**1 Create a shell script that checks if a file exists in the current directory.**

*#!/bin/bash  
file="example.txt"  
# Check if the file exists  
if [ -e "$file" ]; then  
echo "File exists: $file"  
else  
echo "File not found: $file"  
fi*

**Explanation:**

1. #!/bin/bash: This is the shebang line that specifies the interpreter (/bin/bash) to be used for running the script.
2. file="example.txt": This line defines the variable file and assigns the value "example.txt" to it. You can replace this with the name of the file you want to check for.
3. if [ -e "$file" ]; then: This line starts an if statement. The condition [ -e "$file" ] checks if the file specified by the value of the file variable exists. The -e flag is used to check for file existence.
4. echo "File exists: $file": If the condition is true (i.e., the file exists), this line prints a message indicating that the file exists, along with the file's name.
5. else: If the condition is false (i.e., the file doesn't exist), the script executes the code under the else branch.
6. echo "File not found: $file": This line prints an error message indicating that the specified file was not found, along with the file's name.
7. fi: This line marks the end of the if statement.

### 2 Write a shell script that calculates the sum of integers from 1 to N using a loop.

*#!/bin/bash*

*echo "Enter a number (N):"  
read N  
sum=0  
for (( i=1; i<=$N; i++ )); do  
sum=$((sum + i))  
done  
echo "Sum of integers from 1 to $N is: $sum"*

**Explanation:**  
The script starts by asking you to enter a number (N) using read. This number will determine how many times the loop runs.

1. The variable sum is initialized to 0. This variable will keep track of the sum of integers.
2. The for loop begins with for (( i=1; i<=$N; i++ )). This loop structure is used to repeat a set of actions a certain number of times, in this case, from 1 to the value of N.
3. Inside the loop, these things happen:
   * i=1 sets the loop variable i to 1 at the beginning of each iteration.
   * The loop condition i<=$N checks if i is still less than or equal to the given number N.
   * If the condition is true, the loop body executes.
   * sum=$((sum + i)) calculates the new value of sum by adding the current value of i to it. This adds up the integers from 1 to the current i value.
4. After each iteration, i++ increments the value of i by 1.
5. The loop continues running until the condition i<=$N becomes false (when i becomes greater than N).
6. Once the loop finishes, the script displays the sum of the integers from 1 to the entered number N.

### 3) Create a script that searches for a specific word in a file and counts its occurrences.

*#!/bin/bash*

*echo "Enter the word to search for:"  
read target\_word  
echo "Enter the filename:"  
read filename  
count=$(grep -o -w "$target\_word" "$filename" | wc -l)  
echo "The word '$target\_word' appears $count times in '$filename'."*

**Explanation:**

* echo "Enter the word to search for:": This line displays a message asking the user to enter a word they want to search for in a file.
* read target\_word: This line reads the input provided by the user and stores it in a variable named target\_word.
* echo "Enter the filename:": This line displays a message asking the user to enter the name of the file they want to search in.
* read filename: This line reads the input provided by the user and stores it in a variable named filename.
* count=$(grep -o -w "$target\_word" "$filename" | wc -l): This line does the main work of the script. Let's break it down further:
  + grep -o -w "$target\_word" "$filename": This part of the command searches for occurrences of the target\_word in the specified filename. The options -o and -w ensure that only whole word matches are counted.
  + |: This is a pipe, which takes the output of the previous command and sends it as input to the next command.
  + wc -l: This part of the command uses the wc command to count the number of lines in the input. The option -l specifically counts the lines.
  + The entire command calculates the count of occurrences of the target\_word in the file and assigns that count to the variable coun

### 4 Write a function in a shell script that calculates the factorial of a given number.

*#!/bin/bash  
# Define a function to calculate factorial  
calculate\_factorial() {  
num=$1  
fact=1  
for ((i=1; i<=num; i++)); do  
fact=$((fact \* i))  
done  
echo $fact  
}  
# Prompt the user to enter a number  
echo "Enter a number: "  
read input\_num  
# Call the calculate\_factorial function with the input number  
factorial\_result=$(calculate\_factorial $input\_num)  
# Display the factorial result  
echo "Factorial of $input\_num is: $factorial\_result"*

**Explanation:**

1. The script starts with the shebang line #!/bin/bash to specify the interpreter.
2. calculate\_factorial() is defined as a function. It takes one argument, num, which is the number for which the factorial needs to be calculated.
3. Inside the function, fact is initialized to 1. This variable will store the factorial result.
4. The for loop iterates from 1 to the given number (num). In each iteration, it multiplies the current value of fact by the loop index i.
5. After the loop completes, the fact variable contains the calculated factorial.
6. The script prompts the user to enter a number using read.
7. The calculate\_factorial function is called with the user-provided number, and the result is stored in the variable factorial\_result.
8. Finally, the script displays the calculated factorial result.

### 5 Create a script that checks for and removes duplicate lines in a text file.

*#!/bin/bash  
input\_file="input.txt"  
output\_file="*

*sort "$input\_file" | uniq > "$output\_file"  
echo "Duplicate lines removed successfully."*

**Explanation:**

1. The script starts with a shebang (#!/bin/bash), which indicates that the script should be interpreted using the Bash shell.
2. The input\_file variable is set to the name of the input file containing duplicate lines (change this to your actual input file name).
3. The output\_file variable is set to the name of the output file where the duplicates will be removed (change this to your desired output file name).
4. The script uses the sort command to sort the lines in the input file. Sorting the lines ensures that duplicate lines are grouped together.
5. The sorted lines are then passed through the uniq command, which removes consecutive duplicate lines. The output of this process is redirected to the output file.
6. After the duplicates are removed, the script prints a success message.