NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of CSE(Datascience)

III Semester – 2022-23

Unix Shell Programming

(22CDL38D)

Laboratory Manual

Prepared by: **PROF. SUSHMA B S**, ASSISTANT PROFESSOR, DEPARTMENT OF DATASCIENCE

Introduction to Shell scripting:

Use of Basic UNIX Shell Commands and options related to them:

vi, ls, mkdir, rmdir, cd, cat, touch, file, wc, sort, cut, who, man etc.

Commands related to inode, I/O redirection and piping.

Shell Programming: Shell script exercises based on following:

(i) Interactive shell scripts

(ii) Positional parameters

(iii) Arithmetic

(iv) if-then-fi, if-then- else-fi, nested if-else

(v) Logical operators

(vi) else + if equals elif, case structure

(vii) while, until, for loops, use of break

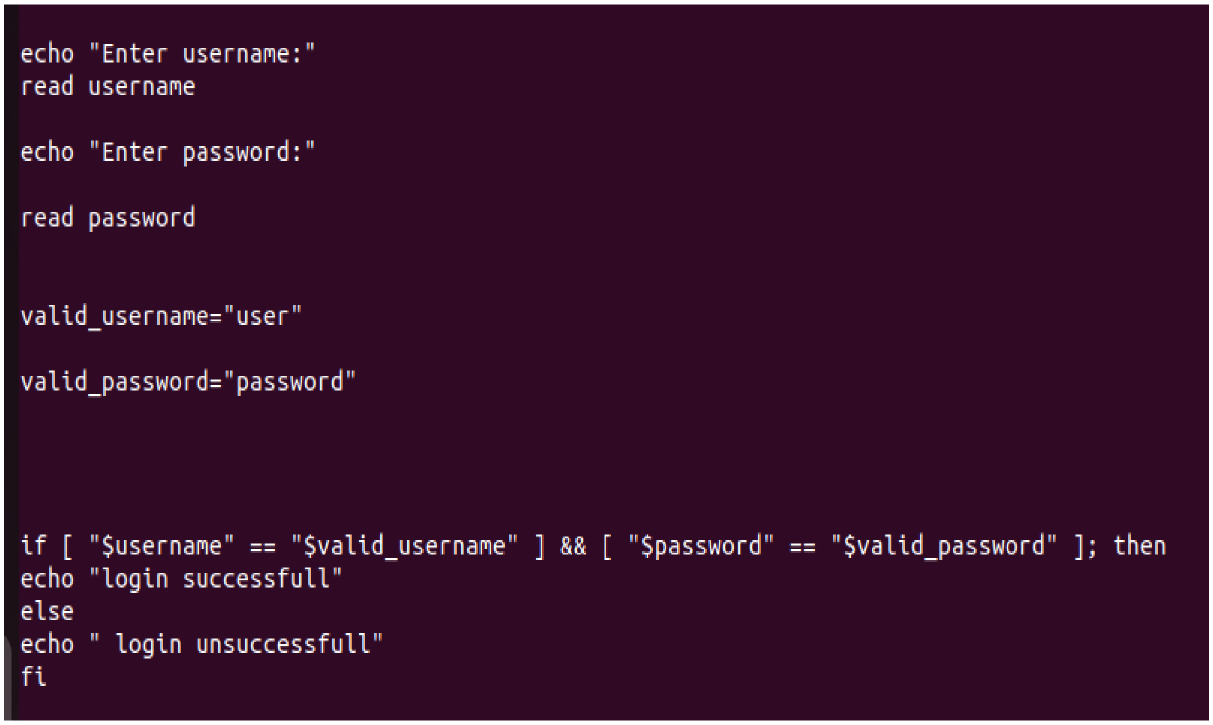
Laboratory Programs:

1. **Write a shell script to check whether the entered username and password is valid or not.**

- echo "Enter username:": This line displays the message "Enter username:" to prompt the user to enter their username.
- read username: This line reads the input provided by the user after the prompt and assigns it to the variable username.
- echo "Enter password:": Similar to the previous line, this displays the message "Enter password:" to prompt the user to enter their password.
- read password: This line reads the input provided by the user after the password prompt and assigns it to the variable password.
- valid_username="user": This line assigns the valid username (in this case, "user") to the variable valid_username.
- valid_password="password": This line assigns the valid password (in this case, "password") to the variable valid_password.

- if ["$username" == "$valid_username"] && ["$password" == "$valid_password"]; then: This line starts an if statement that checks if the entered username is equal to the valid_username AND if the entered password is equal to the valid_password.
- echo "Login successful": If the conditions in the if statement are met (i.e., correct username and password), this line prints "Login successful" indicating that the login was successful.
- else: If the conditions in the if statement are not met (i.e., incorrect username or password), the execution moves to this line.
- echo "Login unsuccessful": This line prints "Login unsuccessful" indicating that the login attempt failed.
- fi: This line marks the end of the if statement.

Overall, the script prompts the user to enter a username and password, compares them to predefined valid credentials, and outputs whether the login was successful or not based on the comparison result.



```
echo "Enter username:"
read username

echo "Enter password:"

read password


valid_username="user"

valid_password="password"




if [ "$username" == "$valid_username" ] && [ "$password" == "$valid_password" ]; then
echo "login successfull"
else
echo " login unsuccessfull"
fi
```

You can save this script in a file, for example, `check_credentials.sh`, make it executable using the `chmod +x check_credentials.sh` command, and then run it using `./check_credentials.sh` in the terminal.

2. Write a shell script to add, subtract, multiply, divide two numbers and add two strings.

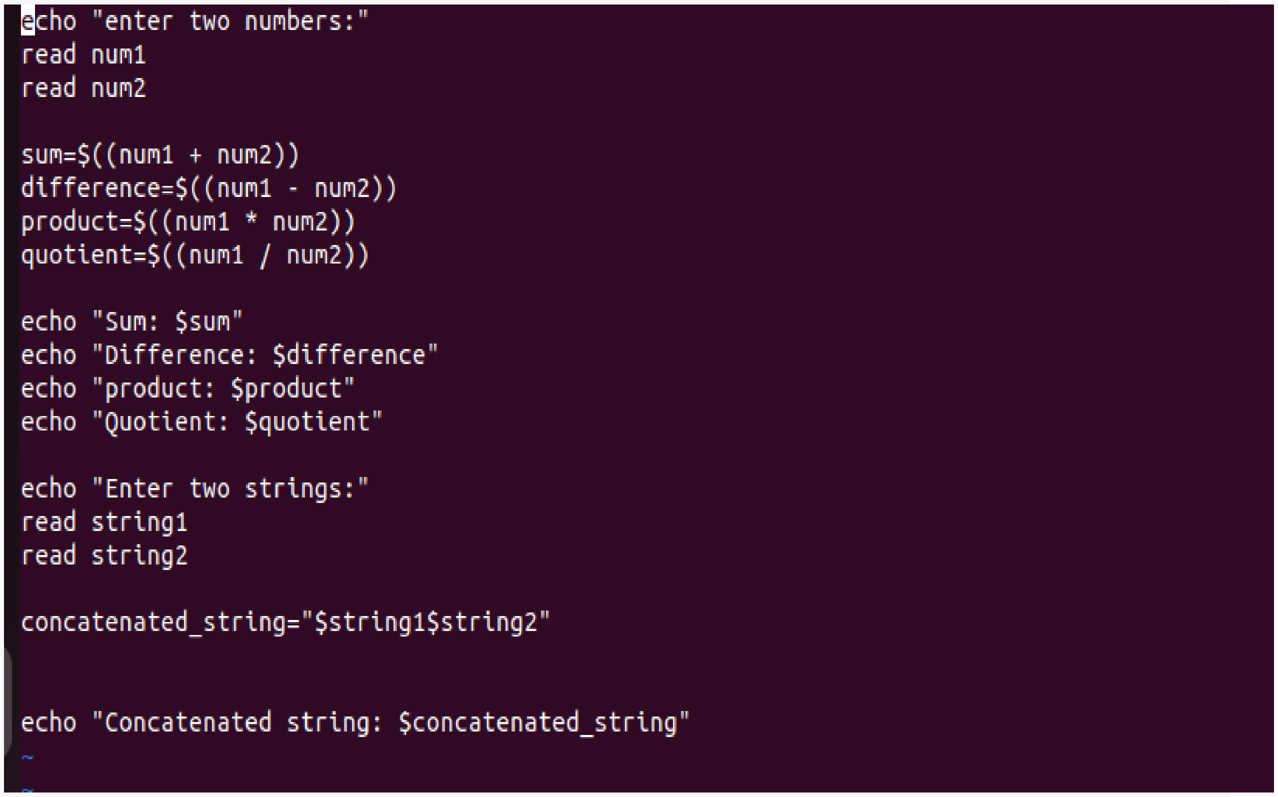
- echo "Enter two numbers:": This line prints a message to the console asking the user to enter two numbers.

- read num1: This line reads the first number input by the user and assigns it to the variable num1.
- read num2: This line reads the second number input by the user and assigns it to the variable num2.
- Arithmetic operations:
    - sum=$((num1 + num2)): Calculates the sum of num1 and num2 and stores the result in the variable sum.
    - difference=$((num1 - num2)): Calculates the difference between num1 and num2 and stores the result in the variable difference.
    - product=$((num1 * num2)): Calculates the product of num1 and num2 and stores the result in the variable product.
    - quotient=$((num1 / num2)): Calculates the quotient of num1 divided by num2 and stores the result in the variable quotient.

- Printing results:
    - echo "Sum: $sum": Prints the sum of the two numbers.
    - echo "Difference: $difference": Prints the difference between the two numbers.
    - echo "Product: $product": Prints the product of the two numbers.
    - echo "Quotient: $quotient": Prints the quotient of the two numbers.

- echo "Enter two strings:": This line prints a message to the console asking the user to enter two strings.
- read string1: This line reads the first string input by the user and assigns it to the variable string1.
- read string2: This line reads the second string input by the user and assigns it to the variable string2.
- Concatenating strings:
    - concatenated_string="$string1$string2": Concatenates string1 and string2 and stores the result in the variable concatenated_string.

- echo "Concatenated string: $concatenated_string": Prints the concatenated string.



```
echo "enter two numbers:"
read num1
read num2

sum=$((num1 + num2))
difference=$((num1 - num2))
product=$((num1 * num2))
quotient=$((num1 / num2))

echo "Sum: $sum"
echo "Difference: $difference"
echo "product: $product"
echo "Quotient: $quotient"

echo "Enter two strings:"
read string1
read string2

concatenated_string="$string1$string2"


echo "Concatenated string: $concatenated_string"
~
~
```

You can save this script in a file, for example, **arithmetic_operations.sh**, make it executable using the **chmod +x arithmetic_operations.sh** command, and then run it using **./arithmetic_operations.sh** in the terminal.

3. Write a linux shell program to perform convert lowercase to uppercase using tr statement.

Certainly! Here's an example of a Linux shell program that converts lowercase characters to uppercase using the `tr` command:

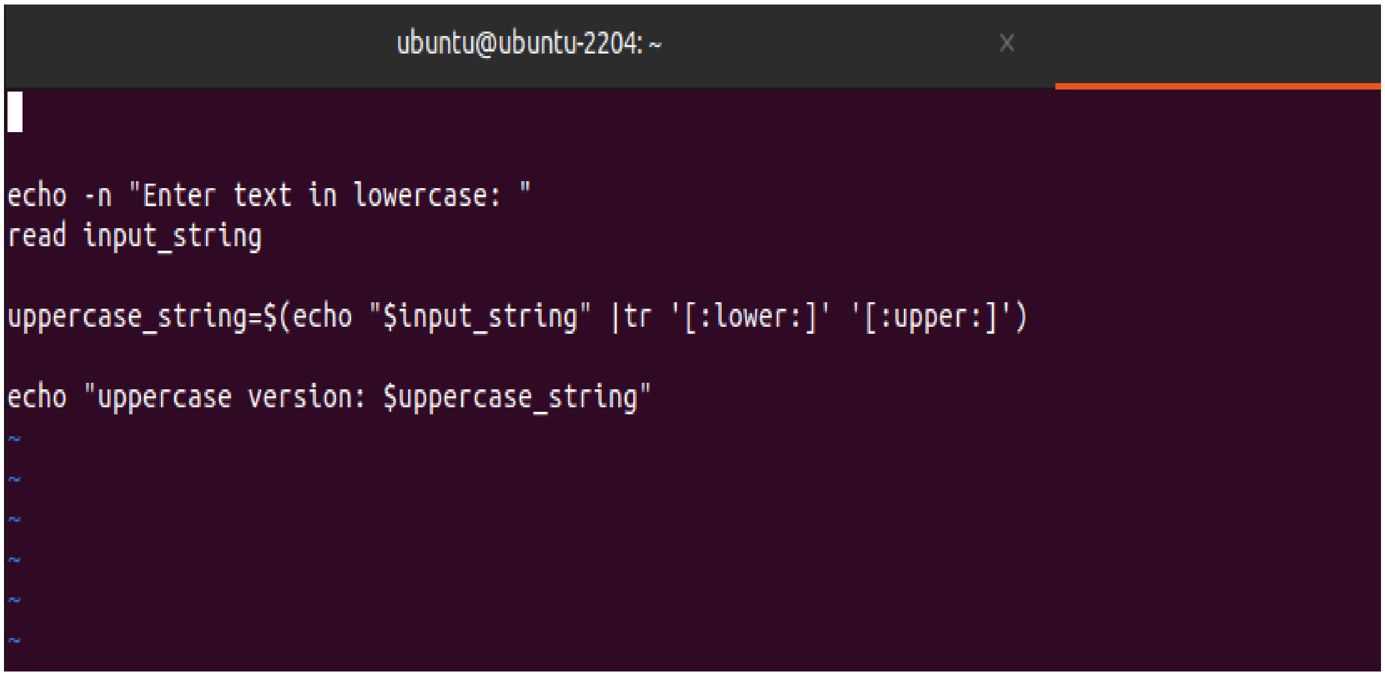
1. echo -n "Enter text in lowercase: ": This line prompts the user to enter text in lowercase. The -n option is used to prevent a newline character from being appended to the prompt.

2. read input_string: This line reads the input from the user and stores it in the variable input_string.

3. uppercase_string=$(echo "$input_string" | tr '[:lower:]' '[:upper:]'): This line converts the input string to uppercase using the tr command. The tr command is used to translate or delete characters. Here, it translates lowercase characters to uppercase characters. The result is stored in the variable uppercase_string.

4. echo "Uppercase version: $uppercase_string": This line prints the uppercase version of the input string. The $uppercase_string variable contains the transformed text, and it is printed along with the message "Uppercase version: ".

Overall, the script takes a user input in lowercase, converts it to uppercase, and then prints the uppercase version of the input string



```
echo -n "Enter text in lowercase: "
read input_string

uppercase_string=$(echo "$input_string" |tr '[:lower:]' '[:upper:]')

echo "uppercase version: $uppercase_string"
~
~
~
~
~
~
```

You can save this script in a file, for example, `convert_lowercase_to_uppercase.sh`, make it executable using the `chmod +x convert_lowercase_to_uppercase.sh` command, and then run it using `./convert_lowercase_to_uppercase.sh` in the terminal.

4. Write a shell script to check the given file is a directory or not.

Certainly! Here's an example of a shell script that checks whether a given file is a directory or not:

1. The script starts by printing the message "Enter the path:" using the echo command.

2. It then uses the read command to take user input and store it in the variable file_path.

3. The script checks whether the entered path is a directory or not using the if statement.

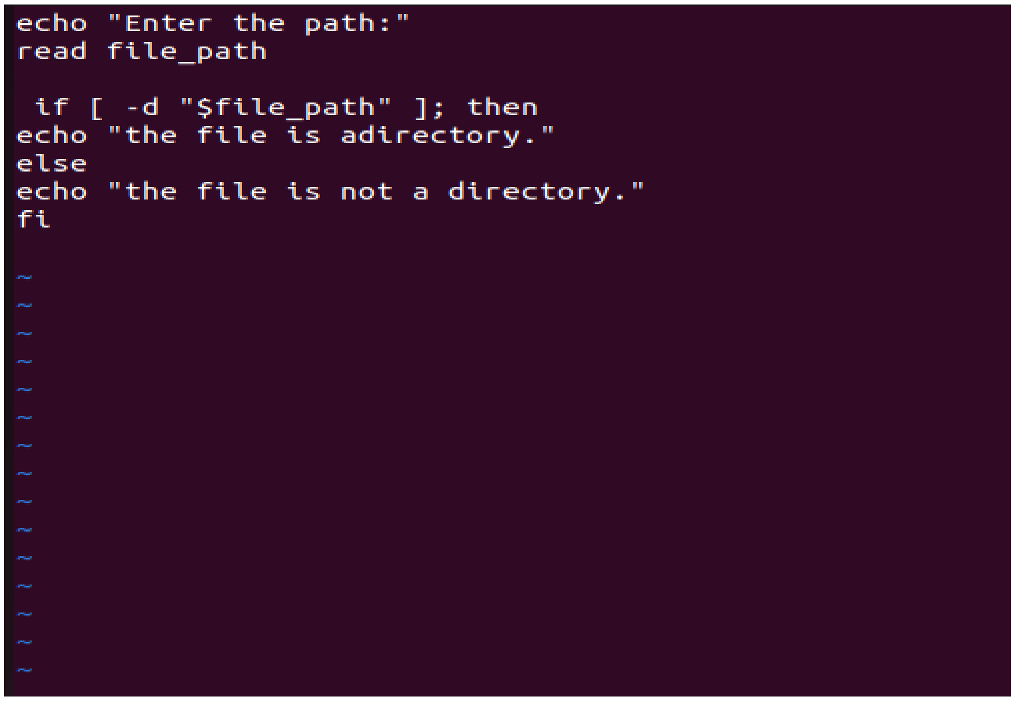
4. Inside the if statement, -d "$file_path" is a condition that checks if the variable file_path corresponds to a directory. The -d flag is used to test if the given path exists and is a directory.

5. If the condition is true (i.e., the entered path is a directory), the script prints "the file is a directory." using echo.

6. If the condition is false (i.e., the entered path is not a directory), the script prints "the file is not a directory." using echo.

7. The fi statement marks the end of the if block, closing it.

Overall, the script allows users to input a file path and then informs them whether the entered path corresponds to a directory or not



```
echo "Enter the path:"
read file_path

 if [ -d "$file_path" ]; then
echo "the file is adirectory."
else
echo "the file is not a directory."
fi

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

You can save this script in a file, for example, `check_directory.sh`, make it executable using the `chmod +x check_directory.sh` command, and then run it using `./check_directory.sh` in the terminal.

5. Write a shell script to perform check pattern matching using grep command.

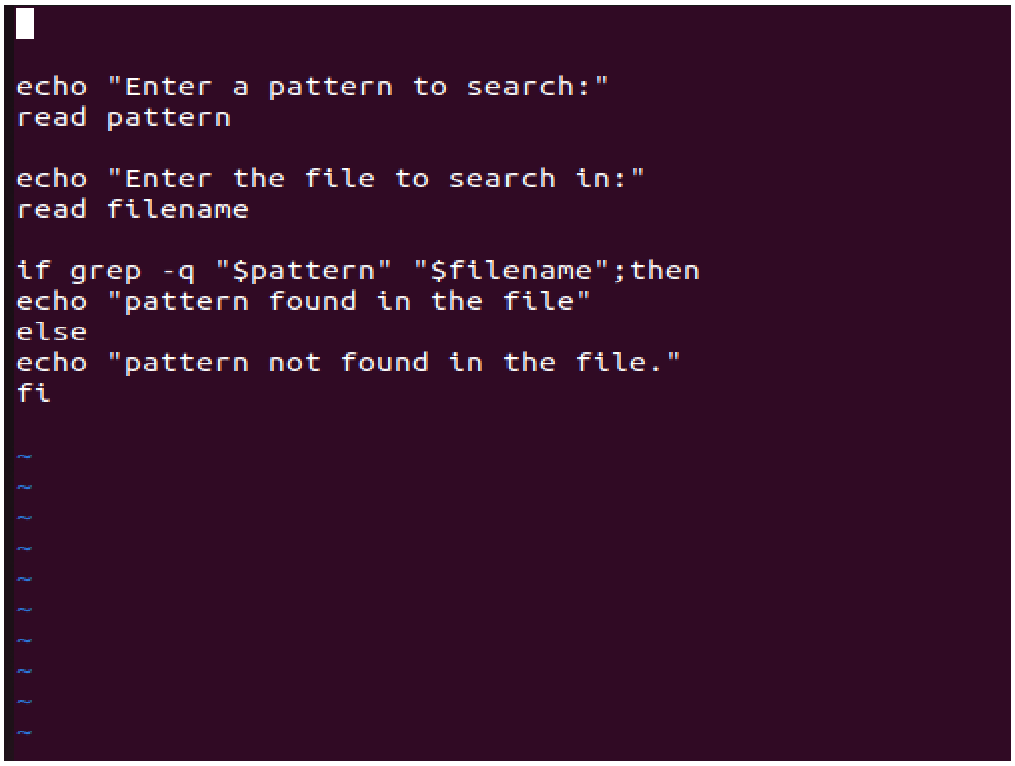
Certainly! This script is a simple bash script that allows a user to search for a specific pattern within a given file.

- echo "Enter a pattern to search:": This line displays the message prompting the user to enter the pattern they want to search for.
- read pattern: This line reads the input provided by the user and assigns it to the variable pattern.

- echo "Enter the file to search in:": This line prompts the user to enter the name of the file they want to search within.

- read filename: This line reads the input provided by the user and assigns it to the variable filename.
- if grep -q "$pattern" "$filename"; then: This line uses the grep command to search for the specified pattern within the given file. The -q option tells grep to perform the search quietly, without displaying any output. If the pattern is found, grep exits successfully, and the then block of the if statement is executed.
- echo "Pattern found in the file.": If the pattern is found, this line displays a message indicating that the pattern was found in the file.
- else: If the pattern is not found in the file, the execution proceeds to this else block.
- echo "Pattern not found in the file.": This line displays a message indicating that the pattern was not found in the file.
- fi: This line marks the end of the if statement.

Overall, this script provides a simple way for users to search for a pattern within a specified file using bash scripting and the grep command



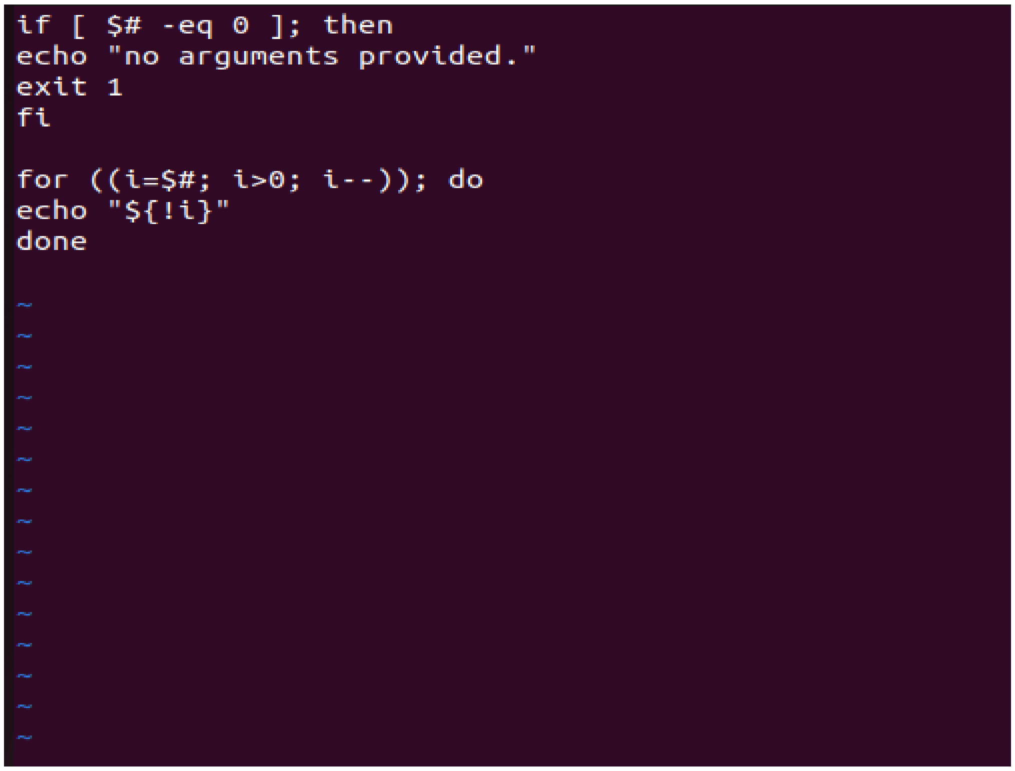
```

echo "Enter a pattern to search:"
read pattern

echo "Enter the file to search in:"
read filename

if grep -q "$pattern" "$filename";then
echo "pattern found in the file"
else
echo "pattern not found in the file."
fi

~
~
~
~
~
~
~
~
~
~
```

You can save this script in a file, for example, `pattern_matching.sh`, make it executable using the `chmod +x pattern_matching.sh` command, and then run it using `./pattern_matching.sh` in the terminal.

6. Write a non-recursive shell script that accepts any number of arguments and prints them in a reverse order.

Certainly! Here's an example of a non-recursive shell script that accepts any number of arguments and prints them in reverse order:

- if [$# -eq 0]; then: This line checks if the number of arguments provided to the script is equal to zero ($# gives the number of arguments). If there are no arguments, it proceeds to the next line.
- echo "no arguments provided.": If there are no arguments, this line prints out the message "no arguments provided."
- exit 1: After printing the message, the script exits with a status code of 1, indicating an error condition.
- fi: Ends the if statement.
- for ((i=$#; i>0; i--)); do: This line initiates a for loop that iterates from the number of arguments down to 1. It starts by setting the variable i to the number of arguments ($#), then decrements i by 1 in each iteration until it reaches 1.
- echo "${!i}": Within the loop, this line prints out the argument corresponding to the current value of i. ${!i} is a way to access the argument positionally, where i represents the position of the argument.
- done: Ends the for loop.



```
if [ $# -eq 0 ]; then
echo "no arguments provided."
exit 1
fi

for ((i=$#; i>0; i--)); do
echo "${!i}"
done

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

You can save this script in a file, for example, `reverse_arguments.sh`, make it executable using the `chmod +x reverse_arguments.sh` command, and then run it using `./reverse_arguments.sh arg1 arg2 arg3 ...` in the terminal, providing any number of arguments you want to reverse and print.

7. Write a shell script to check whether the given year is Leap year or not.

Certainly! Here's an example of a shell script that checks whether a given year is a leap year or not:

1. The script starts by prompting the user to enter a year using the echo command.

2. It then reads the input year from the user using the read command and stores it in the variable year.

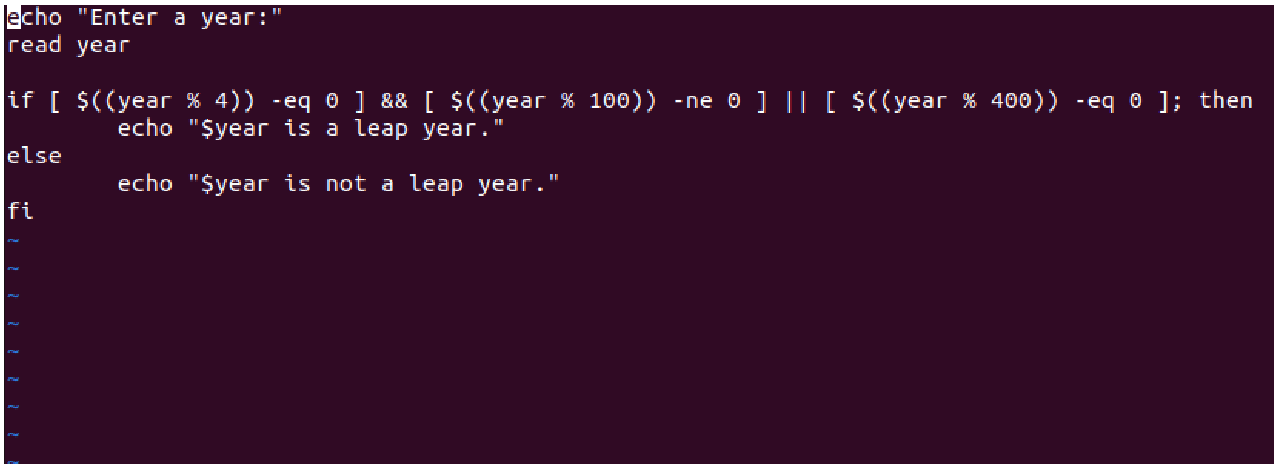
3. Next, the script checks if the entered year is a leap year using an if statement.

4. Inside the if statement, it uses the modulo operator (%) to check if the year is divisible by 4 ($((year % 4))) and not divisible by 100 ($((year % 100)) -ne 0), or if it's divisible by 400 ($((year % 400)) -eq 0).

5. If any of these conditions are true, it means the year is a leap year, and it prints a message indicating so using the echo command.

6. If none of the conditions are true, it means the year is not a leap year, and it prints a message indicating so.

7. The script ends with fi, which marks the end of the if statement.



```
echo "Enter a year:"
read year

if [ $((year % 4)) -eq 0 ] && [ $((year % 100)) -ne 0 ] || [ $((year % 400)) -eq 0 ]; then
        echo "$year is a leap year."
else
        echo "$year is not a leap year."
fi
~
~
~
~
~
~
~
~
```

You can save this script in a file, for example, `leap_year_check.sh`, make it executable using the `chmod +x leap_year_check.sh` command, and then run it using `./leap_year_check.sh` in the terminal

8. Write a shell script to compute GCD & LCM of two numbers.

Certainly! Here's an example of a shell script that computes the GCD (Greatest Common Divisor) and LCM (Least Common Multiple) of two numbers, as well as a basic function to find the GCD and LCM of multiple numbers:

1. The script starts by prompting the user to enter the first number (n1) and then reads the input.

2. Next, it prompts the user to enter the second number (n2) and reads this input as well.

3. Two variables m and n are assigned the values of n1 and n2 respectively. These variables will be used later to print the original numbers in the final output.

4. The variable r is assigned the value of n2. This variable will be used to store the remainder during the GCD calculation.

5. The while loop begins. It continues iterating until r becomes zero, which is the termination condition for the Euclidean algorithm.

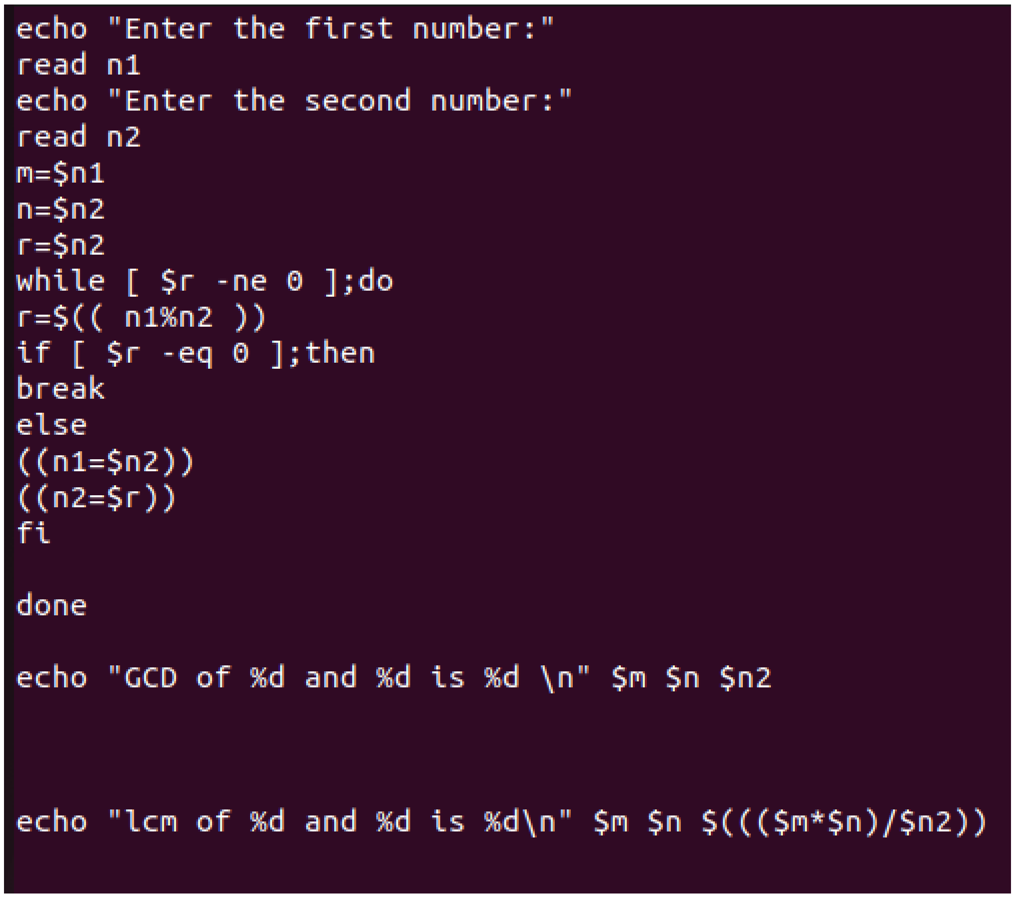
6. Within the loop:

- The remainder r is calculated using the modulo operator % between n1 and n2.
- If r equals 0, indicating that n2 is the GCD, the loop breaks.
- Otherwise, the values are shifted in a way that n2 becomes n1 and r becomes n2, so the loop can continue with the next iteration.

7. After the loop terminates, the script prints out the GCD and LCM of the original numbers (m and n) along with the calculated GCD (n2) using echo statements.

8. The LCM is calculated using the formula: LCM(a, b) = (a * b) / GCD(a, b).

That's the breakdown of how the script works to find the GCD and LCM of two numbers using the Euclidean algorithm.



```
echo "Enter the first number:"
read n1
echo "Enter the second number:"
read n2
m=$n1
n=$n2
r=$n2
while [ $r -ne 0 ];do
r=$(( n1%n2 ))
if [ $r -eq 0 ];then
break
else
((n1=$n2))
((n2=$r))
fi

done

echo "GCD of %d and %d is %d \n" $m $n $n2



echo "lcm of %d and %d is %d\n" $m $n $((($m*$n)/$n2))
```

You can save this script in a file, for example, `gcd.sh`, make it executable using the `chmod +x gcd.sh` command, and then run it using `./gcd.sh` in the terminal

9. Write a shell script to find whether a given number is prime.

1. echo "enter a number:": This line prompts the user to enter a number.

2. read number: This line reads the input from the user and stores it in the variable number.

3. is_prime=true: This initializes a boolean variable is_prime to true. This variable will be used to track whether the input number is prime or not.

4. if [$number -lt 2]; then ... fi: This condition checks if the entered number is less than 2. If it is, the variable is_prime is set to false, as numbers less than 2 are not prime.

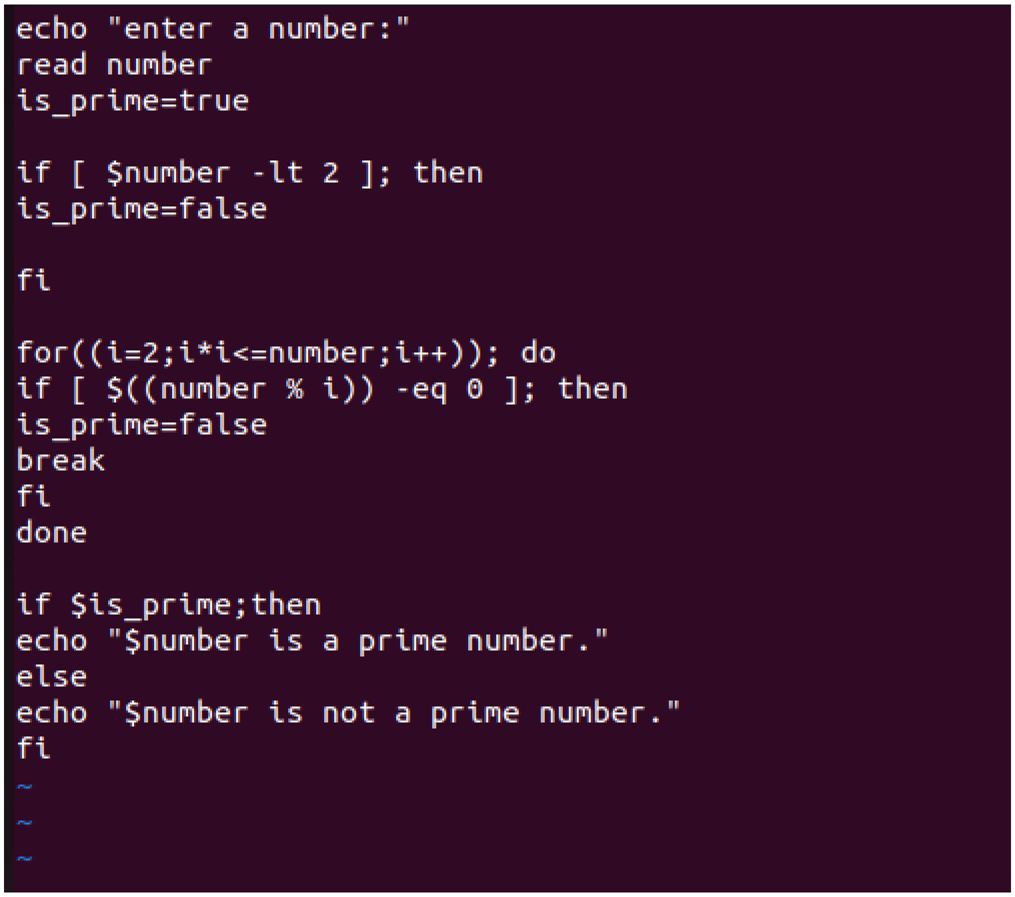
5. for((i=2;i*i<=number;i++)); do ... done: This is a for loop that iterates from 2 to the square root of the input number (i*i<=number). It checks if the input number is divisible by any number in this range.

6. if [$((number % i)) -eq 0]; then ... fi: Within the loop, this condition checks if the input number is divisible evenly (% denotes modulo operation, which gives the remainder) by the current value of i. If it is, the variable is_prime is set to false, indicating that the number is not prime.

7. break: If a divisor is found, the loop breaks because there's no need to continue checking further.

8. if $is_prime; then ... fi: Finally, this condition checks the value of the is_prime variable. If it's true, it means the input number has not been determined to be divisible by any numbers other than 1 and itself, so it's prime. Otherwise, it's not prime.

9. Depending on whether the number is prime or not, appropriate messages are printed using echo



```
echo "enter a number:"
read number
is_prime=true

if [ $number -lt 2 ]; then
is_prime=false

fi

for((i=2;i*i<=number;i++)); do
if [ $((number % i)) -eq 0 ]; then
is_prime=false
break
fi
done

if $is_prime;then
echo "$number is a prime number."
else
echo "$number is not a prime number."
fi
~
~
~
```

You can save this script in a file, for example, `check_prime.sh`, make it executable using the `chmod +x check_prime.sh` command, and then run it using `./check_prime.sh` in the terminal.

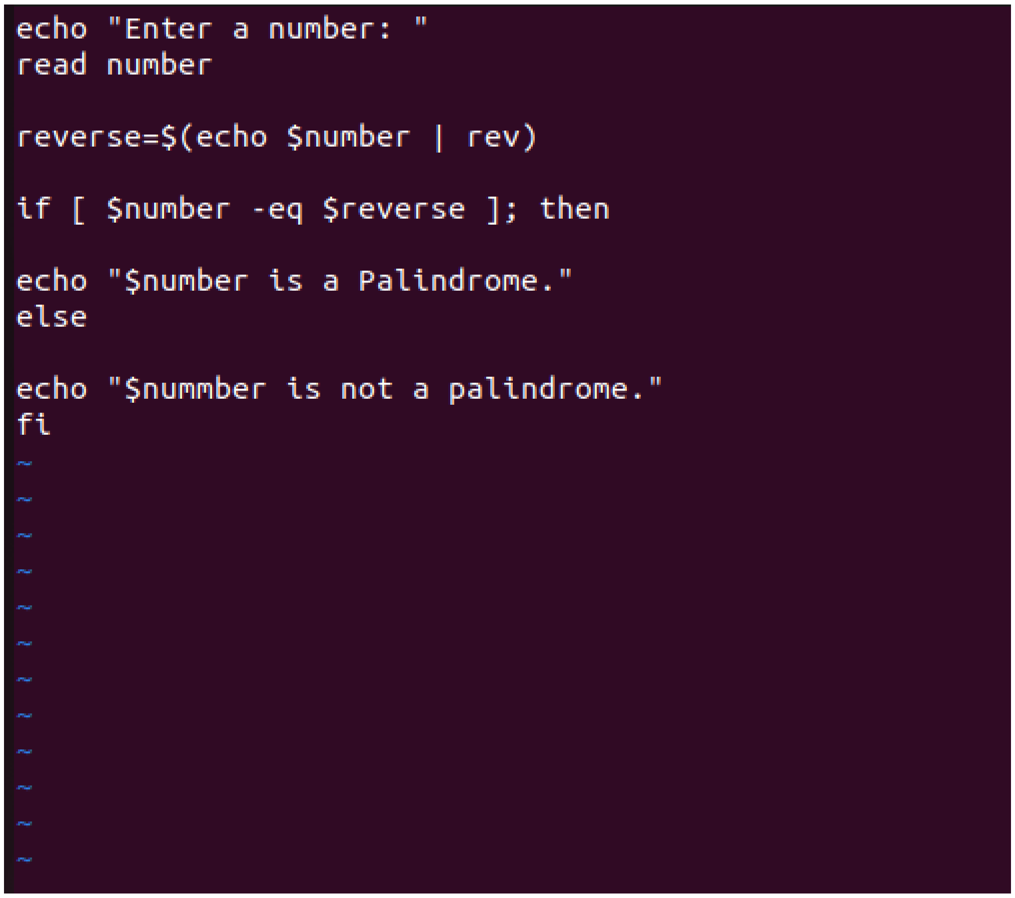
10. Write a shell script to check whether the given string is palindrome or not.

Certainly! Here's an example of a shell script that checks whether a given string is a palindrome or not.

- echo "Enter a number:": This line simply displays the message "Enter a number:" on the terminal.
- read number: This line reads input from the user and stores it in the variable number.
- reverse=$(echo $number | rev): This line reverses the input number using the rev command and stores the result in the variable reverse.
- if ["$number" -eq "$reverse"]; then: This line starts an if statement. It checks if the original number ($number) is equal to its reverse ($reverse). The -eq operator is used for numerical comparison.

- echo "$number is a palindrome.": If the condition in the if statement is true, meaning the number is equal to its reverse, this line is executed, printing a message indicating that the number is a palindrome.
- else: If the condition in the if statement is false, meaning the number is not equal to its reverse, this line is executed, indicating the beginning of the code block to be executed when the condition is false.
- echo "$number is not a palindrome.": Inside the else block, this line is executed, printing a message indicating that the number is not a palindrome.
- fi: This line marks the end of the if statement.

So, the overall purpose of the script is to read a number from the user, check if it is a palindrome (i.e., the number remains the same when read forwards and backwards), and then print an appropriate message indicating whether it's a palindrome or not.



```
echo "Enter a number: "
read number

reverse=$(echo $number | rev)

if [ $number -eq $reverse ]; then

echo "$number is a Palindrome."
else

echo "$nummber is not a palindrome."
fi
~
~
~
~
~
~
~
~
~
~
~
~
```

You can save this script in a file, for example, `check_palindrome.sh`, make it executable using the `chmod +x check_palindrome.sh` command, and then run it using `./check_palindrome.sh` in the terminal.

_______________________________ $ $ $ ___________________________________