Shell Scripting I: Basics & Automation

Session 6

Objectives:

- Know what a shell script is and how to write one
- Understand the use of the shebang (#!) line and permissions
- Be able to use variables, read, and echo
- Write conditional structures like if, else, and case
- Prepare for automation through scripting

What is a shell script?

- A script is a plain text file containing commands designed to be run by Linux
- Executed sequentially by the shell
- Like a to-do list for your terminal!

shell scripts include file manipulation, program execution, and printing text.

Example:

```
#!/bin/
#print Hello, word
echo "Hello, world!«
# show the date
date
```

Creating and running a shell script

Shell scripts have several required constructs that tell the shell environment what to do and when to do it.

shebang →

```
#!/bin/bash
#print Hello, word
echo ‘’Hello, world!’’
# show the date
date
```

#! shebang

The seemingly insignificant #! characters at the beginning of a shell script has a major significance on how your script will be executed.

The shebang is the combination of **the # (pound key) and ! (exclamation mark).** It is used to specify the interpreter with which the given script will be run by default

e.g.:
#!/bin/bash -> interperter should be bash shell
#!/bin/zsh -> interperter should be Z shell

So it is important to precise the shell interperter in the script as different shell have different synthax

Creating and running a shell script

Shell scripts have several required constructs that tell the shell environment what to do and when to do it.

#

```
#!/bin/
#print Hello, word
echo ‘’Hello, world!’’
# show the date
date
```

All the lines starting by **#** (except the shebang) are **comments,** meaning that they are ignored

However, they are very important to give explanations, notes for the collaborators

Creating and running a shell script

Shell scripts have several required constructs that tell the shell environment what to do and when to do it.

command →

```
#!/bin/
#print Hello, word
echo ‘’Hello, world!’’
# show the date
date
```

Then you can write as many command lines as you want. Here, the script will print « Hello, world!»

Creating and running a shell script

- Create a file with .sh extension:

-> gedit hello.sh

- Add commands with a shebang:

```
#!/bin/bash
#print Hello, word
echo ‘’Hello, world!’’
# show the date
date
```

- Save and exit
- Change the permission (session 9)

-> chmod +x hello.sh

- Run it

-> ./hello.sh, you can also run bash hello.sh and it will ignore the shebang

Variables

- Syntax:

```
name="Thomas“
echo "Welcome $name“
```

- NO SPACE BEFORE/AFTER =

- Refer with **$**

```
#!/bin/sh
MY_MESSAGE="Hello World"
echo $MY_MESSAGE
```

- echo “The number of line is: $(wc –l errors.txt)”

Reading input from user

- Syntax:

```
#!/bin/bash
echo “What is your name?”
read MY_NAME
echo "Hello $MY_NAME - hope you're well."
```

- **Read** stores input into a variable

- Use **export** if the variable needs to be available in sub-shells or other script

```
export MY_NAME
```

Reading input from user

- Example myvar.sh:

```
#!/bin/sh
echo "MYVAR is: $MYVAR"
MYVAR="hi there"
echo “MYVAR is: $MYVAR”
```

What is the output, it you run ./myvar.sh ?

Reading input from user

- Example myvar.sh:

```
#!/bin/sh
echo "MYVAR is: $MYVAR"
MYVAR="hi there"
echo “MYVAR is: $MYVAR”
```

What is the output, it you run ./myvar.sh ?

```
MYVAR is:
MYVAR is: hi there
```

Reading input from user

- Now try:

```
$ MYVAR=“hello”
$ ./myvar.sh
```

Reading input from user

- Now try:

```
$ MYVAR=“hello”
$ ./myvar.sh
```

Output:

```
MYVAR is:
MYVAR is: hi there
```

It is the same, because when you run ./myvar.sh, a new shell is used

You can use export:

```
$ MYVAR=“hello”
$ export MYVAR
$ ./myvar.sh
```

Reading input from user

You can use export:

```
$ MYVAR=hello
$ export myvar
$ ./myvar.sh
```

Output:

```
          MYVAR is: hello
          MYVAR is: hi there
```

If you do: $ echo MYVAR you will see hello

Once the shell script exits, its environment is destroyed. But MYVAR keeps its value of hello within your interactive shell.

Reading input from user

```
#!/bin/sh
echo "What is your name?"
read USER_NAME
echo "Hello $USER_NAME"
echo "I will create you a file called $USER_NAME.txt"
touch $USER_NAME.txt
```

What will happen?

Reading input from user

What can we do?

Curly brackets

What can we do? **CURLY brackets**

- Curly braces {} help clarify variable boundaries
- Prevent confusion when variables are next to other characters

```
#!/bin/bash
echo "What is your name?"
read USER_NAME
echo "Hello $USER_NAME"
echo "I will create you a file called ${USER_NAME}.txt"
touch "${USER_NAME}.txt"
```

Conditions if/else

Structure:

```
if [ … ];then
  # if-code
else
  # else-code
fi
```

Common numeric test operators:

-eq (equal)
-ne (not equal)
-gt (greater than)
-lt (less than)
-ge (greater or equal)
-le (less or equal)

Common file test operators:

-f file → is a file
-d dir → is a directory
-x file → is executable
-r file → is readable
-w file → is writable

Examples

Check if a myfile.txt exists:

Examples

Check if a file exists:

```
if [ -f myfile.txt ]; then
  echo "File exists."
else
  echo "File not found."
fi
```

Examples

Check if a file exists:

```
#!/bin/bash
if [ -f myfile.txt ]; then
  echo "File exists."
else
  echo "File not found."
fi
```

Compare two numbers (a,b):

Examples

Check if a file exists:

```
#!/bin/bash
if [ -f myfile.txt ]; then
  echo "File exists."
else
  echo "File not found."
fi
```

Compare two numbers:

```
#!/bin/bash
echo "Enter two numbers:"
read a b
if [ $a -eq $b ]; then
  echo "Both are equal"
elif [ $a -gt $b ]; then
  echo "$a is greater than $b"
else
  echo "$b is greater than $a"
fi
```

Examples

Check if a file exists:

```
#!/bin/bash
if [ -f myfile.txt ]; then
  echo "File exists."
else
  echo "File not found."
fi
```

Compare two numbers:

```
#!/bin/bash
echo "Enter two numbers:"
read a b
if [ $a -eq $b ]; then
  echo "Both are equal"
elif [ $a -gt $b ]; then
  echo "$a is greater than $b"
else
  echo "$b is greater than $a"
fi
```

Ask for username and check if it is root:

Examples

Check if a file exists:

```
#!/bin/bash
if [ -f myfile.txt ]; then
  echo "File exists."
else
  echo "File not found."
fi
```

Compare two numbers:

```
#!/bin/bash

echo "Enter two numbers:"
read a b
if [ $a -eq $b ]; then
  echo "Both are equal"
elif [ $a -gt $b ]; then
  echo "$a is greater than $b"
else
  echo "$b is greater than $a"
fi
```

Ask for username and check if it is root:

```
#!/bin/bash
echo "Enter your username:"
read user
if [ "$user" == "root" ]; then
  echo "You are the administrator."
else
  echo "You are a regular user."
fi
```

Case structure

The case structure is perfect when you have many conditions to evaluate. Instead of chaining multiple if and elif, you define patterns in a clean and readable block.

- Start with case
- Patterns use | for alternates
- Every block ends with ;;
- esac closes the case lock

Case structure

Exemples.

- Give a number between 1 and 3
- Print “your number is one” if it is 1, etc….

```
echo "Enter a number (1-3):"
read choice
if [ "$choice" -eq 1 ]; then
  echo "You chose one."
elif [ "$choice" -eq 2 ]; then
  echo "You chose two."
elif [ "$choice" -eq 3 ]; then
  echo "You chose three."
else
  echo "Invalid choice."
fi
```

Case structure

Exemples.

- Give a number between 1 and 3
- Print “your number is one” if it is 1, etc….

```
echo "Enter a number (1-3):"
read choice
if [ "$choice" -eq 1 ]; then
  echo "You chose one."
elif [ "$choice" -eq 2 ]; then
  echo "You chose two."
elif [ "$choice" -eq 3 ]; then
  echo "You chose three."
else
  echo "Invalid choice."
fi
```

`if` is used for conditional logic; `case` is cleaner for multiple value comparisons. Use `case` when comparing a single variable to many values

```
echo "Enter a number from 1 to 3:"
read choice

case $choice in
  1) echo "You chose one.";;
  2) echo "You chose two.";;
  3) echo "You chose three.";;
  *) echo "Invalid choice.";;
esac
```

What you learn:

You now know:

- How to write and run scripts
- Use variables and input
- Create decisions using if and case
- Apply comments and use the #! line:

Let’s practice!!

Labs

Lab 1: Swapping Two Variables (10 min)

- Create a script swap.sh:
- Ask for 2 values (A and B)
- Swap and display their new values

Labs

Lab 1: Swapping Two Variables (10 min)

```
#!/bin/bash
echo -n "Enter value for A: "
read a
echo -n "Enter value for B: "
read b
t=$a
a=$b
b=$t
echo "Values after Swapping:"
echo "A = $a, B = $b"
```

Labs

Lab 2: Fahrenheit to Celsius (10 min)

- Create degconv.sh:
- Ask user for Fahrenheit temperature
- Convert to Celsius using formula
- Show the result

Labs

Lab 2: Fahrenheit to Celsius (10 min)

```
#!/bin/bash
echo -n "Enter Fahrenheit: "
read f
c=$(( (f - 32) * 5 / 9 ))
echo "Centigrade is: $c"
```

Labs

Lab 3: Biggest of Three Numbers (15 min)

- Script big3.sh:
- Ask user for three numbers
- Use if and -gt to find the biggest
- Display which is biggest

Labs

Lab 3: Biggest of Three Numbers (15 min)

```
#!/bin/bash
echo -n "Enter values for A B and C: "
read a b c
if [ $a -gt $b ] && [ $a -gt $c ]; then
  echo "A is the biggest"
elif [ $b -gt $c ]; then
  echo "B is the biggest"
else
  echo "C is the biggest"
fi
```

Labs

Lab 4: Grade Determination (15 min)

- Script grade.sh:
- Ask for a mark (0–100)
- Output a grade using if/elif logic
- S: >90, A: >80,B:>80,C:>70,D:>60,E>50, U: <50

Labs

Lab 4: Grade Determination (15 min)

```
#!/bin/bash
echo -n "Enter your mark: "
read mark
if [ $mark -gt 90 ]; then
  echo "S Grade"
elif [ $mark -gt 80 ]; then
  echo "A Grade"
elif [ $mark -gt 70 ]; then
  echo "B Grade"
elif [ $mark -gt 60 ]; then
  echo "C Grade"
elif [ $mark -gt 55 ]; then
  echo "D Grade"
elif [ $mark -ge 50 ]; then
  echo "E Grade"
else
  echo "U Grade"
fi
```

Labs

Lab 5: Vowel or Consonant (10 min)

- Script vowel.sh:
- Read a single lowercase character
- Use case to determine if it’s a vowel

Labs

Lab 5: Vowel or Consonant (10 min)

```
#!/bin/bash
echo -n "Enter a lowercase character: "
read choice
case $choice in
a|e|i|o|u)
  echo "It's a vowel";;
*)
  echo "It's a consonant";;
esac
```

Labs

Lab 6: Mini Calculator (20 min)

- Script calc.sh:
- Ask for two numbers
- Show menu: add, subtract, multiply, divide
- Use case and expr to display result

Labs

Lab 6: Mini Calculator (20 min)

```
#!/bin/bash
echo -n "Enter two numbers: "
read a b
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
echo -n "Choose an option: "
read op
case $op in
  1) echo "$a + $b = $((a + b))";;
  2) echo "$a - $b = $((a - b))";;
  3) echo "$a * $b = $((a * b))";;
  4) echo "$a / $b = $((a / b))";;
  *) echo "Invalid option";;
esac
```

Labs

Lab 7: Check lines and words (20 min)

Write a shell script which counts the number of lines and number of words present in a given file.

Use echo, read, wc to do:

- Ask the user for the file name
- Check if the file exist and count number of lines and words

Labs

Lab 7: Check lines and words (20 min)

```
#!/bin/bash

echo "Enter the file name:"
read filename

# Check if the file exists
if [ -f "$filename" ]; then
  echo "Counting lines and words in '$filename'…"
  echo "Number of lines: $(wc -l < "$filename")"
  echo "Number of words: $(wc -w < "$filename")"
else
  echo "File '$filename' not found."
fi
```