

File Operators:

-d	file	True if file is a directory
-e	file	True if file exists
-f	file	True if file exists and is a regular file
-r	file	True if file is readable by you
-s	file	True if file exists and is not empty
-w	file	True if file is writable by you
-x	file	True if file is executable by you

String Operators:

-z	string	True if string is empty
-n	string	True if string is not empty
STRING1 = STRING2		True if strings equal
STRING1 != STRING2		True if strings not equal

Arithmetic Operators :

arg1 -eq arg2	True if arg1 equal arg2
arg1 -ne arg2	True if arg1 not equal arg2
arg1 -lt arg2	True if arg1 less than arg2
arg1 -le arg2	True if arg1 less than or equal arg2
arg1 -gt arg2	True if arg1 greater than arg2
arg1 -ge arg2	True if arg1 greater than or equal arg2

Making Decisions - The if statement

```
if [ condition - "-" true ]
then
        command 1
        command 2
        command n
fi
```

Eg

```
#! /bin/bash
MY_SHELL = "bash"

if [ "$MY_SHELL" : "bash" ]
then
        echo "You like bash shell."
fi
```

if / else :

```
if [ condition - " " - true ]
then
        command N
else
        command N
fi
```

if / elif / else

```
if [ condition - " " - true ]
then
    command N
elif    [ condition - " " - true ]
  then
        command N
    else
        command N
    fi
```

for loop :

```
for    VARIABLE_NAME  in  ITEM_1   ITEM_N
do
        command 1
        command 2
        command n
done
```

Eg : `#! /bin/bash`

```
COLOR1 = " red green blue "

for    COLOR    in    $COLOR1
do
        echo    " COLOR : $ COLOR "
done
```

Eg: Rename all file containing jpg and adding # date + #

```
#!/bin/bash
PICTURES = $(ls *.jpg)
DATE = $(date +%y-%m-%d)

for PICTURE in $PICTURES
do
    echo "Renaming ${PICTURE} to ${DATE}-${PICTURE}"
    mv ${PICTURE} ${DATE}-${PICTURE}
done
```

Positional Parameters :

```
$ script.sh parameter1 parameter2 parameter3

$0 : "script.sh"
$1 : "parameter1"
$2 : "parameter2"
$3 : "parameter3"
```

Eg:

```
#! /bin/bash
USER=$1 # first parameter to the user

echo "Executing script : $0"
echo "Adding user : $USER"

# Lock the account
passwd -l $USER

# Create an archive of the home directory
tar -cf /archive/${USER}.tar.gz /home/${USER}
```

Eg: How to update script to accept one or more parameters

```
#!/bin/bash

echo "Executing script : $0"
for USER in $@
do
    echo "Working User : $USER"
    # Call the account
    passwd -l $USER
    # Create an archive of the home directory
    tar cf /archive/${USER}.tar.gz /home/${USER}
done
```

Accepting User Input (STDIN)

- The read command accepts STDIN

Syntax:

```
read -p "PROMPT" VARIABLE
```

Section : Return Codes and Exit Statuses

Exit Status / Return Code :
- every command return an exit status
- range from 0 to 255
- 0 = success
- use for error checking
- use man or info to find meaning of exit status

Checking Exit Status :
- $? contains the return code of previous executed command

Eg
```
ls /not/here
echo "$?"
```

Eg : Checking if host is reachable using ping :
```
HOST="google.com"
ping -c 1 $HOST
if [ "$?" -eq "0" ]
then
    echo "$HOST reachable"
else
    echo "$HOST unreachable."
fi
```

&& and || :

&& → second command will only run if first succeed,
|| → second command will only run if first fail

Exit Command :
- explicitly define return code of your script :
  - exit 0
  - exit 2
- default is that of last command executed

Eg:

```
#!/bin/bash
HOST="google.com"
ping -c 1 $HOST
if [ "$?" -ne "0" ]
then
    echo "$HOST unreachable."
    exit 1
fi
exit 0
```

Section: Shell functions

Creating a function:

```
function function_name () {
    # code goes here
}

function_name () {
    # code goes here
}
```

Calling a function:

```
function hello() {
    echo "Hello"
}

hello
```

- function can call other function

Positional Parameters:

- functions can accept parameters
- first parameter stored in $1
- $@ contains all # parameters

- Variables are global but must be defined before used

Local Variables:
- only be accessed with in funcs
- use local keyword:
  local LOCAL_VAR=1

Exit Codes:
- Functions have an exit status
- explicitly: return <RETURN_CODE>
- implicitly: last exit status of last command

Section: Shell script order of checklist

1. Shebang
2. Comments / file header
3. Global Variables
4. Functions
   (a) Use local variables
5. Main script contents
6. Exit with an exit status
   (a) exit <STATUS> at various exit points

Section: Wildcards

Wildcards:
- A character or string used for pattern matching.
- Globbing expand. the wildcard pattern into a list of file and/or directories (paths)
- Wildcard. can be used with most commands:
  - ls
  - rm
  - cp

- * - match zero or more characters
- ? - matches exactly one character

- [] - A character class.
  - Matches any of the characters inclosed between the brackets.
    Matches exactly one character
  - To match - one character long filenames that a vowel: [aeiou]
  - Eg: Ca [nt]*
    matches: can, cat, candy, catch

- ! - Matches any of the characters NOT inclosed between brackets.
  Eg: files that doesn't start with a vowel: [!aeiou]*

Ranges:
- use two characters separated by a hyphen to create a range in a character class
  Eg: [a-z]* -> matches all files start with a,b,c,d... - z)

Predefined named character classes:
- [[:alpha:]]
- [[:alnum:]]
- [[:digit:]]
- [[:lower:]]
- [[:space:]]
- [[:upper:]]

Matching Wildcard Pattern:
- \ - escape character. Use it when you want to match a wildcard character
  Eg: *\?. -> match files ending with question marks

Section: Case statement and logger

Eg: case "$VAR" in
pattern_1)
commd. go here
;;
pattern N)
commd. go here
;;
esac

Section: Logging

Logging:
- log who, what, when, where and why
- output may scroll off the screen
- script may run unattended

Syslog:
- syslog the standard used facilities and severities to categorize msgs
- log file location are configured:
  - /var/log/ messages
  - /var/log/ syslog

Logging with logger:
- # logger utility
- by default create user.notice msgs

Eg: logger "Msg"
logger -p local0.info "Msg"
logger -t myscript -p local0.info "Msg"
logger -i -t myscript "Messge"

Section: while loop

Format :

```
while [ condition_is_true ]
do
        command 1
        command 2
        command N
done
```

Eg :

```
INDEX=1
while [ $INDEX -lt 6 ]
do
    echo "Creating project - ${INDEX}"
    mkdir /var/tmp/project-${INDEX}
    ((INDEX++))
done
```

Reading a file, line-by-line

```
LINE_NUM=1
while read LINE
do
    echo "${LINE_NUM}: ${LINE}"
    ((LINE_NUM++))
done < /etc/fstab
```

Section: Debugging your bash

Built in Debugging help:

- -x : print command as they execute
  - #!/bin/bash -x
  - set +x to stop debugging
- -e : Exit on error
  Can be combined with other options
  - #!/bin/bash -ex
- -v : print shell input lines as they are read

Manual Debugging
- use specific variable like DEBUG
  DEBUG = true
  DEBUG = false

Eg: #!/bin/bash
DEBUG = "echo"
$ DEBUG ls

PS4:

Eg: #!/bin/bash -x
PS4 = '+ $(date +%s.%N) ${LINENO}: '
TEST_VAR = "test"
echo "$TEST_VAR"

output: + PS4='+ $(date +%s.%N) ${LINENO}: '
+ ./test.sh: 3: TEST_VAR=test
+ ./test.sh: 4: echo test
test

Section: Data Manipulation and Text Transformation with sed

Sed and Streams

- Sed = Stream editor
- Sed performs text transformations on streams
- Sed works programmatically, not interactively

Eg: sed 's/orginal/changed' filename.txt

- Does not alter orginal file