Hands-on Lab: Advanced Bash Scripting

Estimated time needed: **30** minutes

Welcome to this hands-on lab, where you will take your Bash scripting chops to the next level. The skills you practice here will serve as logical building blocks for an endless variety of scripting applications. These concepts will also be essential for demonstrating your new skills in the Final Project for this course.

As you develop your Bash scripts, it's always recommended that you test your results at each stage to ensure your logic is behaving as expected. Think of each stage as a building block of your final script that accomplishes an easily digestible sub-task.

Learning Objectives

After completing this lab, you will be able to:

- Run sets of commands using conditional statements
- Create `true`/`false` comparisons with logical operators
- Use arithmetic operators to perform basic mathematical calculations
- Use list-like arrays to store and access data
- Execute repetitive tasks with `for` loops

About Skills Network Cloud IDE

Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands-on labs for course and project related labs. Theia is an open-source IDE (Integrated Development Environment) that can be run on a desktop or on the cloud. To complete this lab, we will be using the Cloud IDE based on Theia running in a Docker container.

Important notice about this lab environment

Please be aware that sessions for this lab environment are not persisted. Every time you connect to this lab, a new environment is created for you. Any data you may have saved in the earlier session would get lost. Plan to complete these labs in a single session to avoid losing your data.

Exercise 1 - Using conditional statements and logical operators

In this exercise, you will create a simple Bash script containing a conditional statement to handle the following tasks:

- Prompt the user for a `Yes` or `No` response to a question
- Print a response based on the user's answer

1.1. Create a new Bash script

Create a Bash script file and make it executable.

▼ Click here for Hint

Use the `echo` command to redirect a **shebang** to a new Bash script.

Alternatively, open a new text file using your favourite text editor and add a **shebang** to it. Remember to make your new script executable.

▼ Click here for Solution
Here's a solution using only the command line:

```
1. 1
2. 2
```

```
1. echo '#!/bin/bash' > conditional_script.sh
2. chmod u+x conditional_script.sh
```

Copied! Executed!

1.2. Query the user and store their response

Now get your script to:

1. Ask the user a binary "yes or no" question of your choosing
2. Store the user's answer in a variable.

▼ Click here for Hint

Use the `echo` and `read` commands.

▼ Click here for Solution

Your Bash script should now look something like this:

```
1
2
3
4
5
```

```
#!/bin/bash

echo 'Are you enjoying this course so far?'
echo -n "Enter \"y\" for yes, \"n\" for no."
read response
```

Copied! Executed!

1.3. Use a conditional block to select a response for the user

Finally, use a conditional block to print a message to the user based on their response to your query.

Tip: It's best practice to also handle the case where the reponse doesn't match any of the allowable responses.

▼ Click here for Hint

Use a conditional `if elif else fi` block that uses a logical operator to compare the user's response to the available response options and prints an appropriate message in each case.

▼ Click here for Solution

Now your Bash script should be similar to the following:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
#!/bin/bash

echo 'Are you enjoying this course so far?'
echo -n "Enter \"y\" for yes, \"n\" for no"
read response
if [ "$response" == "y" ]
then
    echo "I'm pleased to hear you are enjoying the course!"
    echo "Your feedback regarding what you have been enjoying would be most welcome!"
elif [ "$response" = "n" ]
then
   echo "I'm sorry to hear you are not enjoying the course."
   echo "Your feedback regarding what we can do to improve the learning experience"
   echo "for this course would be greatly appreciated!"
else
   echo "Your response must be either 'y' or 'n'."
   echo "Please re-run the script to try again."
fi
```

Copied!

Exercise 2 - Performing basic mathematical calculations and numerical logical comparisons

In this exercise, you will create a Bash script that performs basic arithmetic calculations on two integers entered by the user. You will also use logical comparisons to determine which calculation leads to the greatest result.

2.1. Create a Bash script

Create an executable Bash script that prompts the user for two integers, then stores and prints both the sum and product of the two integers.

▼ Click here for Hint

Use the `echo` and `read` commands as in the previous exercise.

Recall the notation for arithmetic calculations.

▼ Click here for Solution

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
#!/bin/bash

echo -n "Enter an integer: "
read n1
echo -n "Enter another integer: "
read n2

sum=$(($n1+$n2))
product=$(($n1*$n2))

echo "The sum of $n1 and $n2 is $sum"
echo "The product of $n1 and $n2 is $product."
```

Copied!

2.2. Add logic to your script

Add logic to your script that determines whether the sum is greater than, less than, or equal to the product. Display an appropriate statement corresponding to each possible result.

Assume the user inputs two integers. Don't worry about handling the case where the user inputs a non-integer string by mistake.

▼ Click here for Hint

Use a conditional block. Recall the notation for logical operators.

▼ Click here for Solution

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

```
#!/bin/bash

echo -n "Enter an integer: "
read n1
echo -n "Enter another integer: "
read n2

sum=$(($n1+$n2))
product=$(($n1*$n2))

echo "The sum of $n1 and $n2 is $sum"
echo "The product of $n1 and $n2 is $product."

if [ $sum -lt $product ]
then
   echo "The sum is less than the product."
elif [[ $sum == $product ]]
then
   echo "The sum is equal to the product."
elif [ $sum -gt $product ]
then
   echo "The sum is greater than the product."
```

```
23. fi
```

Copied!

Exercise 3 - Using arrays for storing and accessing data within *for* loops

In this exercise, you will create a report based on a supplied dataset using the CSV format. You will extract the columns of the dataset into separate arrays and create a new column using arithmetic and array logic. Finally, you will combine the dataset with the new column and save the resulting report as a CSV file.

3.1. Download a CSV file to your current working directory

The file, `arrays_table.csv`, is located at the following url:

```
1. 1
```

```
1. https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-LX0117EN-SkillsNetwork/labs/M3/L2/arrays_table.csv
```

Copied!

▼ Click here for Hint

Use the `wget` command.

▼ Click here for Solution

```
1. 1
2. 2
```

```
1. csv_file="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-LX0117EN-SkillsNetwork/labs/M3/L2/arrays_table.csv"
2. wget $csv_file
```

Copied!

3.2. Display the CSV file to understand what it looks like

▼ Click here for Hint

Use `cat` at the command prompt or open the file using the GUI.

▼ Click here for Solution

```
1. 1
```

```
1. cat arrays_table.csv
```

Copied!

3.3. Create a Bash script that parses table columns into 3 arrays

▼ Click here for Hint

Use command substitution, the `cut` command, and the notation for creating an array from a list of elements.

Also, print the first array to validate your logic.

▼ Click here for Solution

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
```

```
1. #!/bin/bash
2.
3. csv_file="./arrays_table.csv"
4.
5. # parse table columns into 3 arrays
6. column_0=($(cut -d "," -f 1 $csv_file))
7. column_1=($(cut -d "," -f 2 $csv_file))
8. column_2=($(cut -d "," -f 3 $csv_file))
9.
10. # print first array
11. echo "Displaying the first column:"
12. echo "${column_0[@]}"
```

Copied!

3.4. Create a new array as the difference of the third and second columns.

Initialize your new array with a header (a column name), and remember to validate your results.

▼ Click here for Hint 1

Use a loop to populate your array.

Determine the number of elements you need and incorporate within your loop statement.

Print both the number of elements and the contents of your new array to validate your logic.

▼ Click here for Hint 2

Recall the `for` loop notation when you know the number of iterations needed, and that array indexing starts at 0.

To get the number of lines, use command subsitution on a pipeline that uses the `cat` and `wc` commands as filters and store the result in a variable.

▼ Click here for Solution

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

```
#!/bin/bash

csv_file="./arrays_table.csv"

# parse table columns into 3 arrays
column_0=($(cut -d "," -f 1 $csv_file))
column_1=($(cut -d "," -f 2 $csv_file))
column_2=($(cut -d "," -f 3 $csv_file))

# print first array
echo "Displaying the first column:"
echo "${column_0[@]}"

## Create a new array as the difference of columns 1 and 2
# initialize array with header
column_3=("column_3")
# get the number of lines in each column
nlines=$(cat $csv_file | wc -l)
echo "There are $nlines lines in the file"
# populate the array
for ((i=1; i<$nlines; i++)); do
  column_3[$i]=$((column_2[$i] - column_1[$i]))
done
echo "${column_3[@]}"
```

Copied!

3.5. Create a report by combining your new column with the source table

Save your report as a CSV file.

Remember to validate your results.

▼ Click here for Hint 1

Write the new array to file line-by-line.

Intitialize the file with a header.

▼ Click here for Hint 2

Use redirection and recall how to merge two files side-by-side.

Ensure your final report has the correct CSV format.

▼ Click here for Solution

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

```
#!/bin/bash

csv_file="./arrays_table.csv"

# parse table columns into 3 arrays
column_0=($(cut -d "," -f 1 $csv_file))
column_1=($(cut -d "," -f 2 $csv_file))
column_2=($(cut -d "," -f 3 $csv_file))

# print first array
echo "Displaying the first column:"
echo "${column_0[@]}"

## Create a new array as the difference of columns 1 and 2
# initialize array with header
column_3=("column_3")
# get the number of lines in each column
nlines=$(cat $csv_file | wc -l)
echo "There are $nlines lines in the file"
# populate the array
for ((i=1; i<$nlines; i++)); do
  column_3[$i]=$((column_2[$i] - column_1[$i]))
done
echo "${column_3[@]}"

## Combine the new array with the csv file
# first write the new array to file
# initialize the file with a header
echo "${column_3[0]}" > column_3.txt
for ((i=1; i<nlines; i++)); do
  echo "${column_3[$i]}" >> column_3.txt
done
paste -d "," $csv_file column_3.txt > report.csv
```

Copied!

Conclusion

Congratulations! You have just completed a hands-on lab using advanced Bash scripting logic.

In this lab, you learned that you can use:

- Conditional statements to run commands based on whether a specified condition is `true` or `false`
- Logical operators to make `true`/`false` operators
- Arithmetic operators to perform basic mathematical calculations
- List-like arrays to store and access data
- `for` loops to execute repetitive tasks

You covered a lot of material here that will be very useful going forward. You will encounter similar problems in your practice and final projects and, best of all, in your career! Remember - you can always come back and review your labs.

> **Tip**: It is worth noting that had we been only interested in efficiency, one of the steps could have been avoided. Specifically, you could have redirected your calculations to a text file line-by-line rather than storing them in an array and then writing the array to file.

Finally, we encourage you to post a review and a rating for this course at any time!

Author

Jeff Grossman

Other Contributors

Rav Ahuja