Command line peer learning Document

Question 1. Write a bash script to get the current date, time, username, home directory and current working directory.

```
#!/bin/bash

#Question 1

#Command to get current date and time
echo "Current date and time is:  $(date)"

#Command to get only date
echo "Date is: $(date +%F)"

#Command to get only time
echo "Time is: $(date +%T)"

#Command to get username
echo "Username: $(whoami)"

#Command to get Home Directory
echo "Home Directory: $HOME"

#Command to get users current working directory
echo "Current Working Directory: $(pwd)"
```

My script's explanation

echo "Current date and time is: $(date)": This line prints the string "Current date and time is: ", followed by the output of the date command (which returns the current date and time).

echo "Date is: $(date +%F)": This line prints the string "Date is: ", followed by the output of the date +%F command. The %F is a format specifier that tells the date command to output the date in the format "YYYY-MM-DD".

echo "Time is: $(date +%T)": This line prints the string "Time is: ", followed by the output of the date +%T command. The %T format specifier tells the date command to output the time in the format "HH:MM:SS".

echo "Username: $(whoami)": This line prints the string "Username: ", followed by the output of the whoami command (which returns the current user's username).

echo "Home Directory: $HOME": This line prints the string "Home Directory: ", followed by the value of the $HOME environment variable (which contains the path to the current user's home directory).

echo "Current Working Directory: $(pwd)": This line prints the string "Current Working Directory: ", followed by the output of the pwd command (which returns the current working directory).

Atul's Solution

```
current_date=$(date +"Year: %Y, Month: %m, Day: %d") # Command to fetch the date
current_time=$(date +"%T") # Command to fetch the time
current_user=$(whoami) # Command to fetch the current working user
home_directory=$(echo $HOME) # Command to fetch the Home directory
current_directory=$(pwd) # Command to fetch the current wokring directory

# Printing the fetched variables
echo "Current date: $current_date"
echo "Current time: $current_time"
echo "Username: $current_user"
echo "Home directory: $home_directory"
echo "Current working directory: $current_directory"

exit 0
```

current_date=$(date +"Year: %Y, Month: %m, Day: %d"): This line sets the current_date variable to the output of the date +"Year: %Y, Month: %m, Day: %d" command, which returns the current date in the format "Year: YYYY, Month: MM, Day: DD".

current_time=$(date +"%T"): This line sets the current_time variable to the output of the date +"%T" command, which returns the current time in the format "HH:MM:SS".

current_user=$(whoami): This line sets the current_user variable to the output of the whoami command, which returns the current user's username.

home_directory=$(echo $HOME): This line sets the home_directory variable to the value of the $HOME environment variable, which contains the path to the current user's home directory.

current_directory=$(pwd): This line sets the current_directory variable to the output of the pwd command, which returns the current working directory.

echo "Current date: $current_date": This line prints the string "Current date: ", followed by the value of the current_date variable.

echo "Current time: $current_time": This line prints the string "Current time: ", followed by the value of the current_time variable.

echo "Username: $current_user": This line prints the string "Username: ", followed by the value of the current_user variable.

echo "Home directory: $home_directory": This line prints the string "Home directory: ", followed by the value of the home_directory variable.

echo "Current working directory: $current_directory": This line prints the string "Current working directory: ", followed by the value of the current_directory variable.

exit 0: This line exits the script with a status code of 0, indicating success.

Sarthak's solution

```
echo Answer 1-
#command to get only date
echo current date : $(date +%F )
#command to get only time
echo current time : $(date +%T )
#command to get Username
echo Username     : $(whoami)
#command to get Home diretory
echo Home directory: ~
#command to get current working directory
echo Current working directory :$(pwd)
```

echo current date : $(date +%F): This line prints the string "current date : ", followed by the output of the date +%F command. The %F is a format specifier that tells the date command to output the date in the format "YYYY-MM-DD".

echo current time : $(date +%T): This line prints the string "current time : ", followed by the output of the date +%T command. The %T format specifier tells the date command to output the time in the format "HH:MM:SS".

echo Username : $(whoami): This line prints the string "Username : ", followed by the output of the whoami command, which returns the current user's username.

echo Home directory: ~: This line prints the string "Home directory: ~", where the tilde (~) is a shorthand for the current user's home directory. It's equivalent to using the $HOME environment variable, but shorter.

echo Current working directory :$(pwd): This line prints the string "Current working directory :", followed by the output of the pwd command, which returns the current working directory.

Question 2. Write a bash script (name Table.sh) to print the Table of a number by using a while loop. It should support the following requirements. ● The script should accept the input from the command line. ● If you don’t input any data, then display an error message to execute the script correctly.

My approach

```
#!/bin/bash


#Question 2

#Taking user input
echo "Enter the number: "
read n


# Checking through case statement
case $n in
#Regular expression to check if the given input is not a number. If satisfies it results in a invalid input
*[^0-9]*)
     echo "Please input only number!! $n is not a number"
     ;;
#Regex to check for number
[0-9]*)
    i=1

    while [ $i -le 10 ]
    do
    res=`expr $i \* $n`

    echo "$n * $i = $res"

   (( ++i))

    done
    ;;
#Condition that will handle the situation when there is no input
*)
    echo "Error!! Please Provide Input To Get The Table"
    ;;
esac
```

The first line #!/bin/bash specifies the interpreter for the script as bash.

The next line echo "Enter the number: " prompts the user to enter a number.

The following line read n reads the user input and stores it in a variable named n.

The script then uses a case statement to check the value of n against different patterns.

The first pattern *[^0-9]* checks if n is not a number by using a regular expression that matches any input that contains non-numeric characters. If this pattern matches, it prints an error message informing the user that only numbers are accepted as input.

The second pattern [0-9]* matches when n is a number. If this pattern matches, the script runs a loop that prints the multiplication table of n from 1 to 10.

The third pattern * matches when n is an empty input. If this pattern matches, the script prints an error message prompting the user to enter a valid input.

The script ends after the case statement.

Atul's Solution

```
# If no arguments are passed the raise an error.
if [ $# -eq 0 ]; then
  echo "Error, Please enter a argument to generate table"
  exit 1
fi

echo "No of number passed as arguments - $#"
i=1;
for number in "$@"
do
    echo "Argument - $i: $number";
    i=$((i + 1));
done
echo ""
i=1;
j=$#;
while [ $i -le $j ]
do
        n=$1 # Intiliaing the first argument as n
        c=1 # Counter Variable
        echo "Table of $n:"
        # Using while loop to generate the table
        while [ $c -le 10 ] # while counter is less than 10
        do
          result=$(( $n * $c )) # Calculating the product
          echo "$n x $c = $result" # Printing the product
          c=$(( $c + 1 )) # Incrementing the counter
        done
    shift 1;
        i=$((i+1))
        echo ""
done


exit 0
```

if [$# -eq 0]; then: This line checks if the number of arguments passed to the script is equal to zero. If it is, the script prints an error message and exits with status code 1.

echo "No of number passed as arguments - $#": This line prints the number as argument passed to the script.

i=1;: This line initializes a counter variable i to 1.

for number in "$@": This line starts a loop that iterates over the arguments passed to the script.

echo "Argument - $i: $number";: This line prints the current argument number and its value.

i=$((i + 1));: This line increments the counter variable.

while [$i -le $j]: This line starts a loop that iterates over the arguments passed to the script.

n=$1: This line assigns the first argument to the variable n.

c=1: This line initializes a counter variable c to 1.

echo "Table of $n:": This line prints the header for the multiplication table.

while [$c -le 10]: This line starts a loop that generates the multiplication table.

result=$(($n * $c)): This line calculates the product of n and c.

Sarthak’s solution

```
#Question 2
echo "Enter the number -"
#Taking user input
read n

# Cheking through case statement
case $n in
# To check if it is a valid number or not
*[^0-9]*)
        echo "Please provide avalid input"
        ;;
# Regex to check for number
[0-9]*)
i=1
while [ $i -le 10 ]
do
res=`expr $i \* $n`

echo "$n * $i = $res"
((++i))
done
;;
# condtion to check for no input
*)
        echo "Error !Please provide input"
        ;;
esac
```

This script is similar to mine.

Question 3. Write a Function in bash script to check if the number is prime or not? It should support the following requirement. ● The script should accept the input from the User.

```bash
#Question 3

#Taking user input
echo "Enter a number to check for Prime or not"
read number

#Function to check if a number is prime or not
check_prime(){
  count=0
  num=$1
  for (( i=2 ; i<=$num/2 ;i++ ));
  do
    if [ `expr $num % $i` -eq 0 ]
    then
       count=1
    fi
  done
  if [ $count -eq 1 ] | [ $num -eq 1 ]
  then
     echo "The given number $num is not a prime number "
  else
     echo "The given number $num is a prime number"
  fi
}

#Checking through case statement if a number is valid or not.If not valid then display the error message otherwise call the check_prime function to display whet
case $number in
*[^0-9]*)
     echo "Please Enter a Valid Number!! $number is not a number"
     ;;
[0-9]*)
        echo $(check_prime $number)
        ;;
*)
        echo "Error!! Please Provide Input"
        ;;
esac
```

MY approach

echo "Enter a number to check for Prime or not": This line simply displays a message on the console asking the user to enter a number.

read number: This line reads the user input from the console and assigns it to a variable called number.

check_prime(): This is a function definition. It takes one parameter, num, which is the number to be checked for primality.

count=0: Initializes a variable count to zero. This variable will be used to count the number of factors of the given number.

for ((i=2 ; i<=$num/2 ;i++)): This line starts a loop that will iterate from 2 to half of the given number. Since a number can't have a factor greater than its half, we can optimize the checking process by only checking up to half of the number.

if [\expr $num % $i` -eq 0 ]: This line checks whether the remainder of numdivided byiis equal to zero. If yes, it means thatiis a factor ofnum`.

count=1: This line sets count to 1 to indicate that we have found a factor.

if [$count -eq 1] | [$num -eq 1]: This line checks whether the count variable is equal to 1 or the given number is equal to 1. If either of these conditions is true, then the number is not a prime number.

echo "The given number $num is not a prime number ": This line displays a message on the console indicating that the given number is not a prime number.

else: This line starts the else block of the if-else statement.

echo "The given number $num is a prime number": This line displays a message on the console indicating that the given number is a prime number.

case $number in: This line starts a case statement that checks the validity of the user input.

[^0-9]):: This line checks whether the user input contains any characters other than digits. If yes, it displays an error message indicating that the input is invalid.

echo "Please Enter a Valid Number!! $number is not a number": This line displays an error message on the console indicating that the input is invalid.

[0-9]*):: This line checks whether the user input contains only digits. If yes, it calls the check_prime function to check whether the number is prime or not.

echo $(check_prime $number): This line calls the check_prime function with the given input number and displays the result on the console.

*):: This line catches any other input that is not covered by the previous two cases and displays an error message indicating that the input is invalid.

echo "Error!! Please Provide Input": This line displays an error message on the console indicating that the input is invalid.

Atul and sarthak's scripts have similar approach to mine

Question 4. Create a bash script that supports the following requirement. ● Create a folder 'Assignment'. ● Create a file 'File1.txt' inside 'Assignment' Folder. ● Copy all the content of

Table.sh(2nd script) in ‘File1.txt’ without using ‘cp’ and ‘mv’ command. ● Append the text Welcome to Sigmoid’ to the ‘File1.txt’ file. ● List all the directories and files present inside Desktop Folder.
My Approach

```
#!/bin/bash

#Question 4

# Command for creating the Assignment folder
mkdir Assignment

# Command for creating the file "File1.txt" inside Assignment folder
touch Assignment/File1.txt


#Command for copying the content of table.sh in File1.txt
cat Table.sh > Assignment/File1.txt

# Commands to append the text("Welcome to Sigmoid") in File1.txt
str="Welcome to Sigmoid"
echo $str >> Assignment/File1.txt

# Listing all the directories and files present inside Desktop Folder and appending it to another text file
# ls -al ../ >> DesktopListDirectories.txt
ls -al ~/Desktop >> DesktopListDirectories.txt

echo "Created a file DesktopListDirectories.txt containing all the list and directories present inside Desktop folder"

#Command to open the above created text file containing list of all files and directories present inside Desktop folder
open DesktopListDirectories.txt
```

Atul and sarthak have almost same approach

Atul used for loop where i used while
Sarthak also used while loop.

Question 5. You have given an array. Using Bash script, print its length, maximum element and minimum element. arr=(2 3 4 1 6 7).

```
#!/bin/bash

#Question 5

#Taking the user input for size of an array
echo "Enter the size of array: "
read size


# Taking User Input From shell for an array
#Assigining an empty array
arr=()
for(( i=0; i<size ;i++ ));
do
        echo "Enter $(($i+1)) element"
        read n
        arr[$i]=$n
done

echo "Total Number of elements: ${#arr[@]}"
echo "The array elements are: "
echo ${arr[@]}
```

```
# Function to find maximum and minimum element in a given array
max_min_ele(){
        max=${arr[0]}
        min=${arr[0]}

        for ele in "${arr[@]}";
        do
          if [ $ele -gt $max ]
          then
            max=$ele
          fi
          if [ $ele -lt $min ]
          then
                min=$ele
          fi
        done
        echo "Maximum element in an array: $max"
        echo "Minimum element in an array: $min"
}

#Calling the above function max_min_ele
max_min_ele
```

Explanation

It prompts the user to enter the size of an array using the "echo" command and reads the input value using the "read" command.

It initializes an empty array named "arr" and populates it with user input values using a for loop that iterates "size" times. It prompts the user to enter each array element and reads the input value using the "read" command.

It displays the total number of elements in the array and prints the array elements using the "echo" command.

It defines a function named "max_min_ele" that takes no arguments and finds the maximum and minimum element in the array using a for loop that iterates over the array elements. The maximum and minimum elements are stored in the variables "max" and "min", respectively.

It calls the "max_min_ele" function using the "max_min_ele" command.

The script takes user input for an array and finds the maximum and minimum element in that array using a function. It then prints the maximum and minimum element on the command line.

Atul and sarthak used same approach as mine