**Challenge 1**: Write a simple Bash script that prints “Hello DevOps” along with the current date and time.

**Answer:**

**Step 1: Create**hello\_devops.sh & Copy hello\_devops.sh file code there

nano hello\_devops.sh

file code:

#!/bin/bash echo "Hello DevOps! Today's date and time is: $(date)"

**Step 2: Make it executable:**

chmod +x hello\_devops.sh

**Step 3: Run the script:**

./hello\_devops.sh

**Explanation:**

* #!/bin/bash → Shebang to specify Bash as the interpreter.
* echo "Hello DevOps! Today's date and time is: $(date)"
  + echo prints the message.
  + $(date) fetches the current date and time.
  + 

✅ **Challenge 2**: Create a script that checks if a website (e.g., https://www.learnxops.com) is reachable using curl or ping. Print a success or failure message.

**Answer:**

**🛠️ Website Availability Check Script (check\_website.sh)**

#!/bin/bash

# Define the website to check

WEBSITE="https://www.learnxops.com"

DOMAIN="learnxops.com"

# Check website availability using curl

if curl -Is "$WEBSITE" --max-time 5 | head -n 1 | grep -q "200\|301\|302"; then

echo "✅ Success: $WEBSITE is reachable via curl!"

else

echo "⚠️ Curl check failed, trying ping..."

# Check website availability using ping

if ping -c 2 -W 2 "$DOMAIN" > /dev/null 2>&1; then

echo "✅ Success: $DOMAIN is reachable via ping!"

else

echo "❌ Failure: $WEBSITE is not reachable via curl or ping."

fi

fi

**Make it executable:**

chmod +x check\_website.sh

**Run the script:**

./check\_website.sh

**Explanation:**

1. WEBSITE="https://www.learnxops.com" → Defines the website to check.
2. curl -Is "$WEBSITE" --max-time 5 | head -n 1 | grep -q "200\|301\|302"
   * curl -Is → Sends a HEAD request (faster than full request).
   * --max-time 5 → Times out if no response within 5 seconds.
   * grep -q "200\|301\|302" → Checks for HTTP status codes **200 (OK), 301 (Redirect), or 302 (Temporary Redirect)**.
3. If the site is reachable, it prints **"**✅**Success"**, otherwise **"**❌**Failure"**.
4. **Ping Method :**
   * ping -c 2 -W 2 learnxops.com → Sends 2 packets with a 2-second timeout.
   * If the ping succeeds, it prints **Success**, otherwise **Failure**.
   * 

✅ **Challenge 3**: Write a script that takes a filename as an argument, checks if it exists, and prints the content of the file accordingly.

**📄 File Checker Script (check\_file.sh)**

#!/bin/bash

# Check if a filename argument is provided

if [ $# -eq 0 ]; then

echo "❌ Error: No filename provided."

echo "Usage: ./check\_file.sh <filename>"

exit 1

fi

FILENAME="$1"

# Check if the file exists

if [ -f "$FILENAME" ]; then

echo "✅ File '$FILENAME' found. Displaying content:"

cat "$FILENAME"

else

echo "❌ Error: File '$FILENAME' does not exist."

fi

**Explanation:**

1. if [ $# -eq 0 ]; then ... fi
   * Checks if the user provided a filename as an argument.
   * If not, it prints an error and exits.
2. FILENAME="$1" → Stores the first argument as the filename.
3. if [ -f "$FILENAME" ]; then ... fi
   * Checks if the file exists (-f flag for files).
   * If the file exists, it prints **"File found"** and displays its content using cat.
   * Otherwise, it prints **"File does not exist"**.
   * 

✅ **Challenge 4**: Create a script that lists all running processes and writes the output to a file named process\_list.txt.

**Answer:**

**🧠 List Running Processes Script (list\_processes.sh)**

#!/bin/bash

# Define output file

OUTPUT\_FILE="process\_list.txt"

# List all running processes and write to file

ps aux > "$OUTPUT\_FILE"

# Print success message

echo "✅ Process list saved to $OUTPUT\_FILE"

**Explanation:**

1. OUTPUT\_FILE="process\_list.txt" → Defines the output file.
2. ps aux > "$OUTPUT\_FILE"
   * ps aux → Lists all running processes.
   * > → Redirects the output to process\_list.txt.
3. echo "✅ Process list saved to $OUTPUT\_FILE"
   * Prints a confirmation message.
4. 

✅**Challenge 5**: Write a script that installs multiple packages at once (e.g., git, vim, curl). The script should check if each package is already installed before attempting installation.

**Answer:**

**📦 Package Installer Script (install\_packages.sh)**

#!/bin/bash

# Define the list of packages to install

PACKAGES=("git" "vim" "curl")

# Loop through each package and check if it's installed

for PACKAGE in "${PACKAGES[@]}"; do

if dpkg -l | grep -qw "$PACKAGE"; then

echo "✅ $PACKAGE is already installed."

else

echo "⏳ Installing $PACKAGE..."

sudo apt-get install -y "$PACKAGE" && \

echo "✅ $PACKAGE installed successfully." || \

echo "❌ Failed to install $PACKAGE."

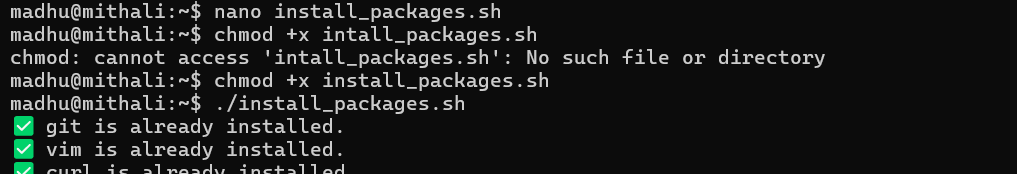
fi

done

**Explanation** :

* **Define the list of packages** in the PACKAGES array.
* **Loop through each package**:
* dpkg -l | grep -qw "$PACKAGE" → Checks if the package is already installed.
* If installed, it prints **"**✅**Already installed."**
* Otherwise, it installs the package using:sudo apt-get install -y "$PACKAGE"
* If the installation succeeds, it prints **"**✅**Installed successfully."**, otherwise **"**❌**Failed to install."**

**Steps to run the script:**

1. **Create the script file and copy above script:**nano install\_packages.sh
2. **Make it executable:**chmod +x install\_packages.sh
3. **Run the Script:**./install\_packages.sh
4. 

✅ **Challenge 6**: Create a script that monitors CPU and memory usage every 5 seconds and logs the results to a file.

**📊 System Resource Monitor Script (monitor\_resources.sh)**

#!/bin/bash

# Define the log file

LOG\_FILE="resource\_usage.log"

echo "Monitoring CPU and Memory usage... Logs will be saved in $LOG\_FILE"

echo "Timestamp | CPU (%) | Memory (%)" > "$LOG\_FILE"

# Infinite loop to log system usage every 5 seconds

while true; do

TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")

# Get CPU usage

CPU\_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')

# Get Memory usage

MEM\_USAGE=$(free | awk '/Mem/ {printf "%.2f", $3/$2 \* 100}')

# Write data to the log file

echo "$TIMESTAMP | $CPU\_USAGE | $MEM\_USAGE" >> "$LOG\_FILE"

# Wait for 5 seconds

sleep 5

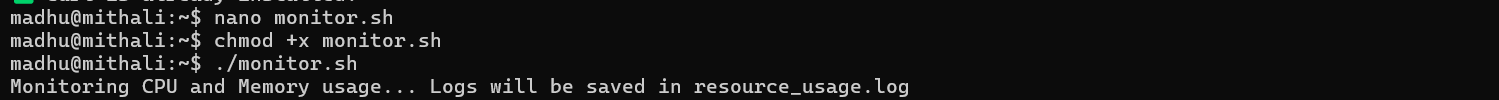
done

**Explanation:**

* **Defines the log file** (resource\_usage.log).
* **Logs column headers** (Timestamp | CPU (%) | Memory (%)).
* **Infinite loop (**while true**)**:
  + Fetches **current timestamp** (date command).
  + Fetches **CPU usage** using top and awk.
  + Fetches **Memory usage** using free and awk.
  + Appends the data to resource\_usage.log.
  + **Waits for 5 seconds** before repeating.

**Steps to Run the Script:**

1. Create a script and copy the above script there:**nano monitor\_resources.sh**
2. Make it executable:**chmod +x monitor\_resources.sh**
3. Run the script in the background:**nohup ./monitor\_resources.sh &**nohup → Keeps the script running even if you log out.& → Runs it in the background.



✅ **Challenge 7**: Write a script that automatically deletes log files older than 7 days from /var/log.

**Answer:**

**🧹 Old Log Cleanup Script (clean\_old\_logs.sh)**

#!/bin/bash

# Define log directory

LOG\_DIR="/var/log"

# Define file age threshold (in days)

DAYS=7

# Find and delete log files older than 7 days

find "$LOG\_DIR" -type f -name "\*.log" -mtime +$DAYS -exec rm -f {} \;

# Print success message

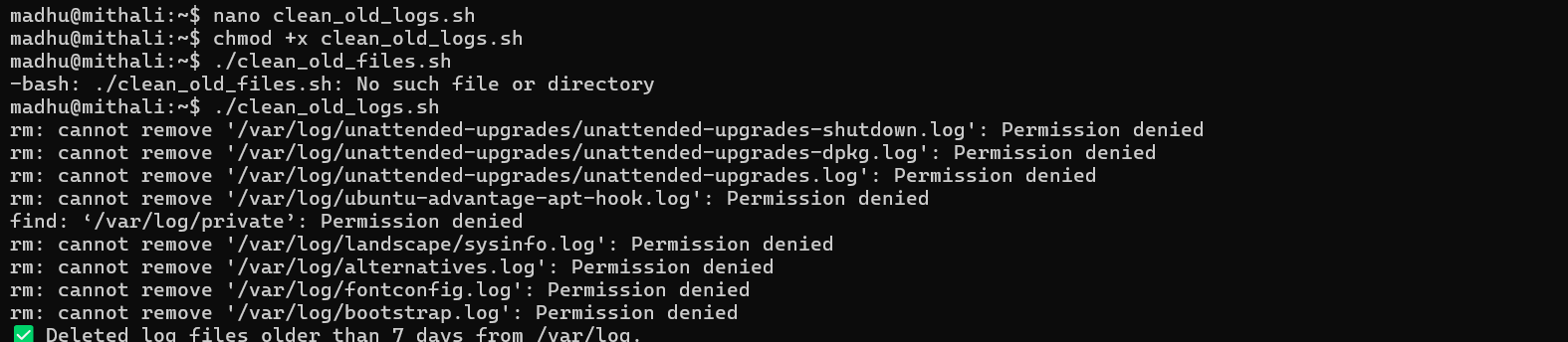
echo "✅ Deleted log files older than $DAYS days from $LOG\_DIR."

**Explanation**:

* **Defines the log directory** (/var/log).
* **Defines the threshold (**DAYS=7**)** for old logs.
* **Finds and deletes old logs using**find:
  + -type f → Selects only files.
  + -name "\*.log" → Matches only .log files.
  + -mtime +7 → Files older than **7 days**.
  + -exec rm -f {} \; → Deletes the found files.

**Steps to run:**

1. create file:nano clean\_old\_logs.sh
2. Make it executable:chmod +x clean\_old\_logs.sh
3. Run the script:sudo ./clean\_old\_logs.sh



✅ **Challenge 8**: Automate user account creation – Write a script that takes the username as an argument, checks, if the user exists, gives the message “user already exists“ else creates a new user, adds it to a “devops“ group, and sets up a default home directory.

**Answer:**

**👤 User Creation Script (create\_user.sh)**

#!/bin/bash

# Check if a username is provided

if [ $# -eq 0 ]; then

echo "❌ Error: No username provided."

echo "Usage: sudo ./create\_user.sh <username>"

exit 1

fi

USERNAME="$1"

GROUP="devops"

# Check if user already exists

if id "$USERNAME" &>/dev/null; then

echo "✅ User '$USERNAME' already exists."

else

# Create the group if it doesn't exist

if ! getent group "$GROUP" > /dev/null; then

echo "⏳ Creating group '$GROUP'..."

sudo groupadd "$GROUP"

fi

# Create user with home directory and add to group

echo "⏳ Creating user '$USERNAME'..."

sudo useradd -m -s /bin/bash -G "$GROUP" "$USERNAME"

# Set a default password (optional, force change on first login)

echo "$USERNAME:ChangeMe123" | sudo chpasswd

sudo passwd --expire "$USERNAME"

echo "✅ User '$USERNAME' created successfully and added to group '$GROUP'."

echo "ℹ️ Default password: ChangeMe123 (User must change it on first login)"

fi

**Explanation:**

* **Checks if a username is provided** ($# -eq 0).
* **Stores username and group (**devops**)**.
* **Checks if the user already exists** using id "$USERNAME" &>/dev/null.
  + If yes, prints **"User already exists"**.
* **Creates the**devops**group** if it doesn't exist.
* **Creates the user** with:
  + -m → Creates a home directory.
  + -s /bin/bash → Sets Bash as the default shell.
  + -G devops → Adds the user to the devops group.
* **Sets a default password** (ChangeMe123) and **forces a password change** on first login.

**Steps to run the script:**

1. Create the script file & copy the above code there:nano create\_user.sh
2. Make it executable:**chmod +x create\_user.sh**
3. Run the script with a username:**sudo ./create\_user.sh devops\_user**



✅ **Challenge 9**: Use awk or sed in a script to process a log file and extract only error messages.

**Answer:**

**🪵 Error Log Extractor Script (extract\_errors.sh)**

#!/bin/bash

# Define log file path

LOG\_FILE="/var/log/syslog" # Change this to your log file

OUTPUT\_FILE="error\_messages.log"

# Check if log file exists

if [ ! -f "$LOG\_FILE" ]; then

echo "❌ Error: Log file '$LOG\_FILE' not found!"

exit 1

fi

# Extract error messages using awk

awk '/error|ERROR|Error/ {print}' "$LOG\_FILE" > "$OUTPUT\_FILE"

# Alternatively, use sed:

# sed -n '/error\|ERROR\|Error/p' "$LOG\_FILE" > "$OUTPUT\_FILE"

echo "✅ Extracted error messages saved to '$OUTPUT\_FILE'."

**Explanation:**

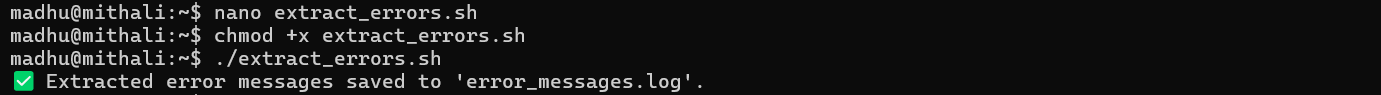
* **Defines the log file (**LOG\_FILE**) and output file (**OUTPUT\_FILE**)**.
* **Checks if the log file exists** before processing.
* **Extracts error messages & saves extracted errors** to error\_messages.log:
  + awk '/error|ERROR|Error/ {print}' "$LOG\_FILE" → Searches for "error" in any case and prints matching lines.
* **Alternative using**sed:**sed -n '/error\|ERROR\|Error/p' "$LOG\_FILE" > "$OUTPUT\_FILE"**

**To run the script:**

1. Create the script file & copy-paste the script above and save it.:

**nano extract\_errors.sh**

1. Make it executable:**chmod +x extract\_errors.sh**
2. Run the script:**./extract\_errors.sh**



✅ **Challenge 10**: Set up a cron job that runs a script to back up (zip/tar) a directory daily.

**Answer:**

**🗂️ Directory Backup Script (backup.sh)**

#!/bin/bash

# Define backup directory and destination

SOURCE\_DIR="/path/to/directory" # Change this to the directory you want to back up

BACKUP\_DIR="/path/to/backup"

TIMESTAMP=$(date +"%Y-%m-%d\_%H-%M-%S")

BACKUP\_FILE="$BACKUP\_DIR/backup\_$TIMESTAMP.tar.gz"

# Create backup directory if it doesn't exist

mkdir -p "$BACKUP\_DIR"

# Create the backup archive

tar -czf "$BACKUP\_FILE" "$SOURCE\_DIR"

# Print success message

echo "✅ Backup created: $BACKUP\_FILE"

**Steps:**

1. Create the “backup.sh“ file and copy the above code there:**nano backup.sh**
2. Make the Script Executable:chmod +x backup.sh
3. Schedule a Daily Cron Job:
   * Open the cron editor:**crontab -e**
   * Add the following line to run the script daily at 2 AM:**0 2 \* \* \* /path/to/backup.sh >> /var/log/backup.log 2>&1**

