My Solution

Question-1:

```
# Answer1
echo Answer 1:
# Display Current date
echo Current date: `date +%F`
# Display Current time
echo Current time: `date +%T`
# Display Username
echo Username: `whoami`
# Display Home directory
echo Home directory: ~
# Display Current working directory
echo Current working directory: `pwd`
```

Explanation:

● Used date command with format code %F with echo for displaying the full date; same as %Y-%m-%d.

● Used date command with format code %T with echo for displaying the time; same as %H:%M:%S.

● Used whoami command with echo for showing the current username.

● Used ~(tilde) with echo for displaying the home directory.

● Used pwd command with echo for displaying the current working directory.

Question-2:

```
# Answer2
# Taking input from the user
echo "Enter the number: "
read n

# Checks and prints message "No input entered" if the user provides no
input
if [[ -z "$n" ]]; then
   echo "No input entered"
   exit 1
fi

# re='^[0-9]+$'
# if ! [[ $n =~ $re ]] ; then
#    echo "error: Not a number";
#    exit 1
# fi

re='^[0-9]+$'
if ! [[ $n =~ $re ]] ; then
      echo "Not a number, invalid input";
      exit 1
fi

# initializing i with 1
i=1

# Looping i, i should be less than
# or equal to 10, i.e., -le stands for less than or equal to
while [ $i -le 10 ]
do
      res=`expr $i \* $n`

      # printing on console
      echo "$n * $i = $res"

      # incrementing i by one
      # i = `expr $i + 1`
      ((++i))

# end of while loop
done
```

Explanation:

● Used echo command prompting the user to enter the number, whose table the user wishes to see.

● Used read command to read user input into a variable n.

● Used -z switch to test if the expansion of "$n" is a null string or not. If it is a null string then the body is executed.

● Defined a string re to used as a regular expression to check whether input is valid number or not.

● re='^[0-9]+$', this regex ensures that the input string starts with a number, a number in between and also ends with a number; disallows any other character.

● Initialized a variable i with 1.

● Used a while loop to iterate over i from i=1 through i=10, -le specifies less than or equal to.

● Used a res variable to store the ith multiple of n.

● Used expr command to evaluate i*n and store the value in res.

● Used echo to display the current multiple of n in the format, n * i = res.

● Incremented the value of i to be used for the next iteration.

● Loops ends when the while condition evaluates to false; i<=10 in this case.

● The 'while loop' statement is closed by 'done' keyword.

Question-3:

```
# Answer3
echo "Enter the number: "
# User input for number
```

```
read n

# Checks and prints message "No input entered" if the user provides no
input
if [[ -z "$n" ]]; then
   echo "No input entered"
   exit 1
fi

re='^[0-9]+$'
if ! [[ $n =~ $re ]] ; then
      echo "Not a number, invalid input";
      exit 1
fi

if [ $n -eq 1 ]; then
      echo "$n is not prime"
      exit 0
fi

# Looping i, from 2 through n/2
for((i=2; i<=$n/2; i++))
do
  ans=$(( n%i ))
  if [ $ans -eq 0 ]
  then
    echo "$n is not a prime number."
    exit 0
  fi
done
echo "$n is a prime number."
```

Explanation:

● Used echo command prompting the user to enter the number, which the user wishes to find out if it is prime or not.

● Used read command to read user input into a variable n.

● Used -z switch to test if the expansion of "$n" is a null string or not. If it is a null string then the body is executed.

● Defined a string re to used as a regular expression to check whether input is valid number or not.

● re='^[0-9]+$', this regex ensures that the input string starts with a number, a number in between and also ends with a number; disallows any other character.

● Placed a check to display the number isn't prime if the value equals 1.

● Used a for loop, initializing i as 2, iterating over i from 2 through n/2.

● Stored remainder of n%i in the variable ans inside the loop.

● If ans equalled 0, displayed the number isn't prime; exits with status of 0, indicating normal, error-free exit.

● Used fi keyword to close the if statement.

● The 'for loop' statement is closed by 'done' keyword.

● Echoes n is a prime number if the loop executed fully without exiting.

Question-4:

```
# Answer4
# making an empty directory, Assignment, in Documents directory
mkdir ~/Documents/Assignment
# making a file, File1.txt, in Assignment directory
touch ~/Documents/Assignment/File1.txt
# moving the contents of the source file Table.sh, to the destination
file File1.txt
cat Table.sh >> ~/Documents/Assignment/File1.txt
# Appending the text "Welcome to Sigmoid" in File1.txt
echo "Welcome to Sigmoid" >> ~/Documents/Assignment/File1.txt
# Displaying all the directories and files present in the Desktop folder
ls -al /Users/ratinderpalsingh/Desktop
```

Explanation:

● Used mkdir command to create a directory in the Documents directory.

● Used touch to create a file File.txt in the Assignment directory.

● Used cat command to redirect the contents of the file Table.sh into File1.txt by giving the absolute file path instead of printing the contents of the file to the standard output.

● Used >> to append the contents instead of writing.

● Used echo with >> to append the string "Welcome to Sigmoid" to the file File1.txt.

● Used ls -al command for displaying all the directories and files present in the Desktop folder; -a, all files and folders, including ones that are hidden and start with a '.'; -l, list all files in long format.

Question-5:

```
# Answer5
echo "Enter an array: "
# User input for the array
read -a arr

# Initialize max variable with arr[0]
max=${arr[0]}
# Initialize min variable with arr[0]
min=${arr[0]}

# Loop through all elements in the array
for i in "${arr[@]}"
do
```

```
    # Update max if applicable
    if [[ "$i" -gt "$max" ]]; then
        max="$i"
    fi

    # Update min if applicable
    if [[ "$i" -lt "$min" ]]; then
        min="$i"
    fi
done

# Output results:
echo "Length of the array is: ${#arr[@]}"
echo "Maximum element is: $max"
echo "Minimum element is: $min"
```

Explanation:

● Used echo command prompting the user to enter an array, whose length, maximum and minimum element the user wishes to see.

● Used read command to read user input as space separated numbers into a variable arr; -a option specifies that the input is assigned to an array.

● Initialized a variable max with first element of the array.

● Initialized a variable min with first element of the array.

● Used a for loop to iterate over the array elements from start to the end; i as the current element of each iteration.

● Used conditional statement checking if i's value is greater than (hence the option -gt) the value in max variable, and updated max if the expression evaluated to true.

● Used conditional statement checking if i's value is lesser than (hence the option -lt) the value in min variable, and updated min if the expression evaluated to true.

● Used fi keyword to close the if statement.

● The 'for loop' statement is closed by 'done' keyword.

● Outputting all results, the length, max element, min element of the array.

● Used echo to display length of the array using ${#arr[@]}; if subscript is @ or *, the word expands to all members of name; by prefixing # to variable we will find length of the array (i.e number of elements).

● Used echo to display maximum element of the array stored in the max variable.

● Used echo to display minimum element of the array stored in the min variable.

Karan’s Solution

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)"
```

Explanation:

He has used the following commands for displaying the required parameters asked in the question.
These commands are the same as mine.

### 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
```

Explanation:

His script takes user input for a number and checks if it is a valid input or not using a case statement. If the input is a number, it generates the multiplication table for that number from 1 to 10. If the input is not a number, it displays an error message. If there is no input, it also displays an error message.

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
whether number is prime or not.If there is no input,it will display
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
```

Explanation:

His code takes user input for a number and checks if it is a valid input or not using a case statement. If the input is a valid number, it calls a function "check_prime" to determine if the number is prime or not. If the input is not a valid number, it displays an error message. If there is no input, it also displays an error message.
The "check_prime" function takes a number as input and checks if it is prime or not by using a for loop to divide the number by all the integers from 2 to half the value of thenumber. If the number is divisible by any of these integers, it is not a prime number. Otherwise, it is a prime number.
His solution is logically the same as mine, just some minor implementations are different.

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
```

Explanation:

His script creates a folder called "Assignment" in the current directory using the "mkdir" command. It then creates a file called "File1.txt" inside the "Assignment" folder using the "touch" command.
The content of the "table.sh" file is then copied to "File1.txt" using the "cat" command.
The text "Welcome to Sigmoid" is then appended to the end of "File1.txt" using the "echo" command.
Then He is listing all the directories and files present inside the Desktop folder and appends the output to a text file called "DesktopListDirectories.txt" using the "ls" command with the "-al" option.

Finally, he is opening the "DesktopListDirectories.txt" file using the "open" command. This command may not work on all systems, as it depends on the operating system and the default file viewer application.

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:

His code prompts the user to enter the size of an array and reads the input using the "read" command. It then creates an empty array and prompts the user to enter each element of the array using a loop. The input is read using the "read" command and added to the array using array indexing.

Then it prints the total number of elements in the array using the "${#arr[@]}" notation, and the array elements using "${arr[@]}".

He defined a function called "max_min_ele" to find the maximum and minimum elements in the array. The function initializes the "max" and "min" variables to the first

element of the array, and then loops over the remaining elements, updating the "max" and "min" variables as necessary. The function then prints the maximum and minimum elements using the "echo" command.Finally,He is calling the "max_min_ele" function to find the maximum and minimum
elements in the array.

Srinivas's Solution

Question-1:

```
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"
```

Question-2:

```
# 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
```

Explanation:

He has verified whether any arguments have been passed, and if not, we issue an error and quit the programme with exit status 1. If not, we run a loop to print all the arguments passed, then another while loop on all the arguments, start a counter internally on the second while loop, increment it after each iteration, fetch the result, and repeat this internal loop for each element in the arguments list until the counter value is less than 10, at which point the programme ends.

Question-3:

```
# Function to check if it is prime or not
function is_prime() {
  num=$1 # Intilizing num by the first argument
  # If num is less than 2 it is not prime
  if [ $num -lt 2 ]; then
    echo "$num is not a prime number."
    return
  fi
  # Loop from 2 to num/2 to verify its divisors
  for (( i=2; i<$((num/2+1)); i++ ))
  do
    if [ $(($num%$i)) -eq 0 ]; then # if found it is not prime numer
      echo "$num is not a prime number."
      return
    fi
  done
  # else it is prime
  echo "$num is a prime number."
}
# Reading input from user and storing in num
read -p "Enter a number: " num
# Passing it to function is_prime
is_prime $num
```

Explanation:

He has created a function called `is prime` to determine whether a number is prime or not. If the number is greater than 2, we indicate that it is not a prime, and if not, we run a loop from `2` to `number/2` to see if any of the above numbers divide the given number. If we

found any numbers, we can conclude that the number is not a prime because it has a divisor other than 1 and itself.
He has receive user input in the main code, store it in a variable, and then call the function while sending the argument as command line arguments.

Question-4:

```
Creating folder using mkdir
mkdir ~/Desktop/Assignment
echo "Created Assignment Folder"
# Creating file using touch
touch ~/Desktop/Assignment/File1.txt
echo "Created File1.txt in Assignment Folder"
# Copying data in q2 to file1 using cat
cat ~/Desktop/projects/commandLine_assignment/Table.sh  >>
~/Desktop/Assignment/File1.txt
echo "Data in Table.sh copied to File1.txt using cat command"
# Appending given text to file1
echo "Welcome to Sigmoid" >> ~/Desktop/Assignment/File1.txt

echo "Folders in the Desktop"
# Printing files and folders in Desktop
ls -la ~/Desktop/

exit 0
```

Explanation:

He has used the following commands to meet the desired requirements and the function of command is as follows.

Command	Function
mkdir ~/Desktop/Assignment	Creating folder using mkdir
touch ~/Desktop/Assignment/File1.txt	Creating file using touch
cat ~/Desktop/Table.sh >> ~/Desktop/Assignment/File1.txt	Copying data in q2 to file1 using cat
echo "Welcome to Sigmoid" >> ~/Desktop/Assignment/File1.txt	Appending given text to file1
ls -la ~/Desktop/	Printing files and folders in Desktop

Question-5:

```
# Declaring the array
arr=( 2 3 4 1 6 7)
echo "Length of the array - "
echo ${#arr[@]}   # Length of the array
# Method - 1
# Using sort function to find the max and min element.
echo "Max and Min of the array using sort function"
IFS=$'\n'
```

```
echo "Maximum in the array - "
echo "${arr[*]}" | sort -nr | head -n1 #Sorting in reverse and fetch the first element
echo "Minimum in the array - "
echo "${arr[*]}" | sort -n | head -n1 # Sorting and fetch the first element
```

Explanation:

He has have declared the array internally in the code, we have used the following commands to fetch the required result

Command	Function
echo ${#arr[@]}	Length of the array

He has made two methods to find the max and min elements in the given array.
In order to discover the maximum and minimum elements in the given array, we utilised the sort function. For the maximum, he sorted the array in reverse order and used the head1 command to collect the first member.

Command	Function
IFS=$'\n'	Internal field separator
echo "${arr[*]}" | sort -nr | head -n1	Sorting in reverse and fetch the first element
echo "${arr[*]}" | sort -n | head -n1	Sorting and fetch the first element