3a

#!/bin/bash

echo "Text Files:"

echo "-----------"

for file in \*.txt; do

[ -f "$file" ] && echo "$file"

done

echo " "

echo "C Files:"

echo "--------"

for file in \*.c; do

[ -f "$file" ] && echo "$file"

done

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

3b

#!/bin/bash

# Check if the correct number of arguments are provided

if [ $# -lt 2 ]; then

echo "Usage: $0 <file1> <file2> ... <fileN> <destination\_directory>"

exit 1

fi

# Get the destination directory (the last argument)

destination="${@: -1}"

# Check if the destination directory exists

if [ ! -d "$destination" ]; then

echo "Destination directory does not exist."

exit 1

fi

# Loop through all arguments (except the last one) which are files

for ((i=1; i<=$#-1; i++)); do

file="${!i}" # Get the file at position i

# Check if the file exists

if [ -f "$file" ]; then

mv "$file" "$destination" # Move the file to the destination directory

echo "Moved $file to $destination"

else

echo "File $file does not exist."

fi

done

echo "All specified files have been moved to $destination"

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

4a

#!/bin/bash

# Function to convert time to seconds since epoch

convert\_to\_seconds() {

local time\_string="$1"

date -d "$time\_string" +%s

}

# Check if user provided a time argument

if [ $# -eq 0 ]; then

echo "Usage: $0 <time>"

exit 1

fi

# Convert provided time argument to seconds since epoch

specified\_time=$(convert\_to\_seconds "$1")

# Get list of currently logged in users

logged\_in\_users=$(who | awk '{print $1}')

# Loop through each logged in user

for user in $logged\_in\_users; do

# Get login time of the user

login\_time=$(who | awk -v user="$user" '$1 == user {print $4}')

login\_seconds=$(convert\_to\_seconds "$login\_time")

# Check if user logged in after specified time

if [ "$login\_seconds" -ge "$specified\_time" ]; then

echo "$user"

fi

done

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

4b

#!/bin/bash

# Function to get the time of day

get\_time\_of\_day() {

local hour=$(date +%H)

if [ "$hour" -ge 5 ] && [ "$hour" -lt 12 ]; then

echo "Morning"

elif [ "$hour" -ge 12 ] && [ "$hour" -lt 18 ]; then

echo "Afternoon"

elif [ "$hour" -ge 18 ] && [ "$hour" -lt 21 ]; then

echo "Evening"

else

echo "Night"

fi

}

# Get the current time of day

time\_of\_day=$(get\_time\_of\_day)

# Display the appropriate greeting

echo "Good $time\_of\_day,Welcome Back $USER!"

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

5a

#!/bin/bash

# Check if file name is provided as argument

if [ $# -eq 0 ]; then

echo "Usage: $0 [FILE]"

exit 1

fi

# Check if the file exists

if [ ! -f "$1" ]; then

echo "Error: File '$1' not found."

exit 1

fi

# Read and output the content of the file

while IFS= read -r line; do

echo "$line"

done < "$1"

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

5b

#!/bin/bash

# Check if correct number of arguments are provided

if [ $# -ne 2 ]; then

echo "Usage: $0 <source\_file> <destination\_file>"

exit 1

fi

# Check if source file exists

if [ ! -f "$1" ]; then

echo "Error: Source file '$1' does not exist."

exit 1

fi

# Copy the source file to the destination

cp "$1" "$2"

# Check if copy operation was successful

if [ $? -eq 0 ]; then

echo "File '$1' copied to '$2' successfully."

else

echo "Error: Failed to copy file '$1' to '$2'."

fi

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

6a

#!/bin/bash

# Check if filename is provided

if [ -z "$1" ]; then

echo "Usage: $0 [filename]"

exit 1

fi

# Check if file exists

if [ ! -f "$1" ]; then

echo "File '$1' not found."

exit 1

fi

# Output the first 10 lines of the file

head -n 10 "$1"

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

6b

#!/bin/bash

# Check if filename is provided

if [ -z "$1" ]; then

echo "Usage: $0 filename"

exit 1

fi

filename=$1

# Check if file exists

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

echo "File '$filename' does not exist."

exit 1

fi

# Default number of lines to display

num\_lines=10

# Check if number of lines is provided

if [ ! -z "$2" ]; then

num\_lines=$2

fi

# Display last part of the file

tail -n $num\_lines "$filename"

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

7a

#!/bin/bash

# Check if correct number of arguments are provided

if [ $# -ne 2 ]; then

echo "Usage: $0 <source> <destination>"

exit 1

fi

# Check if source file exists

if [ ! -e "$1" ]; then

echo "Error: Source file/directory '$1' does not exist."

exit 1

fi

# Check if destination directory exists

if [ ! -d "$(dirname "$2")" ]; then

echo "Error: Destination directory '$(dirname "$2")' does not exist."

exit 1

fi

# Check if destination file already exists

if [ -e "$2" ]; then

echo "Warning: Destination file '$2' already exists. Overwrite? (y/n)"

read -r response

if [ "$response" != "y" ]; then

echo "Operation cancelled."

exit 0

fi

fi

# Perform the move operation

mv "$1" "$2"

echo "File/directory '$1' moved to '$2' successfully."

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

7b

#!/bin/bash

if [ $# -eq 0 ]; then

echo "Usage: $0 <filename>"

exit 1

fi

filename="$1"

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

echo "Error: File '$filename' not found."

exit 1

fi

awk '{ print NR, $0 }' "$filename"

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

8

#!/bin/bash

# Function to handle SIGINT

sigint\_handler() {

echo "Received SIGINT"

exit 1

}

# Function to handle SIGQUIT

sigquit\_handler() {

echo "Received SIGQUIT"

exit 1

}

# Function to handle SIGFPE

sigfpe\_handler() {

echo "Received SIGFPE"

exit 1

}

# Register signal handlers

trap 'sigint\_handler' SIGINT

trap 'sigquit\_handler' SIGQUIT

trap 'sigfpe\_handler' SIGFPE

echo "Running. Press Ctrl+C to trigger SIGINT or send SIGQUIT or SIGFPE."

# Infinite loop to keep the script running

while true; do

kill -FPE $$

sleep 1

done

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

9

#!/bin/bash

echo "Parent process $$ is starting..."

# Define a function for the child process

child\_process() {

echo "Child process $BASHPID is starting..."

sleep 5

echo "Child process $BASHPID is done."

}

# Start the child process in the background

child\_process &

# Intentionally wait indefinitely without handling the child process termination

while true; do

sleep 1

done

echo "Parent process $$ is continuing..."

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

10

#!/bin/bash

# Function for the child process

child\_process() {

echo "Child process created..."

echo "Displaying from Child Process"

}

# Main script

echo "Parent Running..."

echo "Displaying parent from Parent process"

# Fork the child process

child\_process &

# Ensure parent process waits for child to finish before exiting

wait

echo "Parent process is done."

âââââââââââââââââââââââ-

11a

send\_message.sh:

#!/bin/bash

FIFO\_FILE="my\_fifo"

# Check if FIFO file exists, if not create it

if [ ! -p "$FIFO\_FILE" ]; then

mkfifo "$FIFO\_FILE"

fi

# Function for writing data into FIFO

write\_to\_fifo() {

while true; do

echo "Enter your message (type 'exit' to quit): "

read message

if [ "$message" == "exit" ]; then

echo "Exiting..."

break

fi

echo "$message" > "$FIFO\_FILE"

done

}

# Run the write\_to\_fifo function

write\_to\_fifo

read\_message.sh:

#!/bin/bash

FIFO\_FILE="my\_fifo"

# Check if FIFO file exists, if not create it

if [ ! -p "$FIFO\_FILE" ]; then

mkfifo "$FIFO\_FILE"

fi

# Function for reading data from FIFO

read\_from\_fifo() {

while true; do

if read line < "$FIFO\_FILE"; then

echo "Received message: $line"

fi

done

}

# Run the read\_from\_fifo function

read\_from\_fifo

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

11b

consumer.sh:

#!/bin/bash

# Ensure the named pipe exists

PIPE="my\_pipe"

if [ ! -p $PIPE ]; then

echo "Named pipe $PIPE does not exist. Please run the producer script first."

exit 1

fi

# Function to consume data

consume\_data() {

echo "Consuming data..."

while read line; do

echo "Received: $line"

done < $PIPE

echo "Consumer finished consuming data."

}

# Main

consume\_data

producer.sh:

#!/bin/bash

# Create a named pipe if it doesn't exist

PIPE="my\_pipe"

if [ ! -p $PIPE ]; then

mkfifo $PIPE

fi

# Function to produce dataproduce\_data()

{

echo "Producing data..."

for i in {1..3}; do

echo "Data $i" > $PIPE

sleep 1 # Simulate some processing time

done

echo "Producer finished producing data."

}

# Main

produce\_data

---------- Forwarded message ---------

From: Sampath Reddy

<sam

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

12

#!/bin/bash

# Define the directory for message queue files

MESSAGE\_QUEUE\_DIR="/tmp/message\_queue"

# Ensure message queue directory exists

mkdir -p "$MESSAGE\_QUEUE\_DIR" || { echo "Error: Unable to create message queue directory"; exit 1; }

# Function to send a message to the queue

send\_message() {

local message="$1"

local file="$(mktemp "$MESSAGE\_QUEUE\_DIR/message\_XXXXXX")"

echo "$message" > "$file" || { echo "Error: Unable to send message"; exit 1; }

}

# Function to receive a message from the queue

receive\_message() {

local oldest\_file=$(ls -t "$MESSAGE\_QUEUE\_DIR" | grep -E '^message\_' | head -n 1)

if [ -n "$oldest\_file" ]; then

local message=$(cat "$MESSAGE\_QUEUE\_DIR/$oldest\_file" | cut -d':' -f2-)

rm "$MESSAGE\_QUEUE\_DIR/$oldest\_file"

echo "$message"

else

echo "No messages in the queue."

fi

}

# Send a couple of messages

send\_message "Hello from Process 1"

send\_message "Hello from Process 2"

# Receive and display messages

echo "Process 1 received: $(receive\_message)"

echo "Process 1 received: $(receive\_message)"

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

13

#!/bin/bash

# Constants

KEY=1234

SIZE=1024

# Function to create shared memory segment

create\_shared\_memory() {

# Create shared memory segment using 'ipcrm' command

ipcrm -M "$KEY" &>/dev/null

ipcmk -M "$KEY" "$SIZE" || { echo "Failed to crea