EXERCISES

Write a script that takes a filename and 3 keywords. It should grep in the file for all 3 keywords and display for each keyword the number of matches followed by the line numbers where the matches did occur.

• No other output on stdout should be produced by the script

• If the file cannot be read the script should exit with a return code 1, else with code 0 (see help exit if you do not know the exit command)

• Count the number of characters excluding comments

(\*SOLUTION\*)

#!/bin/bash

# Check for command-line arguments

if [ "$#" -ne 4 ]; then

echo "Error: File name and 3 keywords are required."

echo "Usage: ./grep\_keywords.sh <filename> <keyword1> <keyword2> <keyword3>"

exit 1

fi

# Retrieve the command-line arguments

filename=$1

keyword1=$2

keyword2=$3

keyword3=$4

# Check if the file exists and can be read

if [ ! -r "$filename" ]; then

echo "Error: File '$filename' cannot be read."

exit 1

fi

# Grep for the keywords and display results

matches\_keyword1=$(grep -n "$keyword1" "$filename" | wc -l)

matches\_keyword2=$(grep -n "$keyword2" "$filename" | wc -l)

matches\_keyword3=$(grep -n "$keyword3" "$filename" | wc -l)

echo "Matches for '$keyword1': $matches\_keyword1"

echo "Line numbers:"

grep -n "$keyword1" "$filename" | cut -d ':' -f 1

echo "Matches for '$keyword2': $matches\_keyword2"

echo "Line numbers:"

grep -n "$keyword2" "$filename" | cut -d ':' -f 1

echo "Matches for '$keyword3': $matches\_keyword3"

echo "Line numbers:"

grep -n "$keyword3" "$filename" | cut -d ':' -f 1

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Write a shell script that takes 3 arguments and prints them in reverse order If -h is entered anywhere a short description should be printed as well.

(\*SOLUTION\*)

#!/bin/bash

# Function to print the short description

print\_description() {

echo "This script takes 3 arguments and prints them in reverse order."

}

# Check for -h option

if [[ "$\*" == \*-h\* ]]; then

print\_description

fi

# Check for number of arguments

if [ "$#" -ne 3 ]; then

echo "Error: 3 arguments are required."

echo "Usage: ./reverse\_order.sh <arg1> <arg2> <arg3>"

exit 1

fi

# Store the arguments in an array

args=("$@")

# Print the arguments in reverse order

echo "Arguments in reverse order:"

for ((i=${#args[@]}-1; i>=0; i--))

do

echo "${args[$i]}"

done

------------------------------------------------------------------------------------------------------------------------------------------

Write a shell script that does the following when given a path as first arg:

• If the path is a file, print whether it is executable and print the file size

• If the path is a directory move (cd) to it

(\*SOLUTION\*)

#!/bin/bash

# Check if argument is provided

if [ -z "$1" ]; then

echo "Error: Path argument is required."

echo "Usage: ./path\_action.sh <path>"

exit 1

fi

# Get the path argument

path="$1"

# Check if the path is a file

if [ -f "$path" ]; then

echo "Path is a file."

if [ -x "$path" ]; then

echo "File is executable."

else

echo "File is not executable."

fi

size=$(stat -c %s "$path")

echo "File size: $size bytes."

exit 0

fi

# Check if the path is a directory

if [ -d "$path" ]; then

echo "Path is a directory."

cd "$path" || exit 1

echo "Moved to directory: $(pwd)"

exit 0

fi

# If the path is neither a file nor a directory

echo "Error: Path is neither a file nor a directory."

exit 1

-----------------------------------------------------------------------------------------------------------------------------------------

Write a script that takes two integer values as args, I and J. The script should:

• create directories named 1, 2, . . . , I

• Use touch to put empty files named 1 till J in each of these directories

• Print an error if a negative value is provided for I or J

• If any of the files exist, the script should exit with an error.

• Provide help if one of the args is -h, then exit the script.

• If the third argument is a file, the script should copy this file to all locations instead of creating empty files with touch.

(\*SOLUTION\*)

#!/bin/bash

# Function to print help

print\_help() {

echo "This script creates directories and files based on provided arguments."

echo "Usage: ./create\_directories.sh <I> <J> [file]"

echo " - I: The number of directories to create."

echo " - J: The number of files to create in each directory."

echo " - [file]: Optional argument. If provided, this file will be copied to all locations instead of creating empty files."

}

# Check for help option

if [[ "$\*" == \*-h\* ]]; then

print\_help

exit 0

fi

# Check for number of arguments

if [ "$#" -lt 2 ]; then

echo "Error: Insufficient arguments."

echo "Usage: ./create\_directories.sh <I> <J> [file]"

exit 1

fi

# Get the arguments

I=$1

J=$2

file=$3

# Check for negative values

if [ "$I" -lt 0 ] || [ "$J" -lt 0 ]; then

echo "Error: Negative values not allowed."

exit 1

fi

# Create directories

for ((i = 1; i <= I; i++)); do

if [ -d "$i" ]; then

echo "Error: Directory '$i' already exists."

exit 1

fi

mkdir "$i"

done

# Create empty files or copy file

if [ -n "$file" ]; then

if [ -f "$file" ]; then

for ((i = 1; i <= I; i++)); do

cp "$file" "$i/$file"

done

else

echo "Error: File '$file' does not exist."

exit 1

fi

else

for ((i = 1; i <= I; i++)); do

for ((j = 1; j <= J; j++)); do

if [ -e "$i/$j" ]; then

echo "Error: File '$i/$j' already exists."

exit 1

fi

touch "$i/$j"

done

done

fi

echo "Directories and files created successfully."

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Implement the seq command in bash:

• If called with a single argument, print all integers from 1 to this value, i.e. 1 seq 5 should give 1 2 3 4 5 • If called with two arguments, print from the first arg to the second arg, e.g. seq 3 5: 3 4 5 Assume that the first number is always going to be smaller or equal to the second number.

(\*SOLUTION\*)

#!/bin/bash

# Check for number of arguments

if [ "$#" -eq 0 ] || [ "$#" -gt 2 ]; then

echo "Error: Invalid number of arguments."

echo "Usage: ./seq.sh <start> [end]"

exit 1

fi

# Get the start and end values

start=$1

end=$2

# If only one argument is provided, print integers from 1 to the value

if [ "$#" -eq 1 ]; then

for ((i = 1; i <= start; i++)); do

echo "$i"

done

exit 0

fi

# Print integers from start to end

for ((i = start; i <= end; i++)); do

echo "$i"

done

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Write a script that takes the following arguments:

• -h, -q

• --help, --quiet

• -f followed by a filename

• anything else should cause an error message Once the arguments are parsed the script should do the following

• Print help if -h or --help are present, then exit

• Check that the filename provided is a valid file, else throw an error and exit

• Print a nice welcome message, unless --quiet or -q are given

(\*SOLUTION\*)

#!/bin/bash

# Function to print help

print\_help() {

echo "Usage: ./script.sh [-h|--help] [-q|--quiet] -f <filename>"

echo "-h, --help : Print help message and exit"

echo "-q, --quiet : Suppress welcome message"

echo "-f <filename> : Specify a filename"

}

# Parse the arguments

while [[ $# -gt 0 ]]; do

case "$1" in

-h|--help)

print\_help

exit 0

;;

-q|--quiet)

quiet=true

shift

;;

-f)

if [ -z "$2" ]; then

echo "Error: Filename argument is missing."

exit 1

fi

filename="$2"

shift 2

;;

\*)

echo "Error: Invalid argument: $1"

exit 1

;;

esac

done

# Check for filename

if [ -z "$filename" ]; then

echo "Error: Filename is required."

exit 1

fi

# Check if the file exists

if [ ! -f "$filename" ]; then

echo "Error: Invalid filename. File '$filename' does not exist."

exit 1

fi

# Print welcome message if not in quiet mode

if [ "$quiet" != true ]; then

echo "Welcome!"

fi

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Write a script that calculates the cube of N, where N is an integer supplied as the first argument to your script. You should check that N is an integer before entering the routine.

(\*SOLUTION\*)

#!/bin/bash

# Check if N is a valid integer

if ! [[ $1 =~ ^[0-9]+$ ]]; then

echo "Error: N must be a valid integer."

exit 1

fi

# Calculate the cube of N

N=$1

cube=$((N \* N \* N))

echo "The cube of $N is $cube."

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Use bash arithmetic expressions to calculate all primes between 1 and N, where N is a number supplied as the first argument to your script.

(\*SOLUTION\*)

#!/bin/bash

# Check if N is a valid integer

if ! [[ $1 =~ ^[0-9]+$ ]]; then

echo "Error: N must be a valid integer."

exit 1

fi

N=$1

# Function to check if a number is prime

is\_prime() {

local num=$1

if [ "$num" -lt 2 ]; then

return 1

fi

for ((i = 2; i \* i <= num; i++)); do

if [ "$((num % i))" -eq 0 ]; then

return 1

fi

done

return 0

}

# Find and print prime numbers

for ((num = 2; num <= N; num++)); do

if is\_prime "$num"; then

echo "$num"

fi

done

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Write a script that takes either the argument -m or -s, followed by as many numbers as the user wishes. The script should: • Calculate the sum of all numbers if -s is provided • The mean if -m is provided • Give an error if neither -m nor -s are given

(\*SOLUTION\*)

#!/bin/bash

# Function to calculate the sum of numbers

calculate\_sum() {

local sum=0

for number in "$@"; do

sum=$((sum + number))

done

echo "$sum"

}

# Function to calculate the mean of numbers

calculate\_mean() {

local sum=0

local count=0

for number in "$@"; do

sum=$((sum + number))

count=$((count + 1))

done

if [ "$count" -eq 0 ]; then

echo "Error: No numbers provided."

exit 1

fi

local mean=$((sum / count))

echo "$mean"

}

# Check if -m or -s option is provided

if [ "$1" = "-m" ]; then

shift

calculate\_mean "$@"

elif [ "$1" = "-s" ]; then

shift

calculate\_sum "$@"

else

echo "Error: Invalid option. Use -m for mean or -s for sum."

exit 1

fi

exit 0

------------------------------------------------------------------------------------------------------------------------------------------

Give regular expressions that satisfy the following matches does not match chars

a) abbbc, abbc, abc, ac aba 4

regex="^[ab]{1,4}c$"

echo "abbbc" | grep -E "$regex" # Match

echo "abbc" | grep -E "$regex" # Match

echo "abc" | grep -E "$regex" # Match

echo "ac" | grep -E "$regex" # Match

echo "aba" | grep -E "$regex" # No match

b) abbbc, abbc, abc bac, ab 4

regex="ab{1,3}c|bac"

echo "abbbc" | grep -E "$regex" # Match

echo "abbc" | grep -E "$regex" # Match

echo "abc" | grep -E "$regex" # Match

echo "bac" | grep -E "$regex" # Match

echo "ab" | grep -E "$regex" # No match

c) ac, abashc, a123c cbluba, aefg 5

regex="a[^b]{1,4}c"

echo "ac" | grep -E "$regex" # Match

echo "abashc" | grep -E "$regex" # No match

echo "a123c" | grep -E "$regex" # Match

echo "cbluba" | grep -E "$regex" # Match

echo "aefg" | grep -E "$regex" # No match

d) qome, qol , qde eqo, efeq 4

regex="q[ode]{1,2}"

echo "qome" | grep -E "$regex" # Match

echo "qol" | grep -E "$regex" # Match

echo "qde" | grep -E "$regex" # Match

echo "eqo" | grep -E "$regex" # Match

echo "efeq" | grep -E "$regex" # No match

e) arrp, whee bla, kee 4 Note: The art of writing regular expressions is to use the smallest number of characters possible to achieve your goal. The number in the last column gives the number of characters necessary to achieve a possible solution.

regex="[^kw ]{3}"

echo "arrp" | grep -E "$regex" # Match

echo "whee" | grep -E "$regex" # Match

echo "bla" | grep -E "$regex" # Match

echo "kee" | grep -E "$regex" # No match

------------------------------------------------------------------------------------------------------------------------------------------

Write a script, that checks whether a string, provided as an argument to the script, is a palindrome. (A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as madam or “taco cat” or racecar. )

(\*SOLUTION\*)

#!/bin/bash

# Function to check if a string is a palindrome

is\_palindrome() {

local string="$1"

local reversed\_string=$(echo "$string" | rev)

if [ "$string" == "$reversed\_string" ]; then

echo "The string '$string' is a palindrome."

else

echo "The string '$string' is not a palindrome."

fi

}

# Check if argument is provided

if [ -z "$1" ]; then

echo "Error: No argument provided."

exit 1

fi

# Call the function with the argument

is\_palindrome "$1"

------------------------------------------------------------------------------------------------------------------------------------------

Write a script that takes all the files with an extension .txt (in the current directory) and change it to .doc.

(\*SOLUTION\*)

#!/bin/bash

# Check if there are any .txt files in the current directory

txt\_files=$(ls \*.txt 2>/dev/null)

if [ -z "$txt\_files" ]; then

echo "No .txt files found in the current directory."

exit 0

fi

# Rename .txt files to .doc

for file in \*.txt; do

new\_name="${file%.txt}.doc"

mv "$file" "$new\_name"

echo "Renamed file: $file -> $new\_name"

done

echo "All .txt files have been renamed to .doc."

------------------------------------------------------------------------------------------------------------------------------------------

Write a script that calculates whether the year you have provided to a script via „read“ command is a leap year or not.

(\*SOLUTION\*)

#!/bin/bash

# Read the year from user input

echo "Enter a year:"

read year

# Check if the year is divisible by 4 and not divisible by 100, or if it is divisible by 400

if (( year % 4 == 0 && year % 100 != 0 )) || (( year % 400 == 0 )); then

echo "$year is a leap year."

else

echo "$year is not a leap year."

fi

-------------------------------------------------------------------------------------------------------------------------------------

Write a program that asks the user for a number n and prints the sum of the numbers 1 to n. • Modify the previous program (add a function) such that only multiples of three or five are considered in the sum, e.g. 3, 5, 6, 9, 10, 12, 15 for n=17 • Write another function that prints all prime numbers up to number n.

(\*SOLUTION\*)

#!/bin/bash

# Function to calculate the sum of numbers from 1 to n

calculate\_sum() {

local n=$1

local sum=0

for ((i=1; i<=n; i++)); do

sum=$((sum + i))

done

echo "Sum of numbers from 1 to $n: $sum"

}

# Function to calculate the sum of multiples of three or five up to n

calculate\_multiples\_sum() {

local n=$1

local sum=0

for ((i=1; i<=n; i++)); do

if ((i % 3 == 0 || i % 5 == 0)); then

sum=$((sum + i))

fi

done

echo "Sum of multiples of three or five up to $n: $sum"

}

# Function to check if a number is prime

is\_prime() {

local num=$1

if ((num < 2)); then

return 1

fi

for ((i=2; i<=num/2; i++)); do

if ((num % i == 0)); then

return 1

fi

done

return 0

}

# Function to print all prime numbers up to n

print\_primes() {

local n=$1

echo "Prime numbers up to $n:"

for ((i=2; i<=n; i++)); do

if is\_prime "$i"; then

echo "$i"

fi

done

}

# Read the number from user input

echo "Enter a number:"

read num

# Calculate and print the sum of numbers from 1 to n

calculate\_sum "$num"

# Calculate and print the sum of multiples of three or five up to n

calculate\_multiples\_sum "$num"

# Print all prime numbers up to n

print\_primes "$num"

------------------------------------------------------------------------------------------------------------------------------------------

Write a function that prints out the list of the first 100 Fibonacci numbers.

(\*SOLUTION\*)

#!/bin/bash

# Function to print the first 100 Fibonacci numbers

print\_fibonacci\_numbers() {

local n=$1

# Initialize the first two Fibonacci numbers

local fib1=0

local fib2=1

# Print the first two numbers

echo "Fibonacci Numbers:"

echo "$fib1"

echo "$fib2"

# Calculate and print the rest of the numbers

for ((i=3; i<=n; i++)); do

local fib=$((fib1 + fib2))

echo "$fib"

fib1=$fib2

fib2=$fib

done

}

# Call the function to print the first 100 Fibonacci numbers

print\_fibonacci\_numbers 100