



# **NOTRE DAME UNIVERSITY**

## **BANGLADESH**

### **CSE-3206 Lab Report-02**

**Course Title: Operating System Lab**

**Course Code: CSE-3206**

**Submitted by:**

**Name: Istiak Alam**

**ID: 0692230005101005**

**Batch: CSE-20**

**Submission Date: 06-02-25**

**Lab Task Topic : Familiarization Shell Programming.**

**Submitted to:**

**Khorshed Alam**

**Lecturer, NDUB**

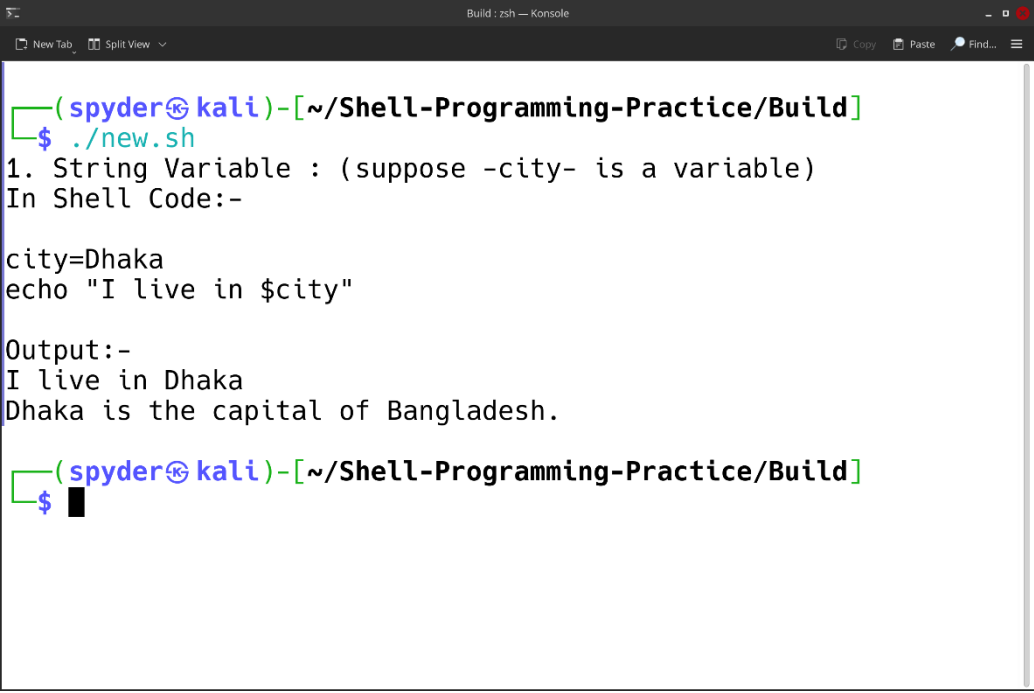
## ❑ Shell Programming :

### 2.1 Variables :

#### Code :

```
#!/bin/bash
echo "1. String Variable : (suppose -city- is a variable)"
printf "In Shell Code:-\n\n"
echo "city=Dhaka"
echo 'echo "I live in $city"'
echo " "
echo "Output:-"
city="Dhaka"
echo "I live in $city"
#SAME AS-
echo "${city} is the capital of Bangladesh."
```

#### Output :



```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
1. String Variable : (suppose -city- is a variable)
In Shell Code:-

city=Dhaka
echo "I live in $city"

Output:-
I live in Dhaka
Dhaka is the capital of Bangladesh.

(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ █
```

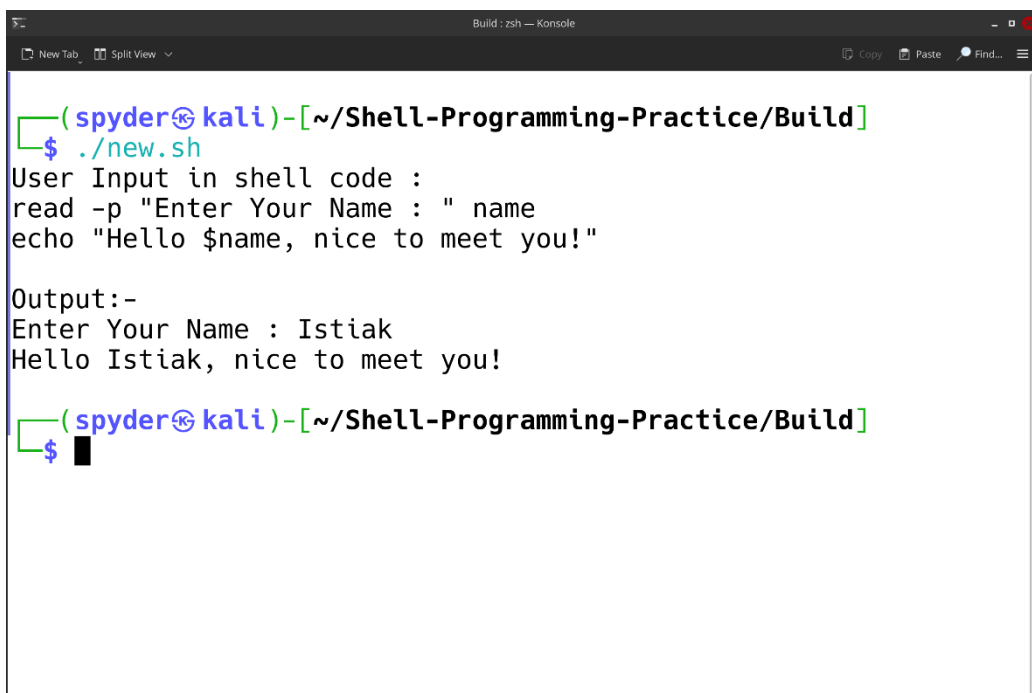
Fig 2.1 : Variable

## 2.2 User input:

### Code :

```
#!/bin/bash
echo "User Input in shell code :"
echo 'read -p "Enter Your Name : " name'
echo 'echo "Hello $name, nice to meet you!"'
echo ""
echo "Output:-"
read -p "Enter Your Name : " name
echo "Hello $name, nice to meet you!"
```

### Output :



```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
User Input in shell code :
read -p "Enter Your Name : " name
echo "Hello $name, nice to meet you!"

Output:-
Enter Your Name : Istiak
Hello Istiak, nice to meet you!

(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

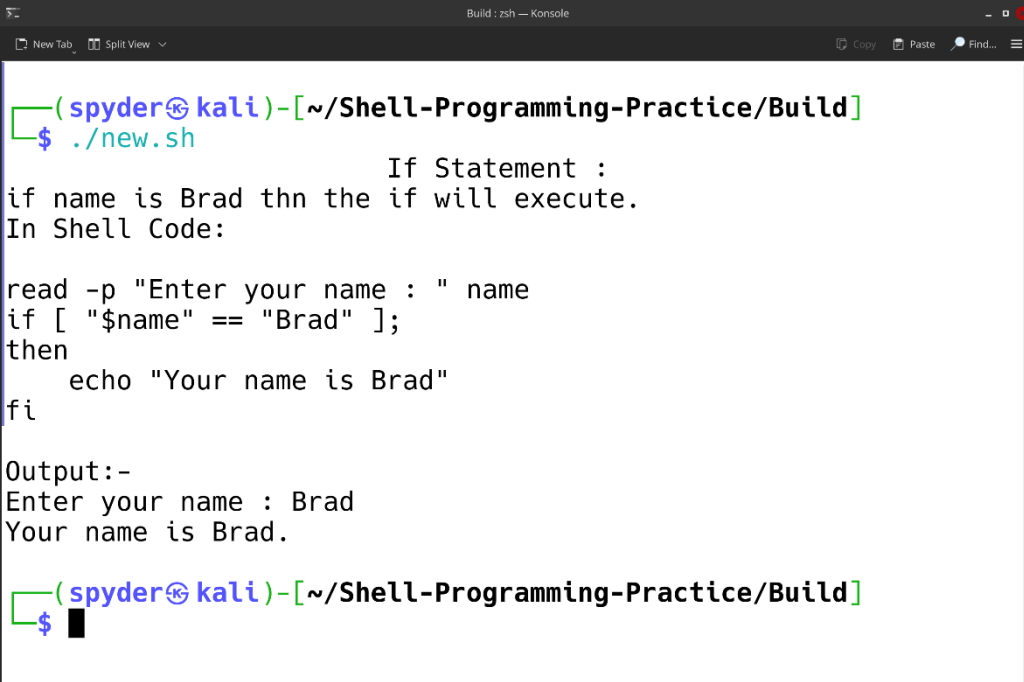
Fig 2.2 : User Input

## 2.3 IF Statement:

### Code :

```
#!/bin/bash
printf "\t\t\tIf Statement :\n"
echo "if name is Brad thn the if will execute."
printf "In Shell Code:\n\n"
echo 'read -p "Enter your name : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
fi'
echo ""
echo "Output:-"
read -p "Enter your name : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
fi
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script named "new.sh". The prompt is "(spyder@kali)-[~/Shell-Programming-Practice/Build]". The script output is as follows:

```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
If Statement :
if name is Brad thn the if will execute.
In Shell Code:

read -p "Enter your name : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
fi

Output:-
Enter your name : Brad
Your name is Brad.

(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

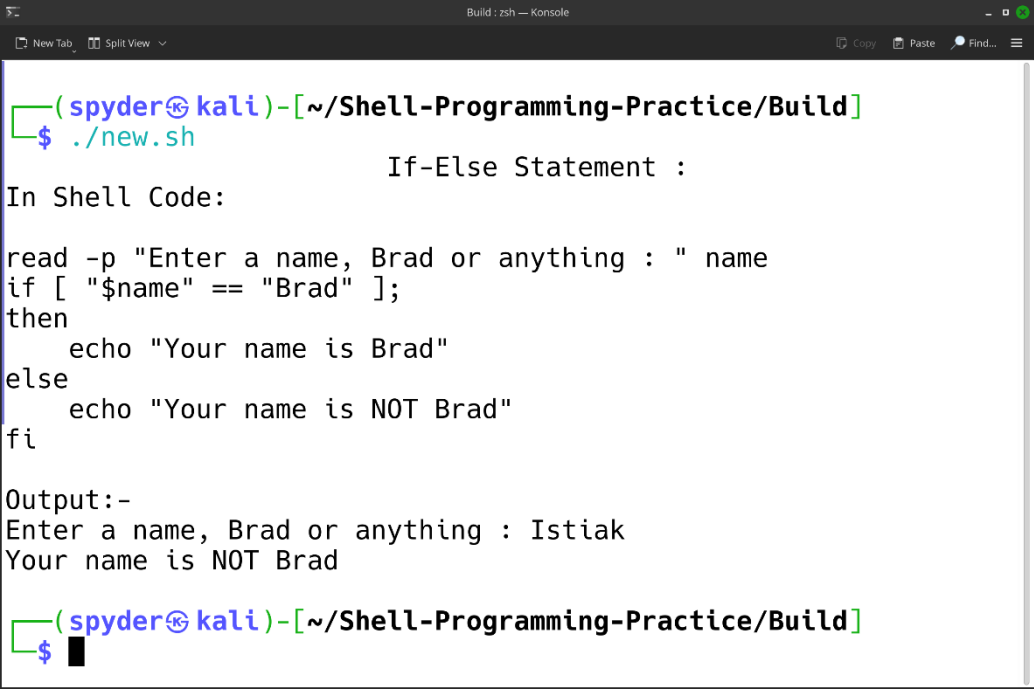
Fig 2.3 : If Statement

## 2.4 IF-ELSE Statement:

### Code :

```
#!/bin/bash
printf "\t\t\t4. If-Else Statement :\n"
printf "In Shell Code:\n\n"
echo 'read -p "Enter a name, Brad or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
else
    echo "Your name is NOT Brad"
fi'
printf "\nOutput:-\n"
read -p "Enter a name, Brad or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
else
    echo "Your name is NOT Brad"
fi
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali) - [~/Shell-Programming-Practice/Build]". The user enters "./new.sh". The script outputs "If-Else Statement :", "In Shell Code:", and the script code. Then it prompts "Output:-" and "Enter a name, Brad or anything :". The user enters "Istiak". The script outputs "Your name is NOT Brad". The prompt returns to "(spyder@kali) - [~/Shell-Programming-Practice/Build]".

```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
If-Else Statement :
In Shell Code:
read -p "Enter a name, Brad or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
else
    echo "Your name is NOT Brad"
fi
Output:-
Enter a name, Brad or anything : Istiak
Your name is NOT Brad
(spyder@kali) - [~/Shell-Programming-Practice/Build]
```

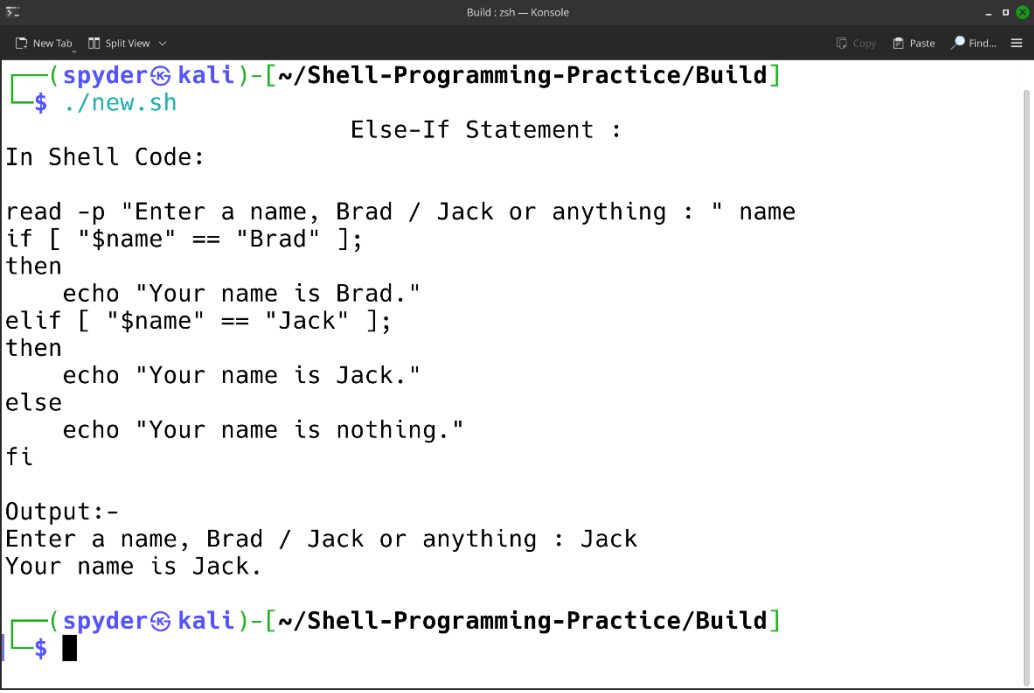
Fig 2.4 : If-else Statement

## 2.5 ELSE-IF Statement:

### Code :

```
#!/bin/bash
printf "\t\t\tElse-If Statement :\n"
printf "In Shell Code:\n\n"
echo 'read -p "Enter a name, Brad / Jack or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
elif [ "$name" == "Jack" ];
then
    echo "Your name is Jack."
else
    echo "Your name is nothing."
fi'
printf "\nOutput:-\n"
read -p "Enter a name, Brad / Jack or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
elif [ "$name" == "Jack" ];
then
    echo "Your name is Jack."
else
    echo "Your name is nothing."
fi
```

### Output :



The screenshot shows a terminal window titled "Build : zsh — Konsole". The prompt is "(spyder@kali)-[~/Shell-Programming-Practice/Build]". The user enters the command "\$ ./new.sh". The output of the script is as follows:

```
Else-If Statement :
In Shell Code:
read -p "Enter a name, Brad / Jack or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
elif [ "$name" == "Jack" ];
then
    echo "Your name is Jack."
else
    echo "Your name is nothing."
fi
Output:-
Enter a name, Brad / Jack or anything : Jack
Your name is Jack.
```

The terminal prompt returns to "(spyder@kali)-[~/Shell-Programming-Practice/Build]" with a new line cursor.

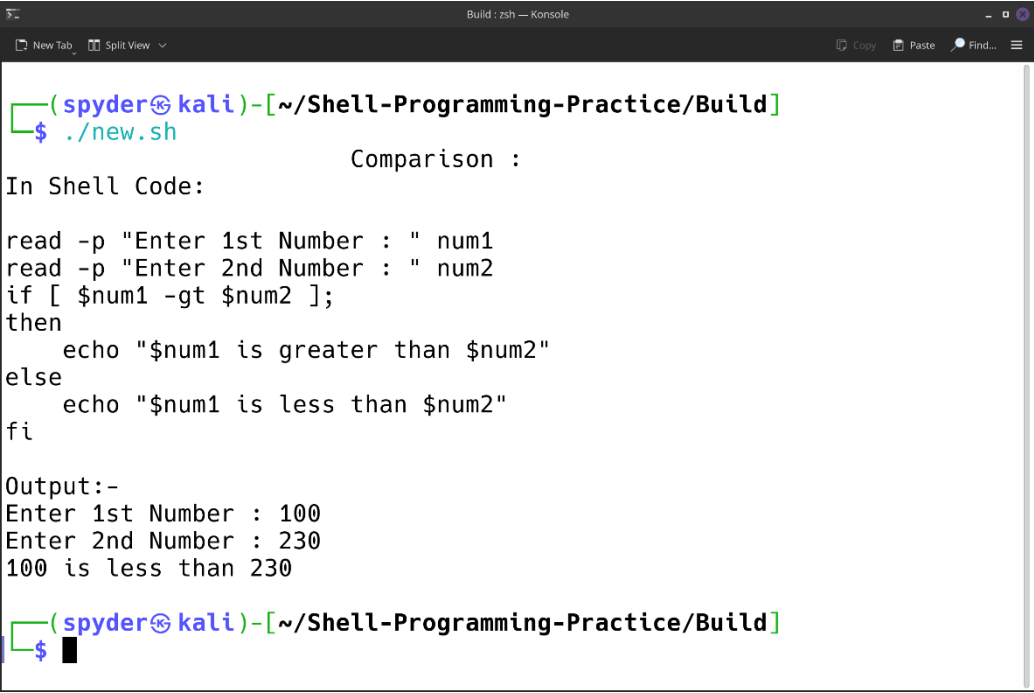
Fig 2.5 : Else-If Statement

## 2.6 Comparison using IF-ELSE Statement:

### Code :

```
#!/bin/bash
printf "\t\t\tComparison : \n"
printf "In Shell Code: \n\n"
echo 'read -p "Enter 1st Number : " num1
read -p "Enter 2nd Number : " num2
if [ $num1 -gt $num2 ];
then
    echo "$num1 is greater than $num2"
else
    echo "$num1 is less than $num2"
fi'
echo ""
echo "Output:-"
read -p "Enter 1st Number : " num1
read -p "Enter 2nd Number : " num2
if [ $num1 -gt $num2 ];
then
    echo "$num1 is greater than $num2"
else
    echo "$num1 is less than $num2"
fi
```

### Output :



The screenshot shows a terminal window titled "Build : zsh — Konsole". The prompt is "(spyder@kali)-[~/Shell-Programming-Practice/Build]". The user enters "\$ ./new.sh". The script outputs "Comparison :" and "In Shell Code:". It then prompts for "Enter 1st Number : " and "Enter 2nd Number : ". The user enters "100" and "230" respectively. The script outputs "100 is less than 230". The prompt returns to "(spyder@kali)-[~/Shell-Programming-Practice/Build]".

```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Comparison :
In Shell Code:
read -p "Enter 1st Number : " num1
read -p "Enter 2nd Number : " num2
if [ $num1 -gt $num2 ];
then
    echo "$num1 is greater than $num2"
else
    echo "$num1 is less than $num2"
fi
Output:-
Enter 1st Number : 100
Enter 2nd Number : 230
100 is less than 230
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

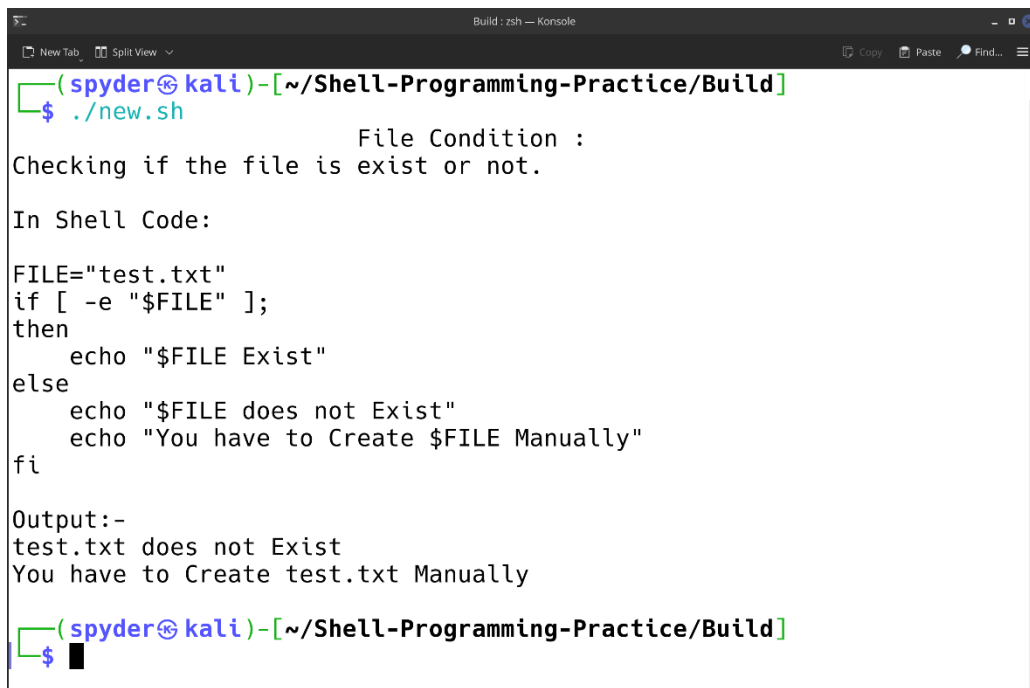
Fig 2.6 : Comparison using If Else

## 2.7 File Condition :

### Code :

```
#!/bin/bash
printf "\t\t\tFile Condition :\n"
printf "Checking if the file exists or not.\n\n"
printf "In Shell Code:\n\n"
echo 'FILE="test.txt"'
if [ -e "$FILE" ];
then
    echo "$FILE Exist"
else
    echo "$FILE does not Exist"
    echo "You have to Create $FILE Manually"
fi'
echo ""
echo "Output:-"
FILE="test.txt"
if [ -e "$FILE" ];
then
    echo "$FILE Exist"
else
    echo "$FILE does not Exist"
    echo "You have to Create $FILE Manually"
fi
```

### Output :



```
Build : zsh — Konsole
New Tab Split View Copy Paste Find...
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
File Condition :
Checking if the file is exist or not.
In Shell Code:
FILE="test.txt"
if [ -e "$FILE" ];
then
    echo "$FILE Exist"
else
    echo "$FILE does not Exist"
    echo "You have to Create $FILE Manually"
fi
Output:-
test.txt does not Exist
You have to Create test.txt Manually
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.7 : File Condition

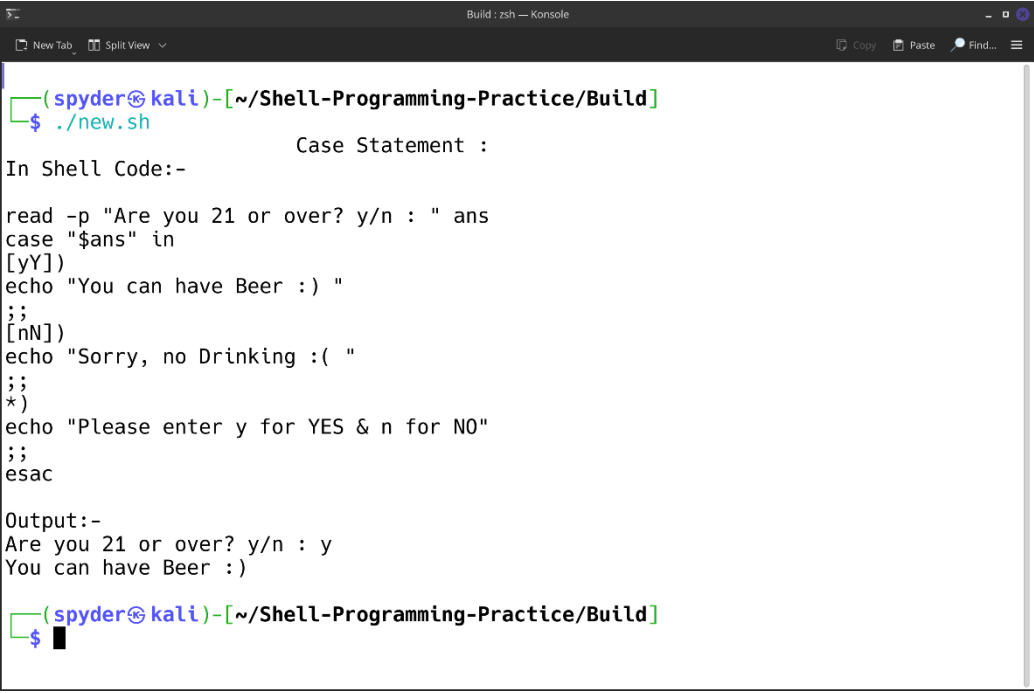


## 2.8 Case Statement:

### Code :

```
#!/bin/bash
printf "\t\t\tCase Statement :\n"
printf "In Shell Code:-\n\n"
echo 'read -p "Are you 21 or over? y/n : " ans
case "$ans" in [yY])
echo "You can have Beer  " ;;
[nN])
echo "Sorry, no Drinking  " ;;
*)
echo "Please enter 'y' for YES & 'n' for NO" ;;
esac'
echo ""
echo "Output:-"
read -p "Are you 21 or over? y/n : " ans
case "$ans" in [yY])
echo "You can have Beer :) "
;;
[nN])
echo "Sorry, no Drinking :( "
;;
*)
echo "Please enter 'y' for YES & 'n' for NO"
;;
esac
```

### Output :



```
Build : zsh — Konsole
New Tab Split View Copy Paste Find...
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Case Statement :
In Shell Code:-
read -p "Are you 21 or over? y/n : " ans
case "$ans" in
[yY])
echo "You can have Beer :) "
;;
[nN])
echo "Sorry, no Drinking :( "
;;
*)
echo "Please enter y for YES & n for NO"
;;
esac

Output:-
Are you 21 or over? y/n : y
You can have Beer :)

(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

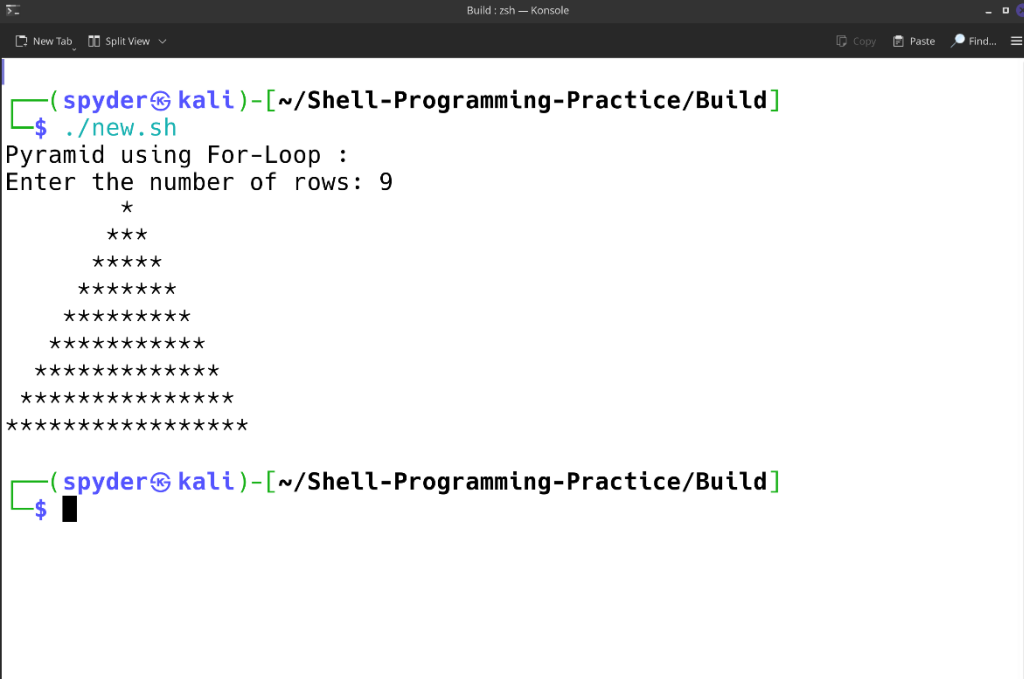
Fig 2.8 : Case Statement

## 2.9 Pyramid using For Loop Statement:

### Code :

```
#!/bin/bash
printf "Pyramid using For-Loop :\n"
read -p "Enter the number of rows: " rows
# Loop to print the pyramid
for (( i=1; i<=rows; i++ ))
do
    # Print spaces before stars
    for (( j=i; j<rows; j++ ))
    do
        echo -n " "
    done
    # Print stars
    for (( j=1; j<=(2*i-1); j++ ))
    do
        echo -n "*"
    done
    # Move to the next line
    echo
done
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows a user prompt "(spyder@kali)-[~/Shell-Programming-Practice/Build]" followed by the command "\$ ./new.sh". The output of the script is a pyramid of stars with 9 rows. The first row has 1 star, the second has 3 stars, the third has 5 stars, and so on, up to the ninth row which has 17 stars. The prompt "(spyder@kali)-[~/Shell-Programming-Practice/Build]" is shown again at the bottom of the terminal window.

```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Pyramid using For-Loop :
Enter the number of rows: 9
  *
 ***
*****
*****
*****
*****
*****
*****
*****
*****

(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

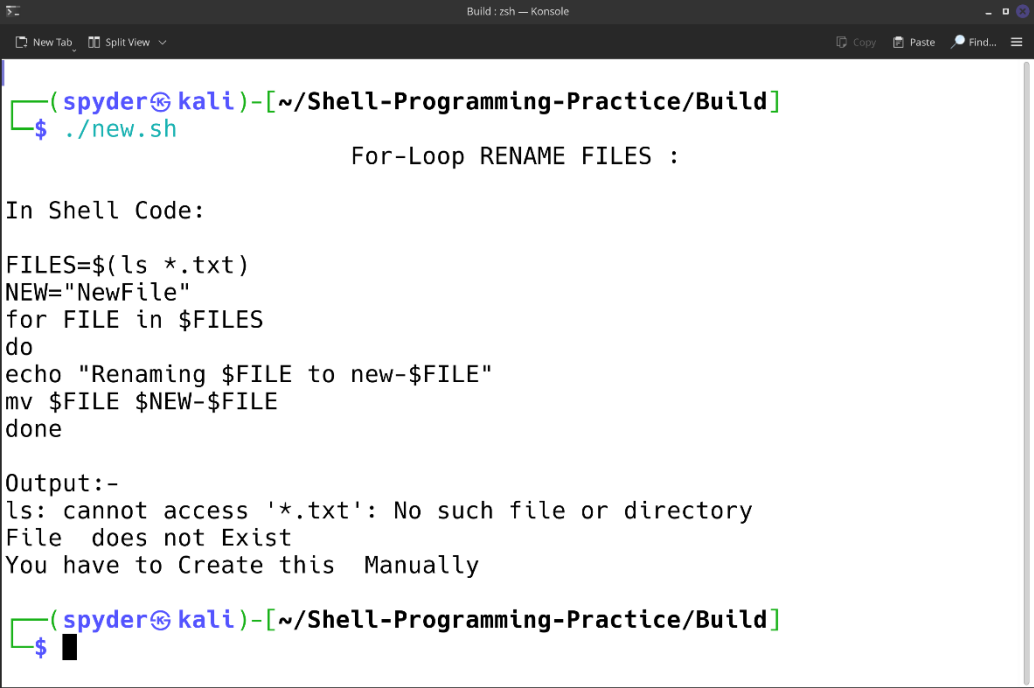
Fig 2.9 : Pyramid using For Loop

## 2.10 For Loop File Renaming :

### Code :

```
#!/bin/bash
printf "\t\t\tFor-Loop RENAME FILES :\n"
printf "\nIn Shell Code:\n\n"
echo 'FILES=$(ls *.txt)
NEW="NewFile"
for FILE in $FILES
do
echo "Renaming $FILE to new-$FILE"
mv $FILE $NEW-$FILE
done'
printf "\nOutput:-\n"
FILES=$(ls *.txt)
NEW="NewFile"
if [ -e "$FILES" ];
then {
for FILE in $FILES
do
echo "Renaming $FILE to new-$FILE"
mv $FILE $NEW-$FILE
done
}
else
echo "File $FILE does not Exist"
echo "You have to Create this $FILE Manually"
fi
```

### Output :



The screenshot shows a terminal window titled "Build : zsh — Konsole". The prompt is "(spyder@kali)-[~/Shell-Programming-Practice/Build]". The user enters the command `./new.sh`. The output of the script is as follows:

```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
For-Loop RENAME FILES :

In Shell Code:

FILES=$(ls *.txt)
NEW="NewFile"
for FILE in $FILES
do
echo "Renaming $FILE to new-$FILE"
mv $FILE $NEW-$FILE
done

Output:-
ls: cannot access '*.txt': No such file or directory
File does not Exist
You have to Create this Manually

(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.10 : File Rename using For loop

## 2.11 Until Loop Statement:

### Code :

```
#!/bin/bash
printf "\t\t\t\t\tUntil Loop Statement : \n"
printf "\nIn Shell Code:\n\n"
echo 'count=5
until [ $count -le 0 ]
do
    echo "Countdown: $count"
    ((count--))
done'
printf "\nOutput:-\n"
count=5
until [ $count -le 0 ]
do
    echo "Countdown: $count"
    ((count--))
done
```

### Output :



```
Build : zsh - Konsole
New Tab Split View Copy Paste Find...
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Until Loop Statement :

In Shell Code:

count=5
until [ $count -le 0 ]
do
    echo "Countdown: $count"
    ((count--))
done

Output:-
Countdown: 5
Countdown: 4
Countdown: 3
Countdown: 2
Countdown: 1

(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

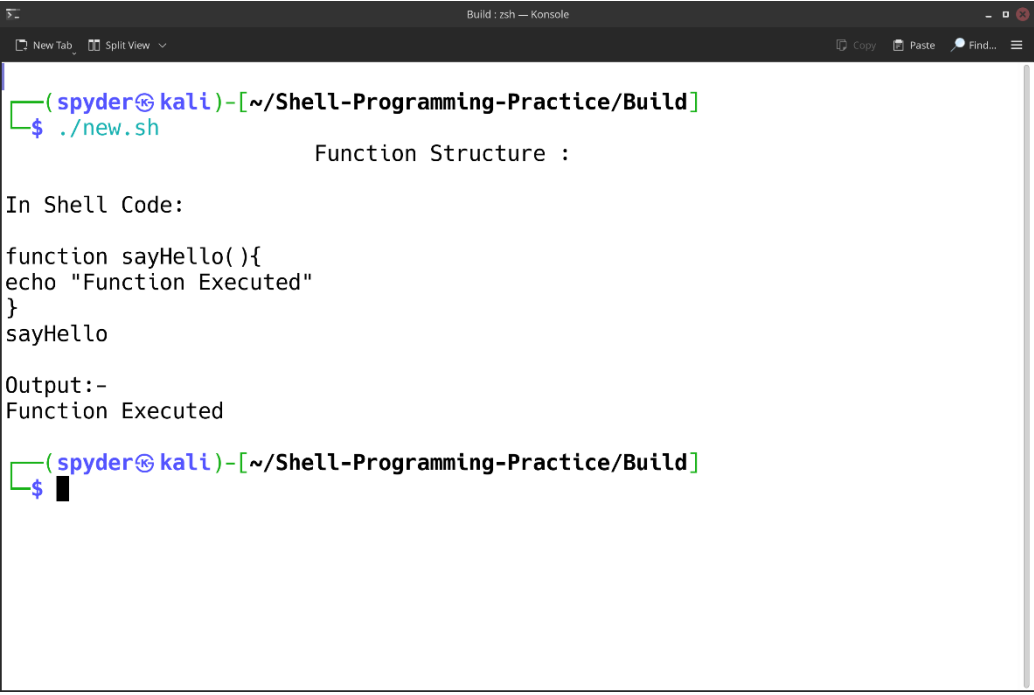
Fig 2.11 : Until Loop Statement

## 2.12 Function Structure :

### Code :

```
#!/bin/bash
printf "\t\t\tFunction Structure :\n"
printf "\nIn Shell Code:\n\n"
#FUNCTION STRUCTURE
echo 'function sayHello(){
echo "Function Executed"
}
sayHello'
printf "\nOutput:-\n"
function sayHello(){
echo "Function Executed"
}
sayHello
```

### Output :



```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Function Structure :

In Shell Code:

function sayHello(){
echo "Function Executed"
}
sayHello

Output:-
Function Executed

(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

Fig 2.12 : Function Structure

## 2.13 Function with parameter:

### Code :

```
#!/bin/bash
printf "\t\t\tFunction with Parameter :\n"
printf "\nIn Shell Code:\n\n"
echo 'function greet(){
echo "Hello, I am $1 and I am $2"
}
greet "Brad" "36"'
printf "\nOutput:-\n"
#FUNCTION WITH PARAMETERS
function greet(){
echo "Hello, I am $1 and I am $2"
}
greet "Brad" "36"
```

### Output :

```
(spyder@kali)~[~/Shell-Programming-Practice/Build]
$ ./new.sh
Function with Parameter :
In Shell Code:
function greet(){
echo "Hello, I am $1 and I am $2"
}
greet "Brad" "36"
Output:-
Hello, I am Brad and I am 36
(spyder@kali)~[~/Shell-Programming-Practice/Build]
```

Fig 2.13 : Function with Parameter

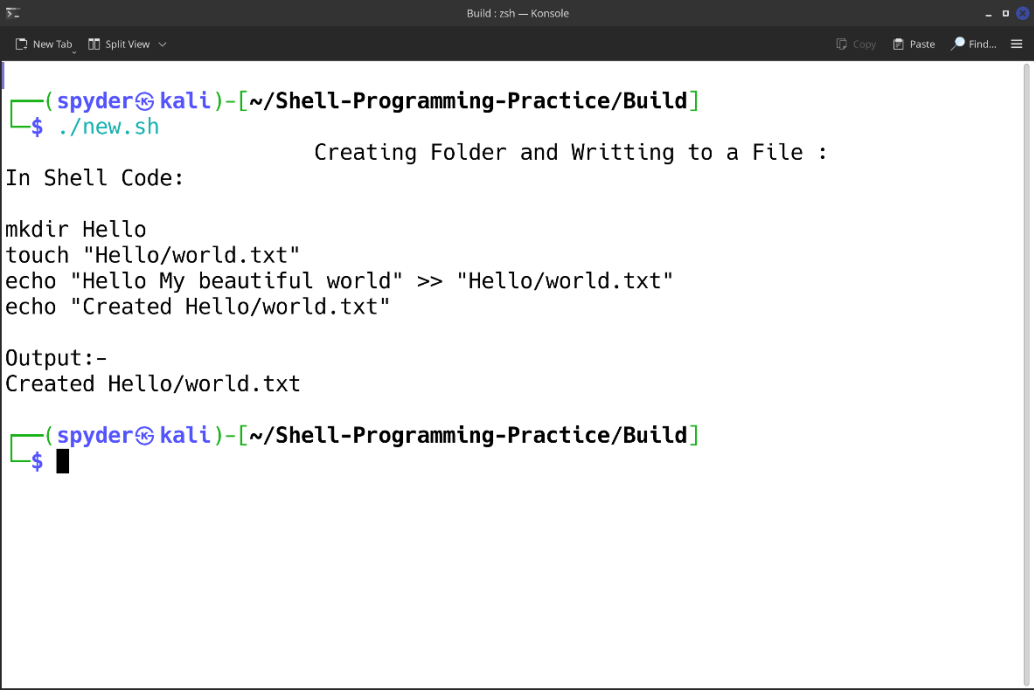
## 2.14 Creating Folder and writing to a file:

### Code :

```
#!/bin/bash
printf "\t\t\t\t\tCreating Folder and writting to a File : "
printf "\nIn Shell Code:\n\n"
echo 'mkdir Hello
touch "Hello/world.txt"
echo "Hello My beautiful world" >> "Hello/world.txt"
echo "Created Hello/world.txt"'

printf "\nOutput:-\n"
#CREATE FOLDER AND WRITE TO A FILE
mkdir Hello
touch "Hello/world.txt"
echo "Hello My beautiful world" >> "Hello/world.txt"
echo "Created Hello/world.txt"
```

### Output :



```
Build : zsh — Konsole
New Tab Split View
Copy Paste Find...
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Creating Folder and Writting to a File :
In Shell Code:
mkdir Hello
touch "Hello/world.txt"
echo "Hello My beautiful world" >> "Hello/world.txt"
echo "Created Hello/world.txt"
Output:-
Created Hello/world.txt
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.14 : Folder & write in file

## 2.15 Array Declaration:

### Code :

```
#!/bin/bash
printf "\t\t\t\tArray Declaration\n"
printf "In Shell Code:-\n\n"
echo 'fruits=("apple" "banana" "cherry")'
echo 'echo "Array of fruits : ${fruits[]}"'
printf "\nOutput:-\n"
fruits=("apple" "banana" "cherry")
echo "Array of fruits : ${fruits[]}"
echo ""
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali)~[~/Shell-Programming-Practice/Build]". The user enters "\$ ./new.sh". The output is: "Array Declaration", "In Shell Code:-", "fruits=(\"apple\" \"banana\" \"cherry\")", "echo \"Array of fruits : \${fruits[\*]}\"", "Output:-", "Array of fruits : apple banana cherry". The prompt then returns to "(spyder@kali)~[~/Shell-Programming-Practice/Build]".

```
(spyder@kali)~[~/Shell-Programming-Practice/Build]
$ ./new.sh
Array Declaration
In Shell Code:-
fruits=("apple" "banana" "cherry")
echo "Array of fruits : ${fruits[*]}"
Output:-
Array of fruits : apple banana cherry
(spyder@kali)~[~/Shell-Programming-Practice/Build]
```

Fig 2.15 : Array Declaration

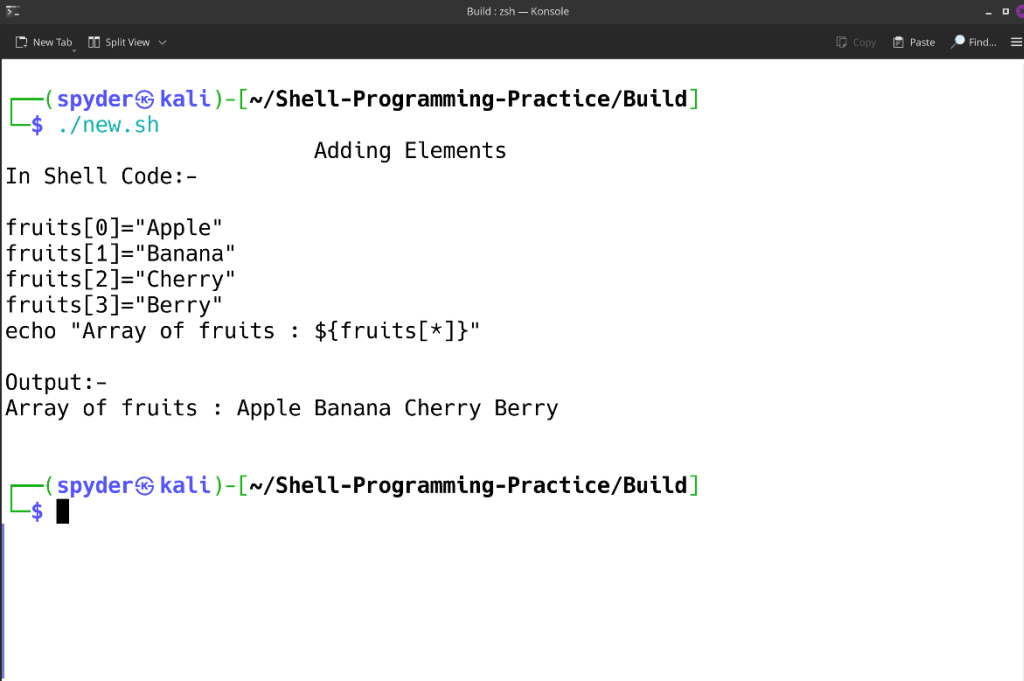


## 2.16 Adding elements in Array :

### Code :

```
#!/bin/bash
printf "\t\t\tAdding Elements\n"
printf "In Shell Code:-\n\n"
echo 'fruits[0]="Apple"'
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Array of fruits : ${fruits[*]}"
printf "\nOutput:-\n"
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Array of fruits : ${fruits[*]}"
echo ""
```

### Output :



```
Build : zsh — Konsole
New Tab Split View
Copy Paste Find...
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Adding Elements
In Shell Code:-
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Array of fruits : ${fruits[*]}"
Output:-
Array of fruits : Apple Banana Cherry Berry
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

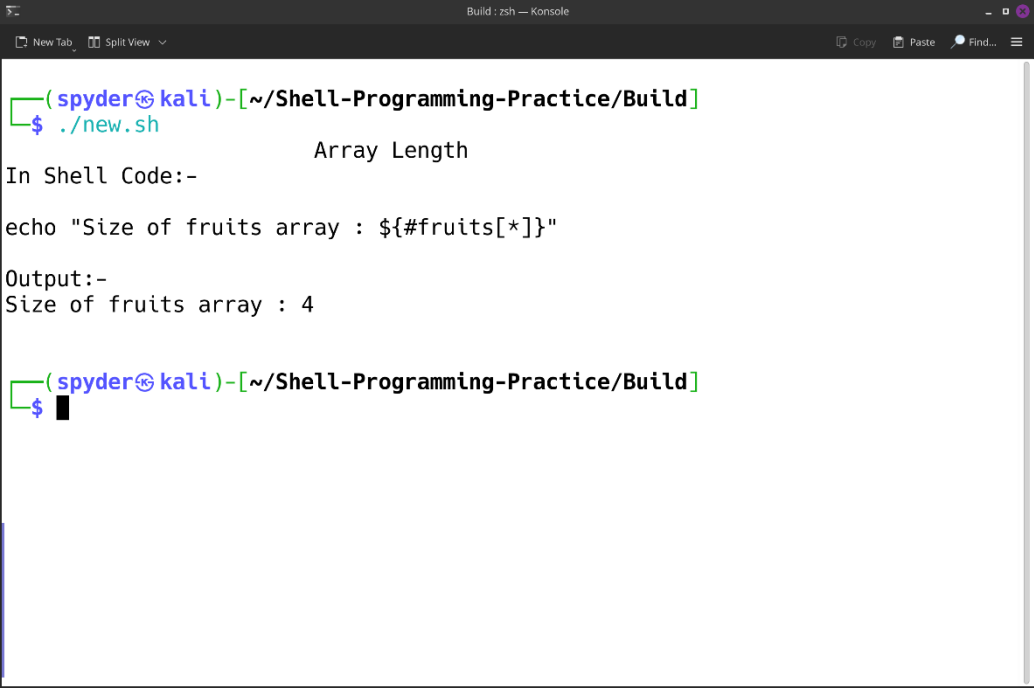
Fig 2.16 : Adding elements in Array

## 2.17 Array Length :

### Code :

```
#!/bin/bash
printf "\t\t\tArray Length\n"
printf "In Shell Code:-\n\n"
echo 'echo "Size of fruits array : ${#fruits[*]}"'
printf "\nOutput:-\n"
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Size of fruits array : ${#fruits[*]}"
echo ""
```

### Output :



```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Array Length
In Shell Code:-
echo "Size of fruits array : ${#fruits[*]}"
Output:-
Size of fruits array : 4
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

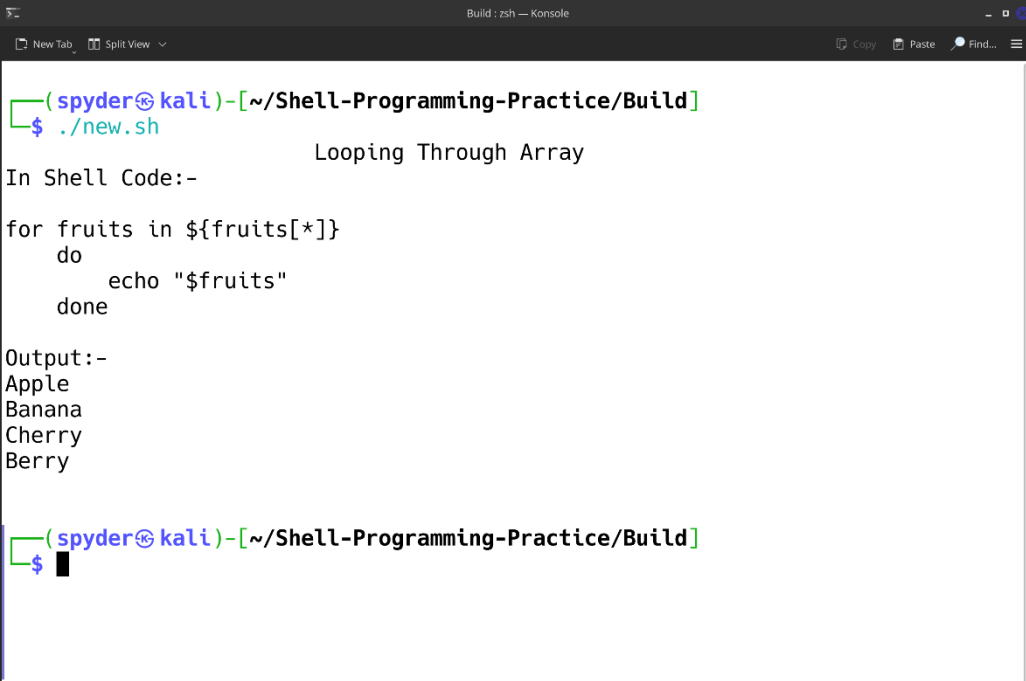
Fig 2.17 : Array length

## 2.18 Looping through Array :

### Code :

```
#!/bin/bash
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
printf "\t\t\tLooping Through Array\n"
printf "In Shell Code:-\n\n"
echo 'for fruits in ${fruits[*]}
do
    echo "$fruits"
done'
printf "\nOutput:-\n"
for fruits in ${fruits[*]}
do
echo "$fruits"
done
echo ""
```

### Output :



```
Build : zsh — Konsole
New Tab Split View Copy Paste Find...
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Looping Through Array
In Shell Code:-
for fruits in ${fruits[*]}
do
    echo "$fruits"
done
Output:-
Apple
Banana
Cherry
Berry
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.18 : looping through array

## 2.19 Update Array Element :

### Code :

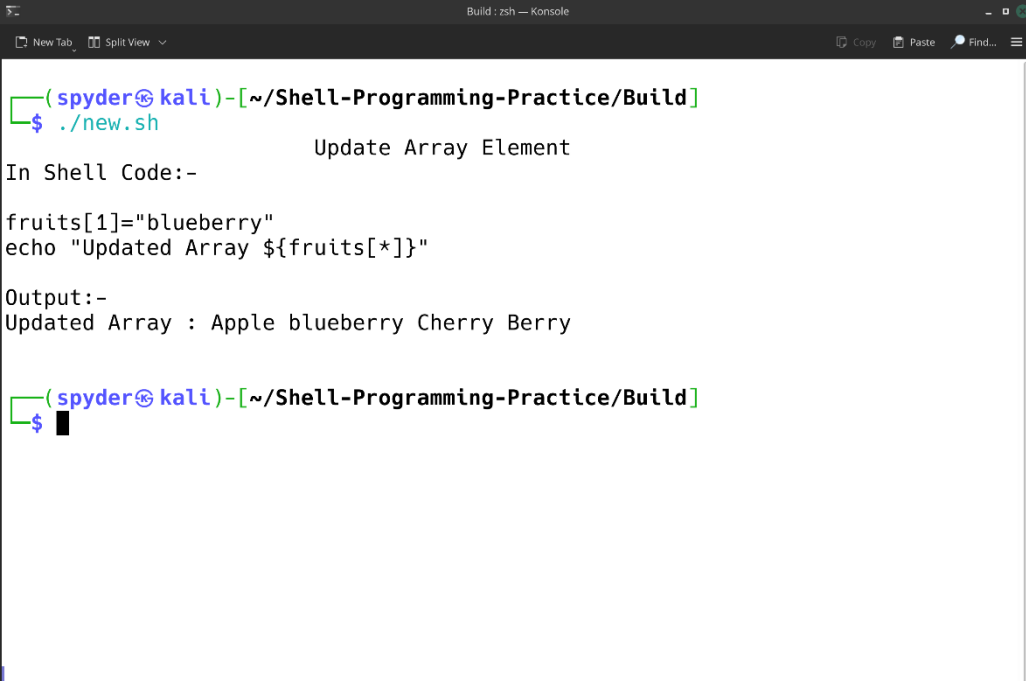
```
#!/bin/bash
```

```
fruits[0]="Apple"  
fruits[1]="Banana"  
fruits[2]="Cherry"  
fruits[3]="Berry"
```

```
printf "\t\t\tUpdate Array Element\n"  
printf "In Shell Code:-\n\n"  
echo 'fruits[1]="blueberry"  
echo "Updated Array ${fruits[*]}"'
```

```
printf "\nOutput:-\n"  
fruits[1]="blueberry"  
echo "Updated Array : ${fruits[*]}" # Output: apple blueberry  
cherry  
echo ""
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali) - [~/Shell-Programming-Practice/Build]". The user enters "\$ ./new.sh". The script outputs "Update Array Element", "In Shell Code:-", "fruits[1]='blueberry'", and "echo 'Updated Array \${fruits[\*]}'". Then it outputs "Output:-" and "Updated Array : Apple blueberry Cherry Berry". The prompt returns to "(spyder@kali) - [~/Shell-Programming-Practice/Build]".

```
(spyder@kali) - [~/Shell-Programming-Practice/Build]  
$ ./new.sh  
Update Array Element  
In Shell Code:-  
fruits[1]="blueberry"  
echo "Updated Array ${fruits[*]}"  
  
Output:-  
Updated Array : Apple blueberry Cherry Berry  
  
(spyder@kali) - [~/Shell-Programming-Practice/Build]  
$
```

Fig 2.19 : Update element

## 2.20 Insert Element in Array :

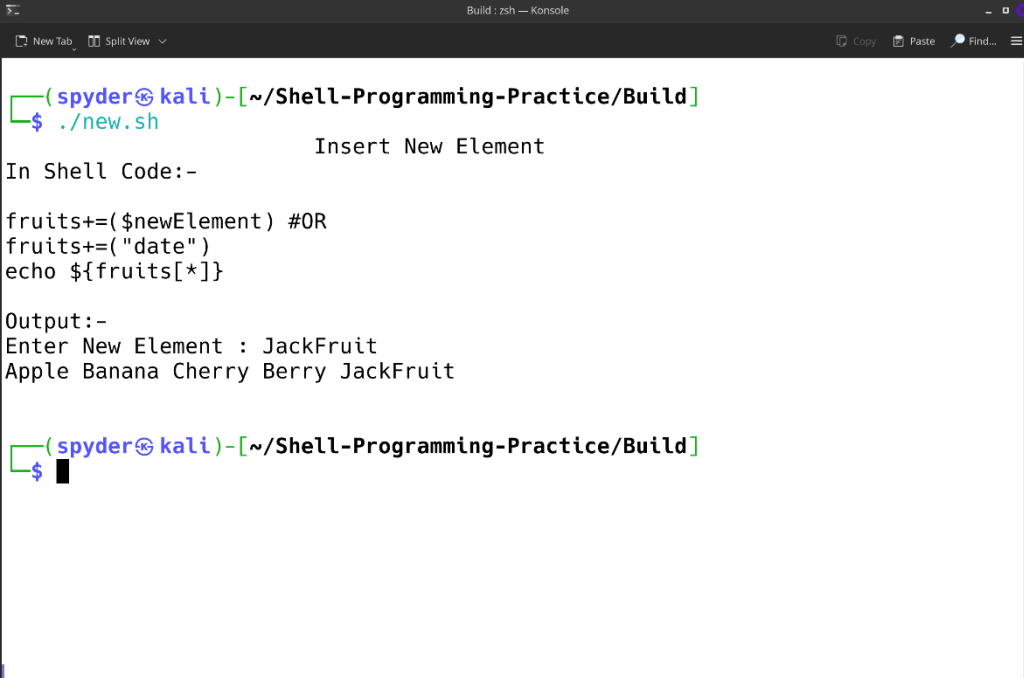
### Code :

```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tInsert New Element\n"
printf "In Shell Code:-\n\n"
echo 'fruits+=($newElement) #OR
fruits+=("date")
echo ${fruits[*]}'
printf "\nOutput:-\n"
read -p "Enter New Element : " nw
fruits+=($nw)
echo ${fruits[*]}
echo ""
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali) - [~/Shell-Programming-Practice/Build]". The user enters "./new.sh". The script outputs "Insert New Element", followed by "In Shell Code:-". It then displays the script's logic: "fruits+=(\$newElement) #OR", "fruits+=('date')", and "echo \${fruits[\*]}". Below this, it says "Output:-" and prompts "Enter New Element : ". The user enters "JackFruit". The final output is "Apple Banana Cherry Berry JackFruit". The terminal prompt returns to "(spyder@kali) - [~/Shell-Programming-Practice/Build]".

```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Insert New Element
In Shell Code:-
fruits+=($newElement) #OR
fruits+=('date')
echo ${fruits[*]}
Output:-
Enter New Element : JackFruit
Apple Banana Cherry Berry JackFruit
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

Fig 2.20 : Insert New Element

## 2.21 Delete Element in Array :

### Code :

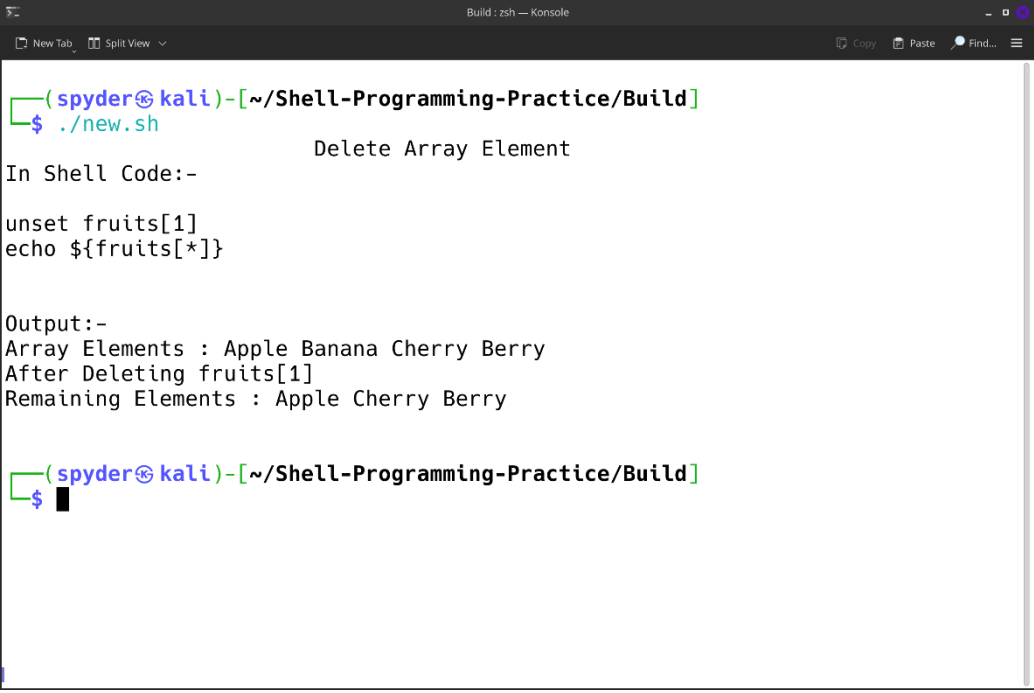
```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tDelete Array Element\n"
printf "In Shell Code:-\n\n"
echo 'unset fruits[1]
echo ${fruits[*]}
'

printf "\nOutput:-\n"
echo "Array Elements : ${fruits[*]}"
echo "After Deleting fruits[1]"
unset fruits[1]
echo "Remaining Elements : ${fruits[*]}"
echo ""
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali)~[~/Shell-Programming-Practice/Build]". The user enters "\$ ./new.sh". The script outputs "Delete Array Element", "In Shell Code:-", "unset fruits[1]", and "echo \${fruits[\*]}". Then it outputs "Output:-", "Array Elements : Apple Banana Cherry Berry", "After Deleting fruits[1]", and "Remaining Elements : Apple Cherry Berry". The prompt returns to "(spyder@kali)~[~/Shell-Programming-Practice/Build]".

```
(spyder@kali)~[~/Shell-Programming-Practice/Build]
$ ./new.sh
Delete Array Element
In Shell Code:-
unset fruits[1]
echo ${fruits[*]}

Output:-
Array Elements : Apple Banana Cherry Berry
After Deleting fruits[1]
Remaining Elements : Apple Cherry Berry

(spyder@kali)~[~/Shell-Programming-Practice/Build]
$
```

Fig 2.21 : Delete Element

## 2.22 Delete Entire Array :

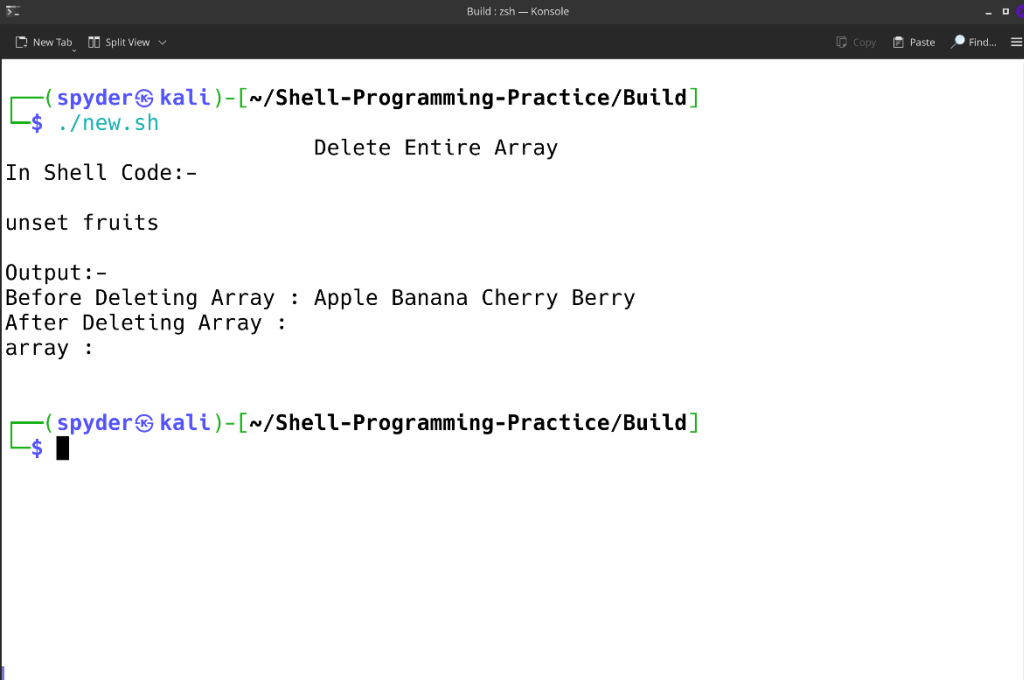
### Code :

```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tDelete Entire Array\n"
printf "In Shell Code:-\n\n"
echo 'unset fruits'
printf "\nOutput:-\n"
echo "Before Deleting Array : ${fruits[*]}"
echo "After Deleting Array :"
unset fruits
echo "array : ${fruits[*]}"
echo ""
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali)-[~/Shell-Programming-Practice/Build]". The user enters "\$ ./new.sh". The output is: "Delete Entire Array", "In Shell Code:-", "unset fruits", "Output:-", "Before Deleting Array : Apple Banana Cherry Berry", "After Deleting Array :", and "array :". The prompt returns to "(spyder@kali)-[~/Shell-Programming-Practice/Build]".

```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Delete Entire Array
In Shell Code:-
unset fruits
Output:-
Before Deleting Array : Apple Banana Cherry Berry
After Deleting Array :
array :
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.22 : Delete an Array

## 2.23 Declaring Associative Array :

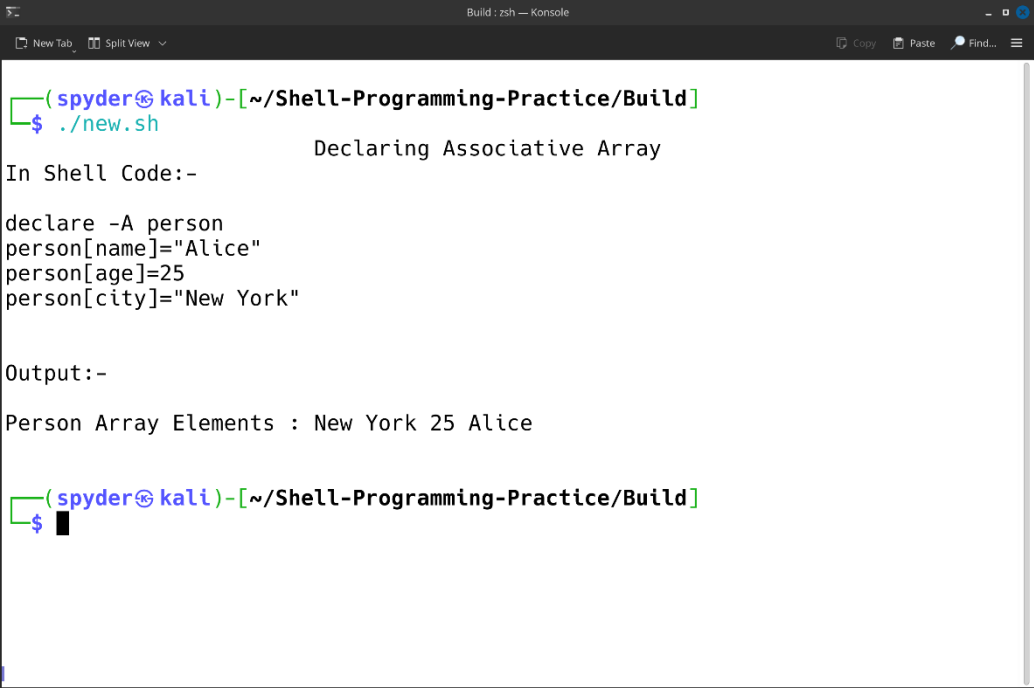
### Code :

```
#!/bin/bash

printf "\t\t\tDeclaring Associative Array\n"
printf "In Shell Code:-\n\n"
echo 'declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
'

printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
echo ""
echo "Person Array Elements : ${person[*]}"
echo ""
```

### Output :

A screenshot of a terminal window titled "Build : zsh — Konsole". The terminal shows the execution of a script. The prompt is "(spyder@kali) - [~/Shell-Programming-Practice/Build]". The user enters "\$ ./new.sh". The output is: "Declaring Associative Array", "In Shell Code:-", "declare -A person", "person[name]='Alice'", "person[age]=25", "person[city]='New York'", "Output:-", and "Person Array Elements : New York 25 Alice". The prompt returns to "(spyder@kali) - [~/Shell-Programming-Practice/Build]".

```
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Declaring Associative Array
In Shell Code:-
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"

Output:-
Person Array Elements : New York 25 Alice

(spyder@kali) - [~/Shell-Programming-Practice/Build]
```

Fig 2.23 : Associative Array



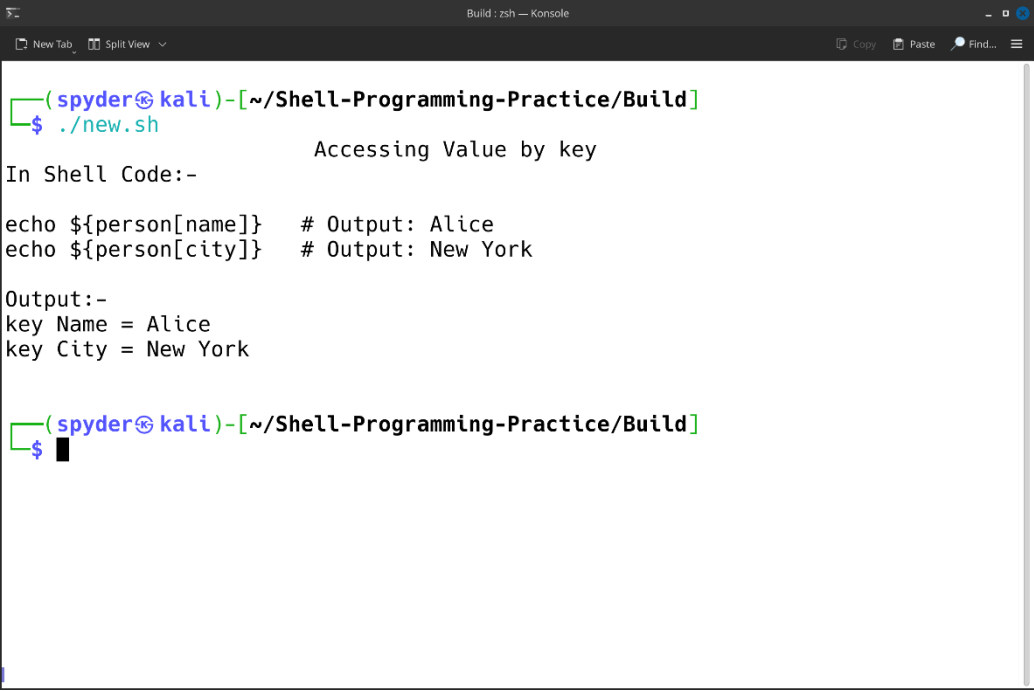
## 2.24 Accessing Value by key in Associative array :

### Code :

```
#!/bin/bash

printf "\t\t\tAccessing Value by key\n"
printf "In Shell Code:-\n\n"
echo 'echo ${person[name]}    # Output: Alice'
echo ${person[city]}    # Output: New York'
printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
echo "key Name = ${person[name]}"    # Output: Alice
echo "key City = ${person[city]}"    # Output: New York
echo ""
```

### Output :



```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Accessing Value by key
In Shell Code:-
echo ${person[name]}    # Output: Alice
echo ${person[city]}    # Output: New York
Output:-
key Name = Alice
key City = New York
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.24 : Accessing value by Key

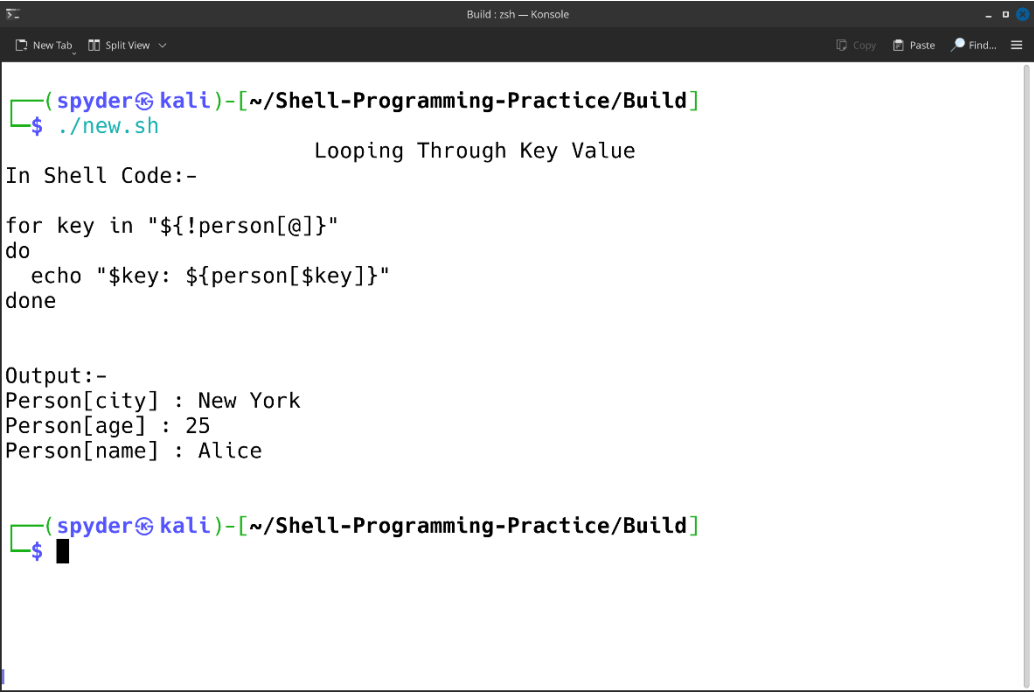
## 2.25 Looping through key in Associative array :

### Code :

```
#!/bin/bash
printf "\t\t\t\tLooping Through Key Value\n"
printf "In Shell Code:-\n\n"
echo 'for key in "${!person[@]}"
do
    echo "$key: ${person[$key]}"
done
'

printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
for key in "${!person[@]}"
do
    echo "Person[$key] : ${person[$key]}"
done
echo ""
```

### Output :



```
Build : zsh — Konsole
New Tab Split View
Copy Paste Find...
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$ ./new.sh
Looping Through Key Value
In Shell Code:-
for key in "${!person[@]}"
do
    echo "$key: ${person[$key]}"
done

Output:-
Person[city] : New York
Person[age] : 25
Person[name] : Alice
(spyder@kali) - [~/Shell-Programming-Practice/Build]
$
```

Fig 2.25 : Looping through Key value

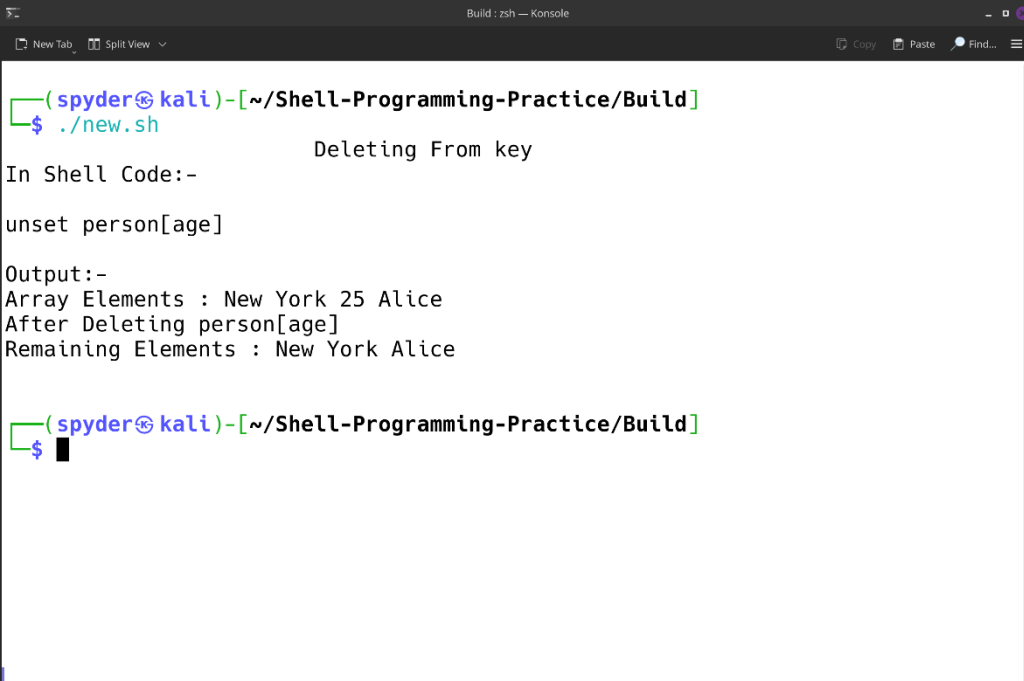
## 2.26 Deleting from key in Associative array :

### Code :

```
#!/bin/bash

printf "\t\t\tDeleting From key\n"
printf "In Shell Code:-\n\n"
echo 'unset person[age]'
printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
echo "Array Elements : ${person[*]}"
echo "After Deleting person[age]"
unset person[age]
echo "Remaining Elements : ${person[*]}"
echo ""
```

### Output :



The screenshot shows a terminal window titled "Build : zsh — Konsole". The prompt is "(spyder@kali) - [~/Shell-Programming-Practice/Build]". The user enters the command `./new.sh`. The script's output is displayed: "Deleting From key", "In Shell Code:-", "unset person[age]", "Output:-", "Array Elements : New York 25 Alice", "After Deleting person[age]", and "Remaining Elements : New York Alice". The terminal shows the command prompt again after the script finishes.

Fig 2.26 : Deleting From key

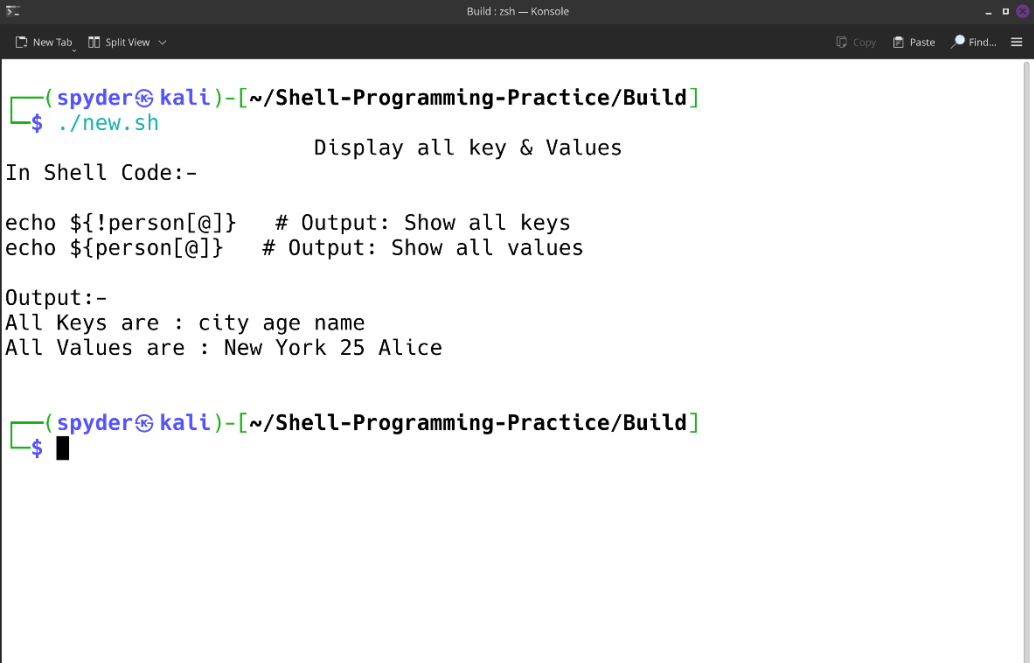
## 2.27 Display all key & values of Associative array :

### Code :

```
#!/bin/bash

printf "\t\t\tDisplay all key & Values\n"
printf "In Shell Code:-\n\n"
echo 'echo ${!person[@]} # Output: Show all keys'
echo '${person[@]} # Output: Show all values'
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
printf "\nOutput:-\n"
echo "All Keys are : ${!person[@]}" # Output: Show all keys
echo "All Values are : ${person[@]}" # Output: Show all values
echo ""
```

### Output :



The screenshot shows a terminal window titled "Build : zsh — Konsole". The prompt is "(spyder@kali)-[~/Shell-Programming-Practice/Build]". The user runs the command "\$ ./new.sh". The output of the script is displayed, including the header "Display all key & Values", the message "In Shell Code:-", and the two echo commands. Below this, the output of the script is shown: "Output:-", "All Keys are : city age name", and "All Values are : New York 25 Alice". The terminal window also shows the prompt "(spyder@kali)-[~/Shell-Programming-Practice/Build]" and a cursor at the end of the line.

```
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$ ./new.sh
Display all key & Values
In Shell Code:-
echo ${!person[@]} # Output: Show all keys
echo ${person[@]} # Output: Show all values
Output:-
All Keys are : city age name
All Values are : New York 25 Alice
(spyder@kali)-[~/Shell-Programming-Practice/Build]
$
```

Fig 2.27 : Display all key & values