Bash scrip Task

**Bash Script Tasks**

1. Create a bash script to **check if a directory is available or not**.
2. Create a bash script to **create multiple files**.
3. Create a bash script to **take a backup of a directory**.
4. Create a bash script to **install Nginx on an EC2 server**.
5. Create a bash script to **install Apache Tomcat on an EC2 server**.
6. Create a bash script to **check if the Nginx service is running**, if not running then script should start the service.
7. Create a bash script for a **calculator**.
8. Create a bash script to **check if a directory exists**, if not then create a directory.
9. Create a bash script to **delete the last 3 lines of a file**.
10. Bash script to monitor cpu and if it is more than 80% then send email notification.
11. Bash script to monitor disk space and if it is more than 80% then send email notification.
12. Bash script to monitor memory and if it is more than 80% then send email notification.

**1. Check if a Directory is Available or Not**

#!/bin/bash

DIR="/path/to/directory"

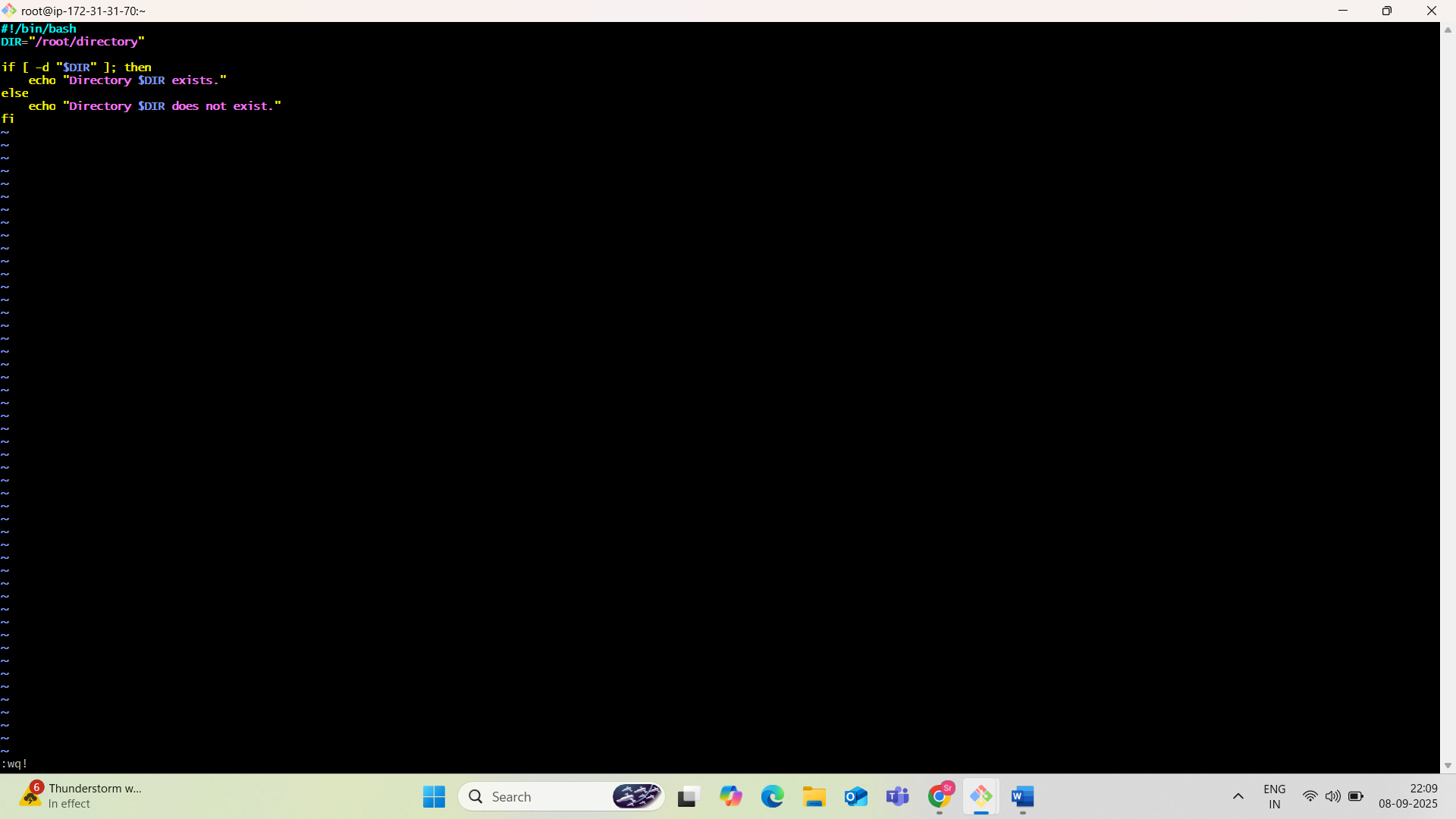
if [ -d "$DIR" ]; then

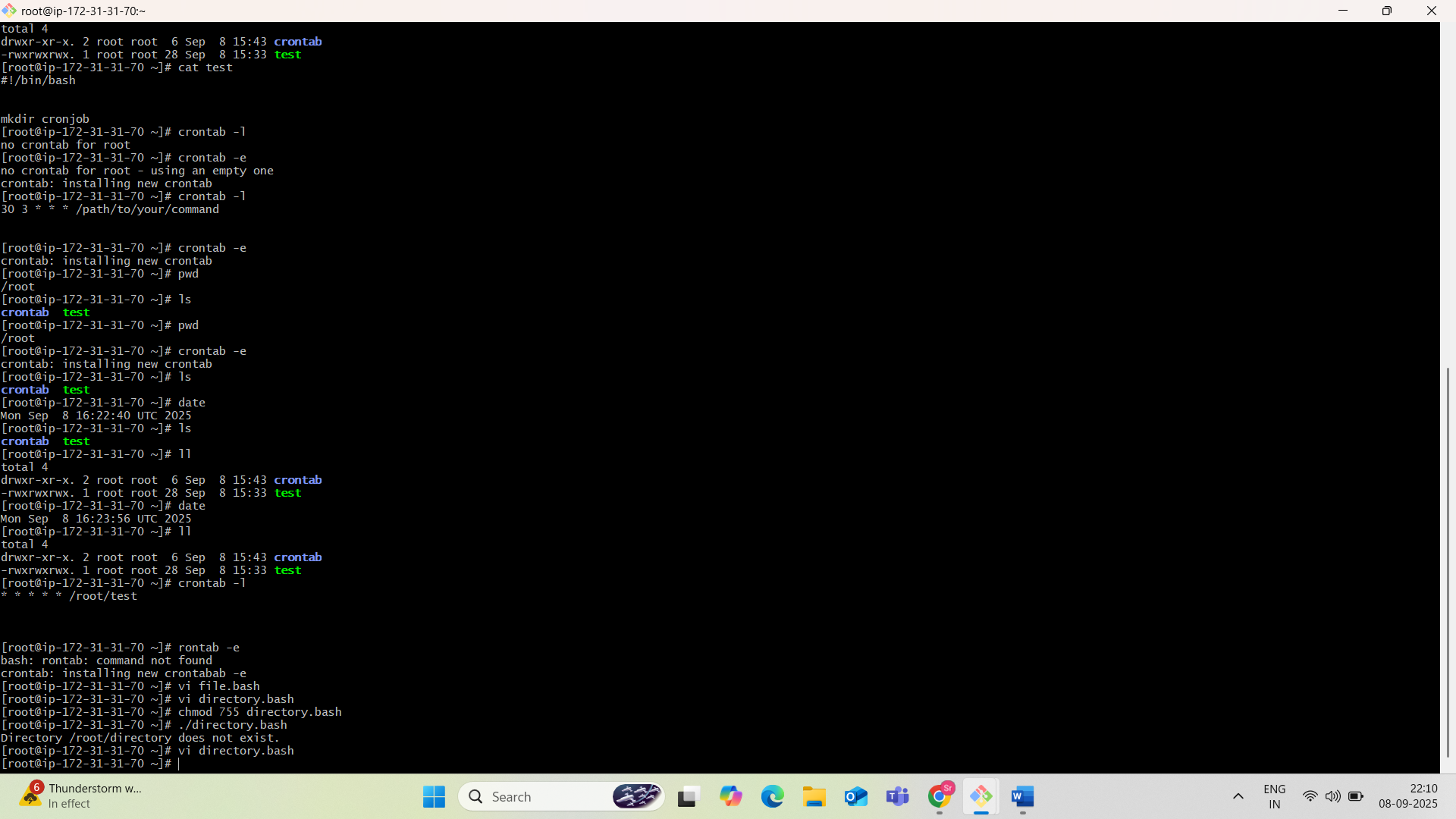
echo "Directory exists."

else

echo "Directory does not exist."

fi





**Bash Script: 2) Create Multiple Files**

**Vi multiplefiles.bash**

#!/bin/bash

# Ask the user how many files to create

read -p "How many files do you want to create? " COUNT

# Ask for base file name

read -p "Enter the base name for the files (e.g., file, report, log): " BASENAME

# Ask for file extension (optional)

read -p "Enter the file extension (e.g., txt, log), leave blank for none: " EXT

# Loop to create the files

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

if [ -z "$EXT" ]; then

touch "${BASENAME}\_${i}"

echo "Created: ${BASENAME}\_${i}"

else

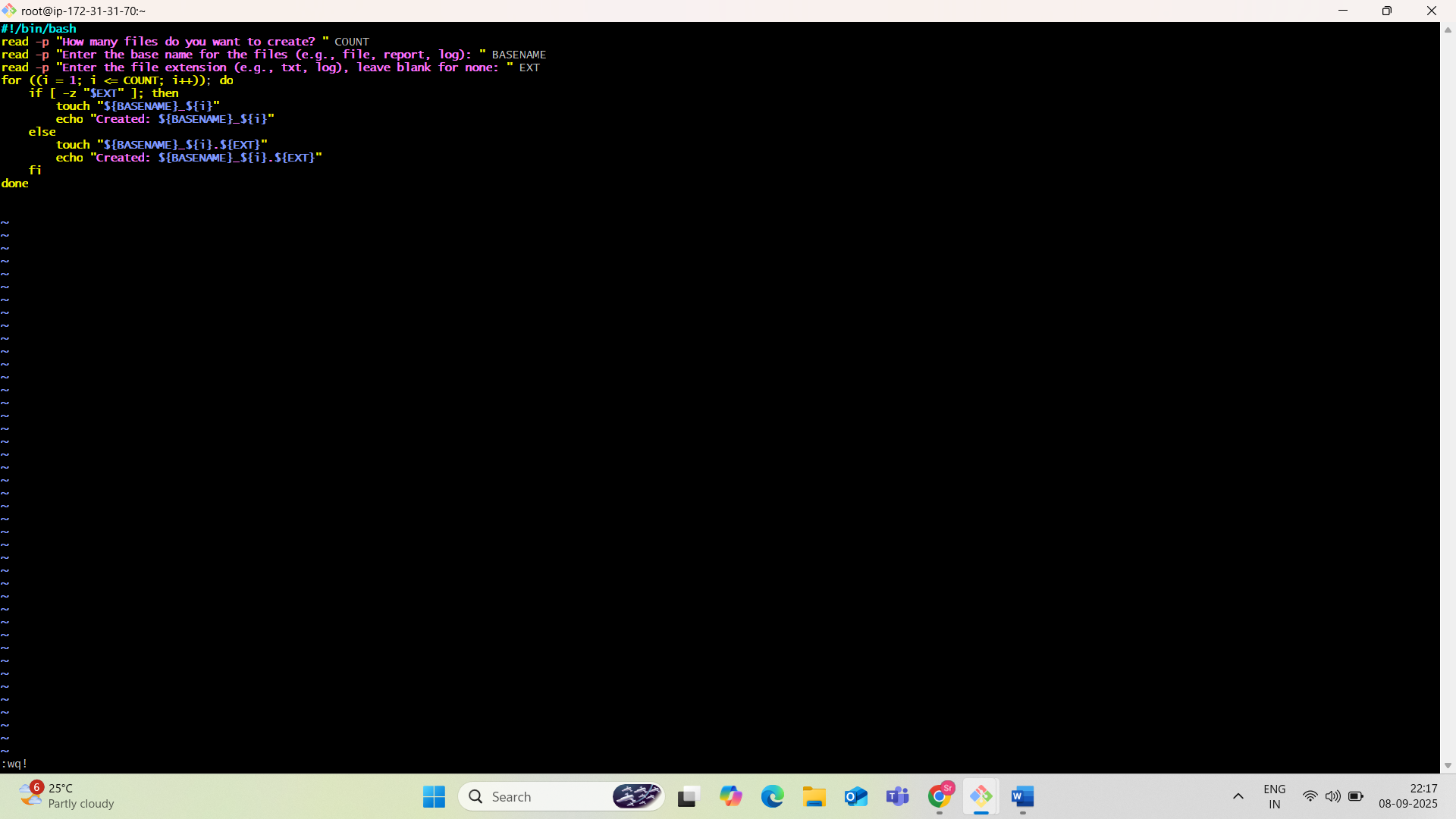
touch "${BASENAME}\_${i}.${EXT}"

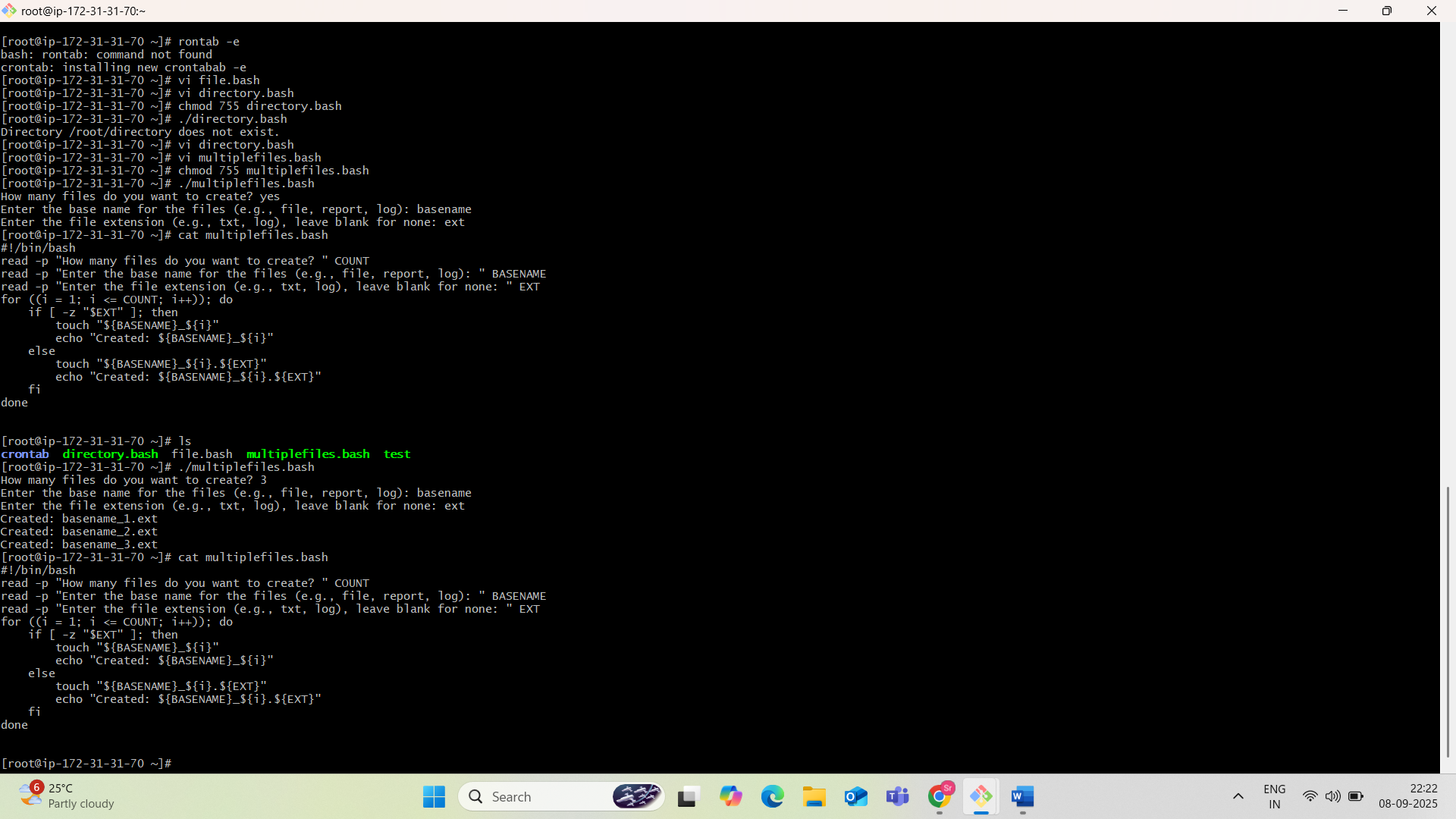
echo "Created: ${BASENAME}\_${i}.${EXT}"

fi

done

echo "✅ $COUNT files created successfully."





3)Create a bash script to **take a backup of a directory**

**What This Bash Script Will Do:**

* Prompt the user for:
  + The **source directory** (to be backed up)
  + The **backup destination directory**
* Create a compressed .tar.gz backup with a **timestamped filename**
* Save it to the destination directory
* Display a message when the backup is complete

Create a file named backupdir.bash

#!/bin/bash

read -p "Enter the full path of the directory to back up: " SOURCE\_DIR

read -p "Enter the backup destination directory: " BACKUP\_DIR

if [ ! -d "$SOURCE\_DIR" ]; then

echo "❌ Source directory does not exist: $SOURCE\_DIR"

exit 1

fi

# Create destination directory

mkdir -p "$BACKUP\_DIR"

# Get current timestamp

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

# Get the base name of the source directory (e.g., 'myfolder' from '/home/user/myfolder')

SOURCE\_NAME=$(basename "$SOURCE\_DIR")

# Compose backup filename

BACKUP\_FILE="$BACKUP\_DIR/${SOURCE\_NAME}\_backup\_$TIMESTAMP.tar.gz"

# Create the compressed backup

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

# Check if the backup was successful

if [ $? -eq 0 ]; then

echo "✅ Backup created successfully at: $BACKUP\_FILE"

else

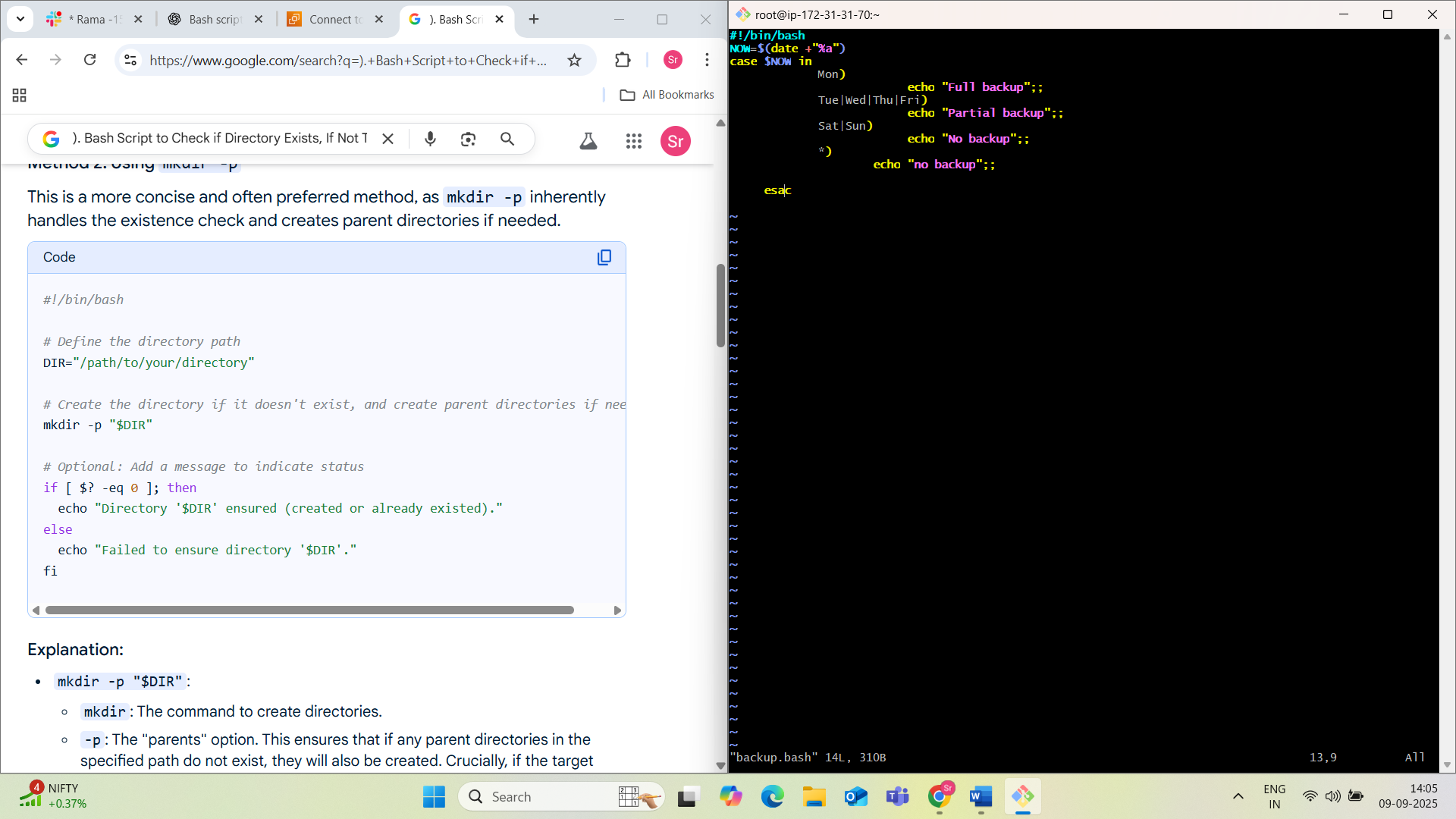
echo "❌ Backup failed!"

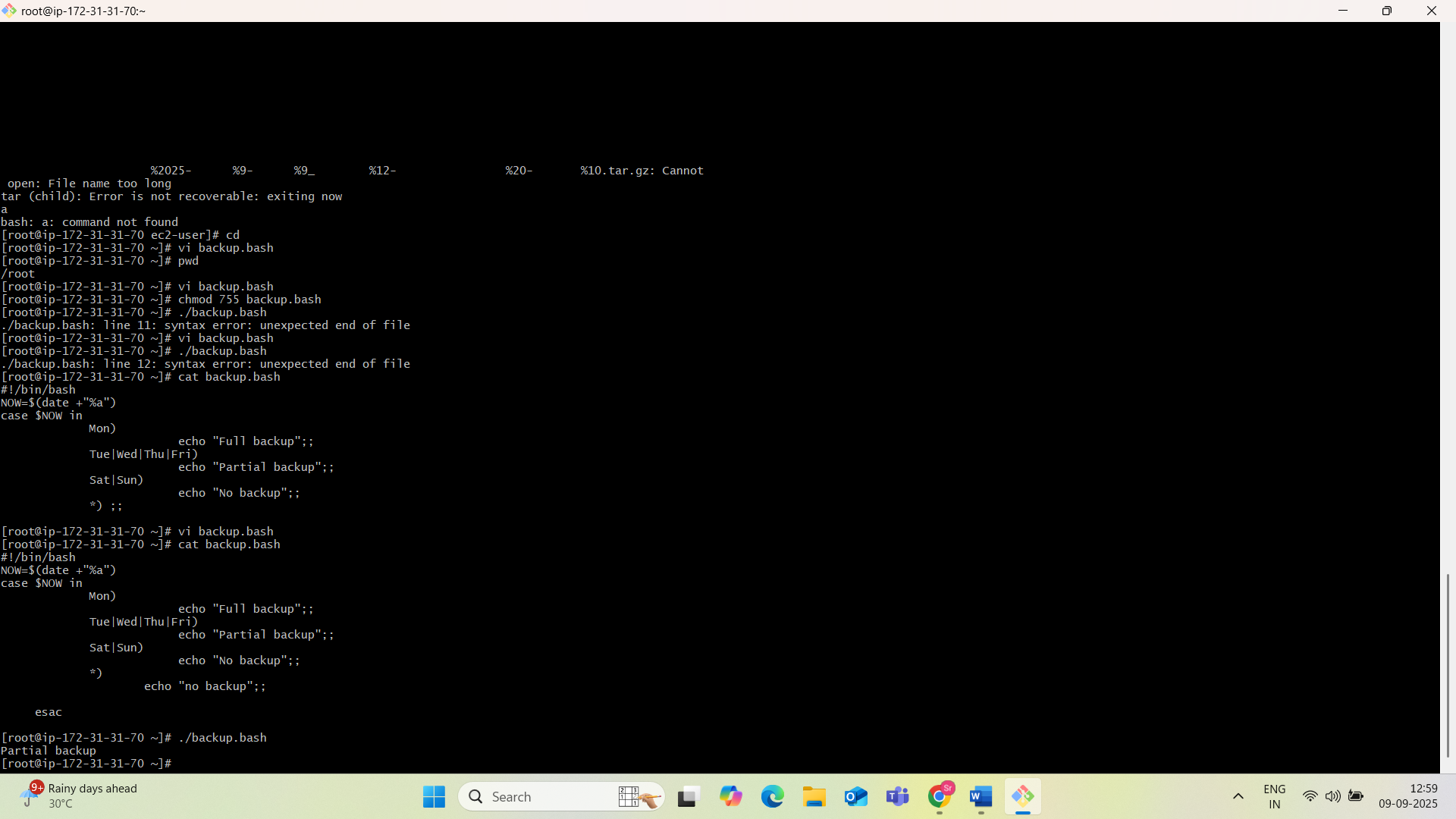
fi

3)Create a bash script to **take a backup of a directory**

**Objective:**

Create a Bash script that:

* Takes a source directory as input.
* Creates a compressed .tar.gz backup file.
* Saves it in a backup directory with a timestamp.
* 



Can be scheduled using cron (optional for automation).

4)**Bash Script to Install Nginx on an EC2 Server**

**#!/bin/bash**

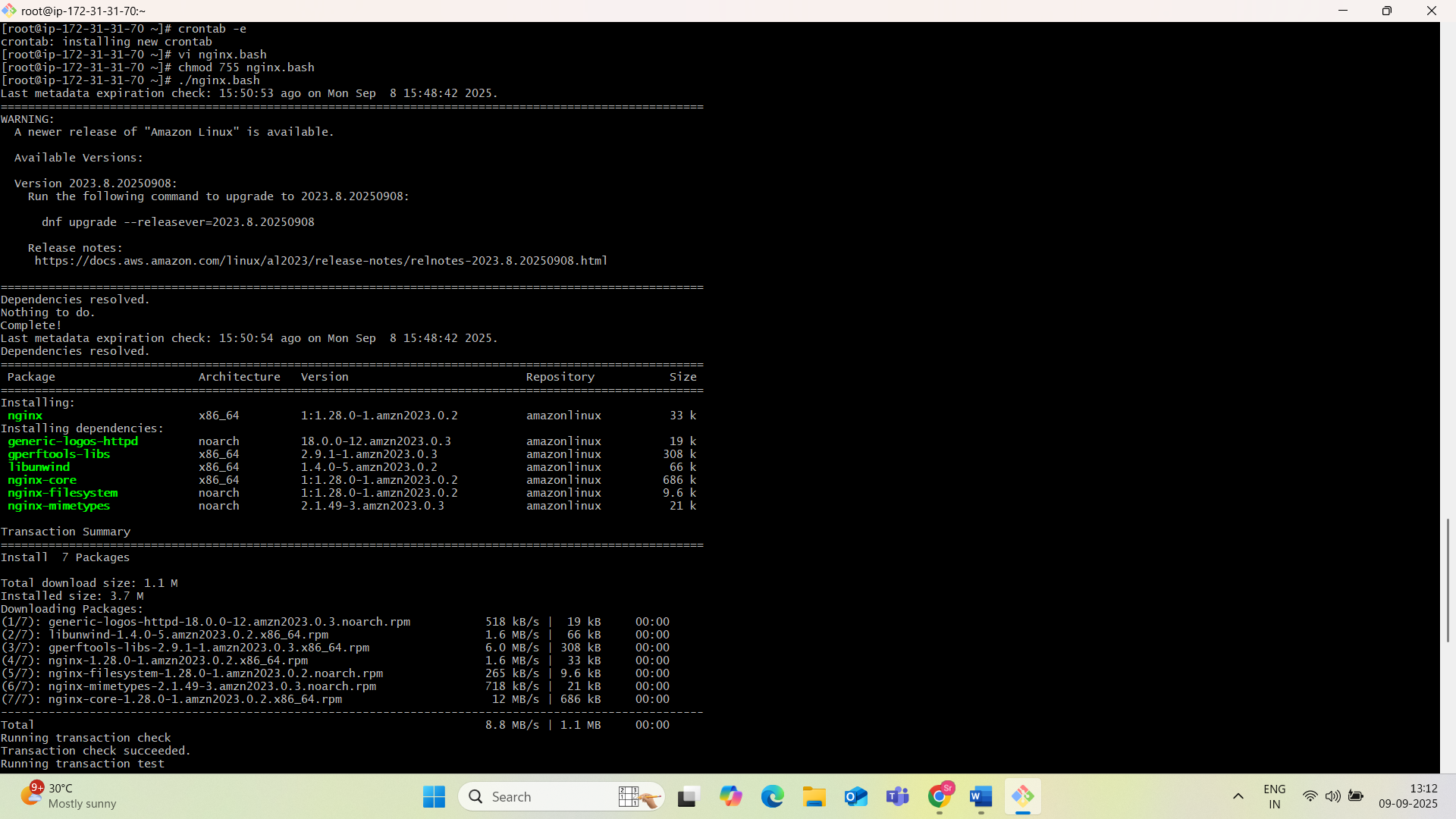
**yum update -y**

**yum install nginx -y**

**systemctl enable nginx**

**systemctl start nginx**

**echo "<html><h1>Hello from your Nginx server on Amazon Linux!</h1></html>" > /usr/share/nginx/html/index.html**



**5). Bash Script to Install Apache Tomcat on an EC2 Server**

#!/bin/bash

# Update system packages

sudo yum update -y

# Install Java Development Kit (JDK) - adjust version as needed

sudo yum install java-1.8.0-openjdk-devel -y

# Download Apache Tomcat (adjust version and URL as needed)

TOMCAT\_VERSION="9.0.80" # Example version

TOMCAT\_URL="https://archive.apache.org/dist/tomcat/tomcat-9/v${TOMCAT\_VERSION}/bin/apache-tomcat-${TOMCAT\_VERSION}.tar.gz"

wget ${TOMCAT\_URL} -P /tmp

# Create a directory for Tomcat installation

sudo mkdir -p /opt/tomcat

# Extract Tomcat to the installation directory

sudo tar -xvzf /tmp/apache-tomcat-${TOMCAT\_VERSION}.tar.gz -C /opt/tomcat --strip-components=1

# Create a Tomcat user and group for security

sudo groupadd tomcat

sudo useradd -s /bin/nologin -g tomcat -d /opt/tomcat tomcat

# Set ownership and permissions

sudo chown -R tomcat:tomcat /opt/tomcat

sudo chmod -R u+rwx,g+rx,o-rwx /opt/tomcat/conf

sudo chmod -R u+x,g+x,o-x /opt/tomcat/bin

# Configure Systemd Service for Tomcat

sudo bash -c 'cat > /etc/systemd/system/tomcat.service <<EOF

[Unit]

Description=Apache Tomcat Web Application Container

After=network.target

[Service]

Type=forking

User=tomcat

Group=tomcat

Environment="JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk" # Adjust if different JDK path

Environment="CATALINA\_PID=/opt/tomcat/temp/tomcat.pid"

Environment="CATALINA\_HOME=/opt/tomcat"

Environment="CATALINA\_BASE=/opt/tomcat"

ExecStart=/opt/tomcat/bin/startup.sh

ExecStop=/opt/tomcat/bin/shutdown.sh

RestartSec=10

Restart=always

[Install]

WantedBy=multi-user.target

EOF'

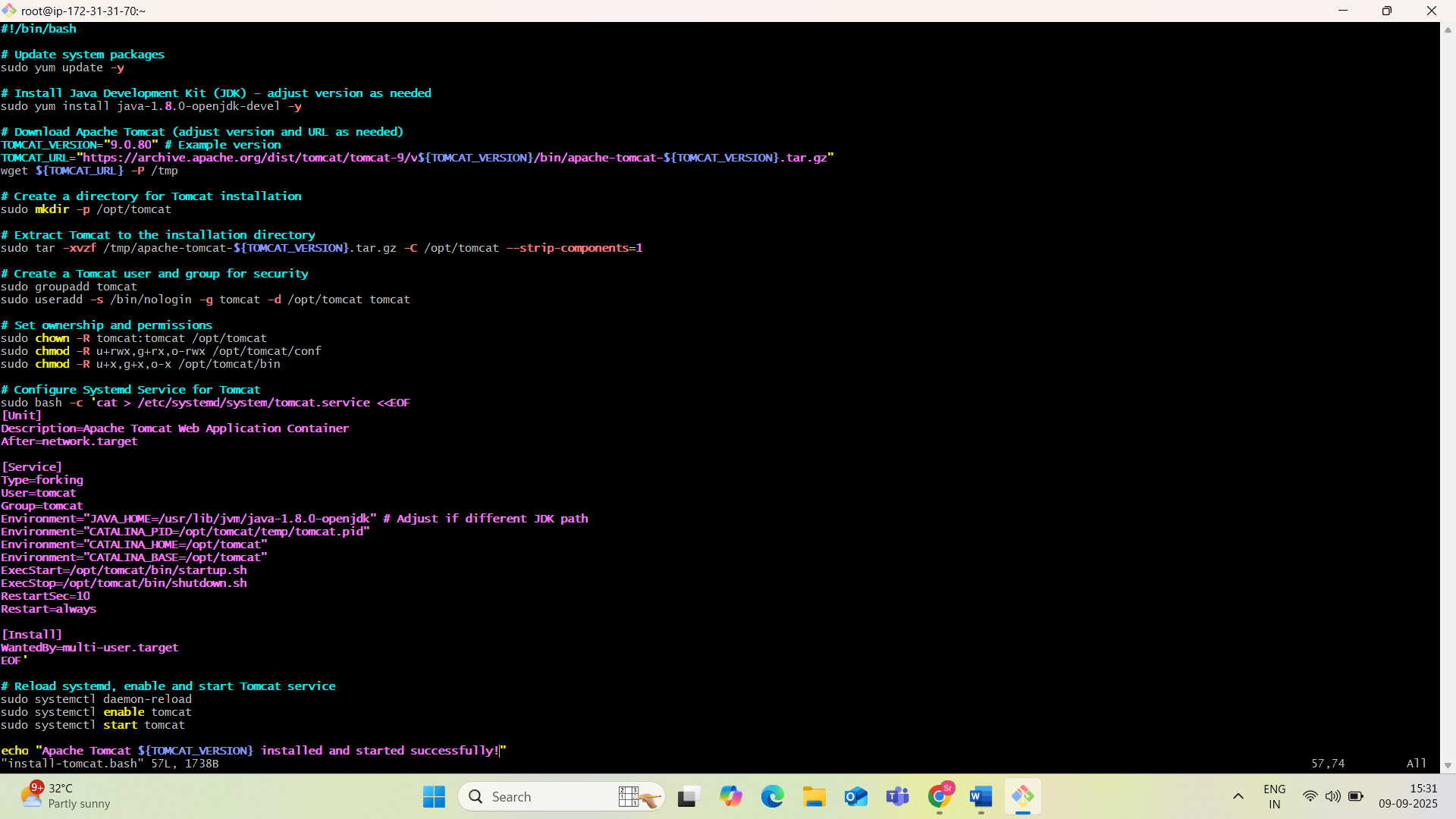
# Reload systemd, enable and start Tomcat service

sudo systemctl daemon-reload

sudo systemctl enable tomcat

sudo systemctl start tomcat

echo "Apache Tomcat ${TOMCAT\_VERSION} installed and started successfully!"





**✅ Steps:**

1. Install Java (Tomcat requires it).and install tomcat.
2. Create a dedicated Tomcat user.
3. Download and extract Tomcat.
4. Set permissions.
5. Create a systemd service for Tomcat.
6. Start and enable the service.

6)**. Bash Script to Check if Nginx Is Running and Start It If Not**

A Bash script to check if Nginx is running and start it if it is not can be created as follows:**#!/bin/bash**

**# Define the service name**

**SERVICE\_NAME="nginx"**

**# Check if Nginx is running using systemctl**

**if systemctl is-active --quiet "$SERVICE\_NAME"; then**

**echo "$SERVICE\_NAME is running."**

**else**

**echo "$SERVICE\_NAME is not running. Attempting to start it..."**

**# Start Nginx**

**sudo systemctl start "$SERVICE\_NAME"**

**# Check if Nginx started successfully**

**if systemctl is-active --quiet "$SERVICE\_NAME"; then**

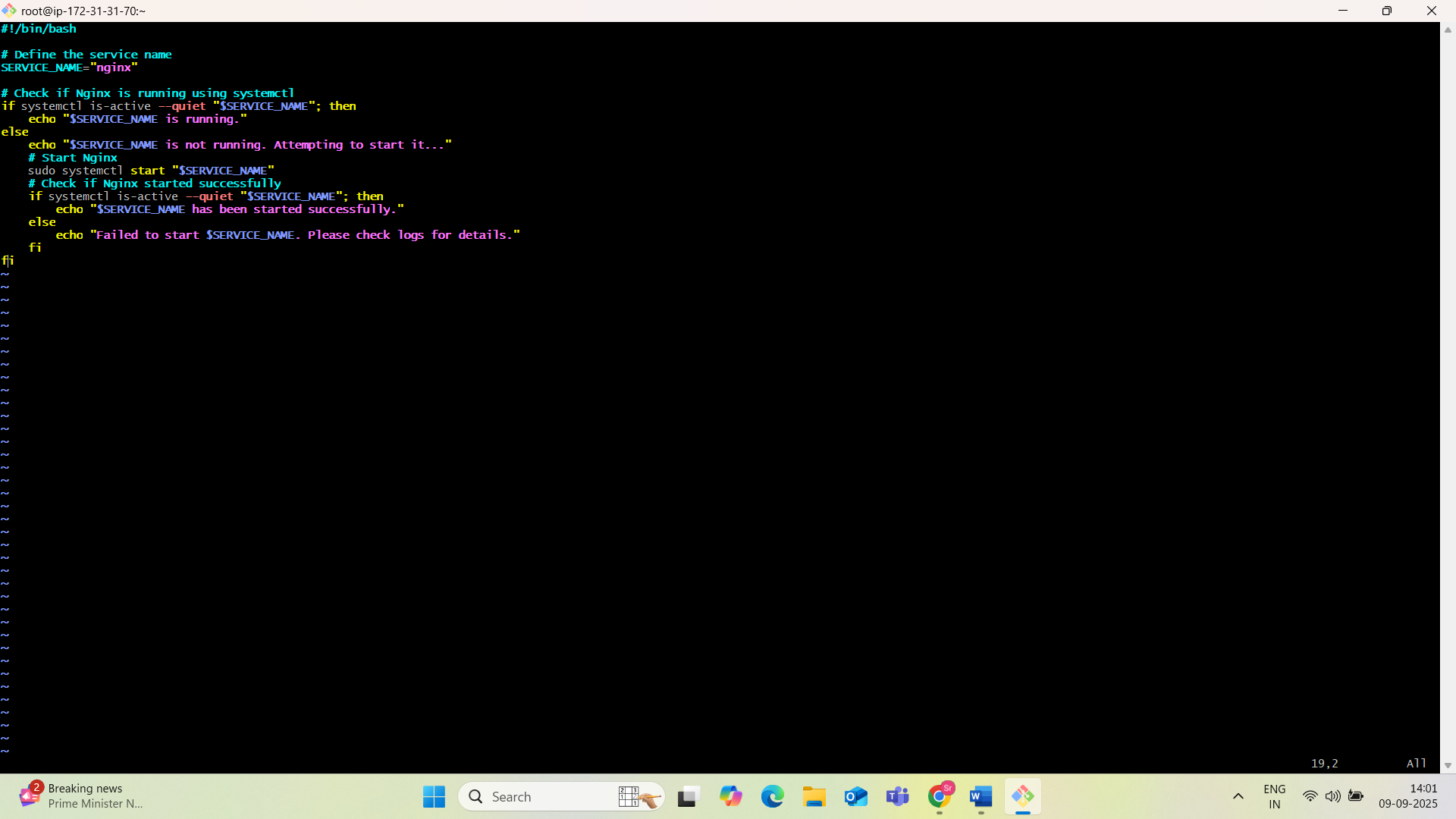
**echo "$SERVICE\_NAME has been started successfully."**

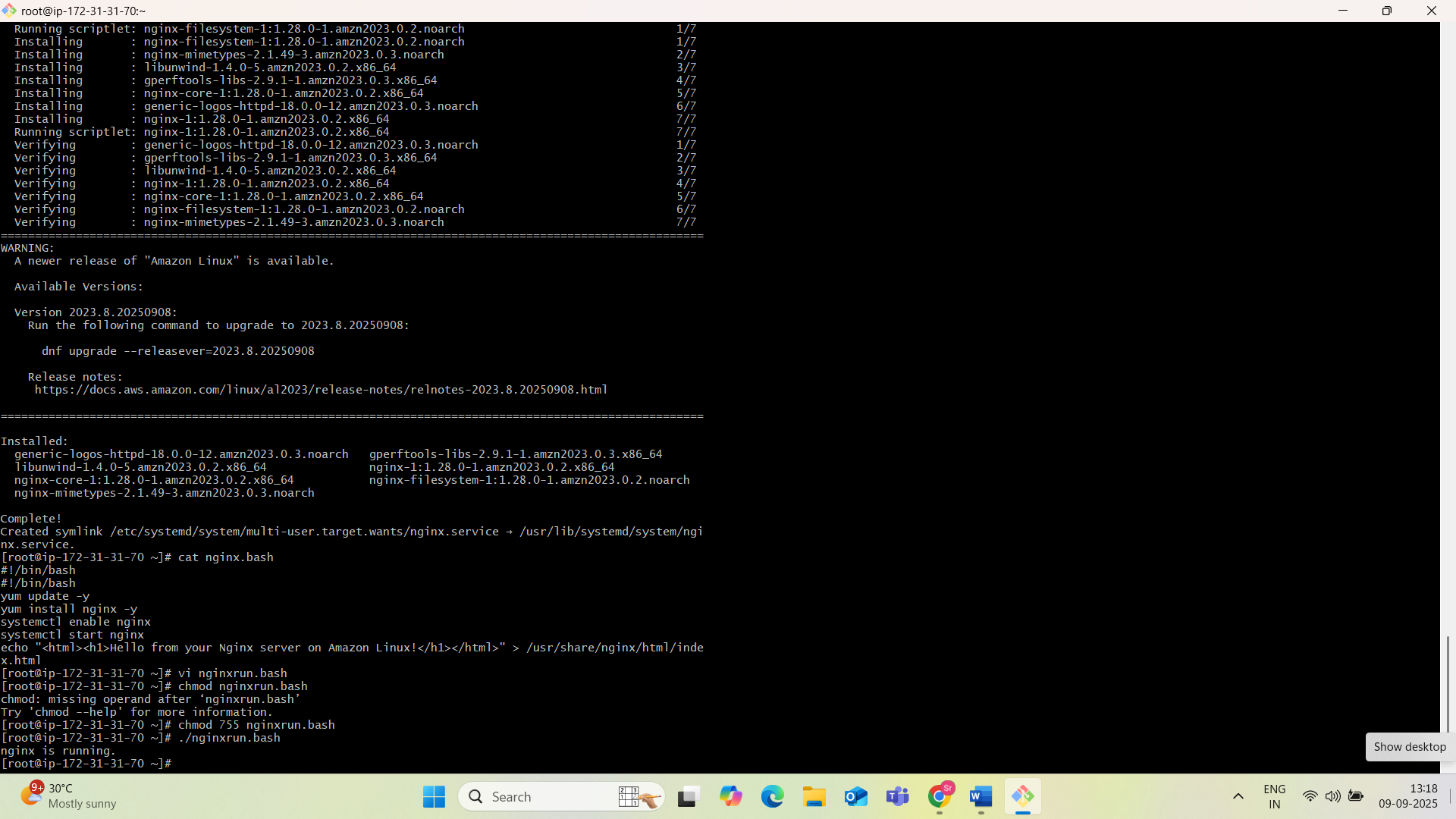
**else**

**echo "Failed to start $SERVICE\_NAME. Please check logs for details."**

**fi**

**fi**



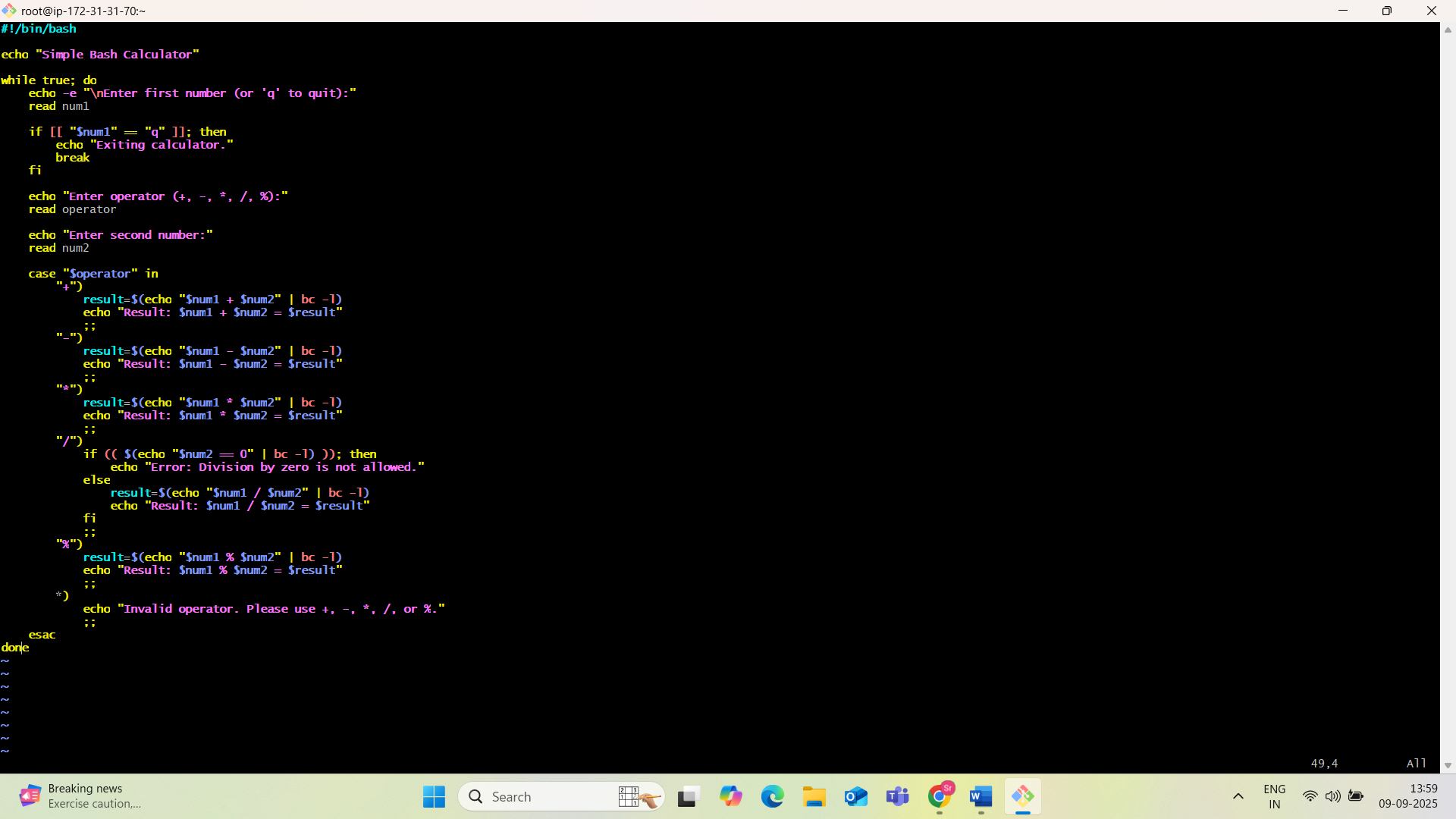


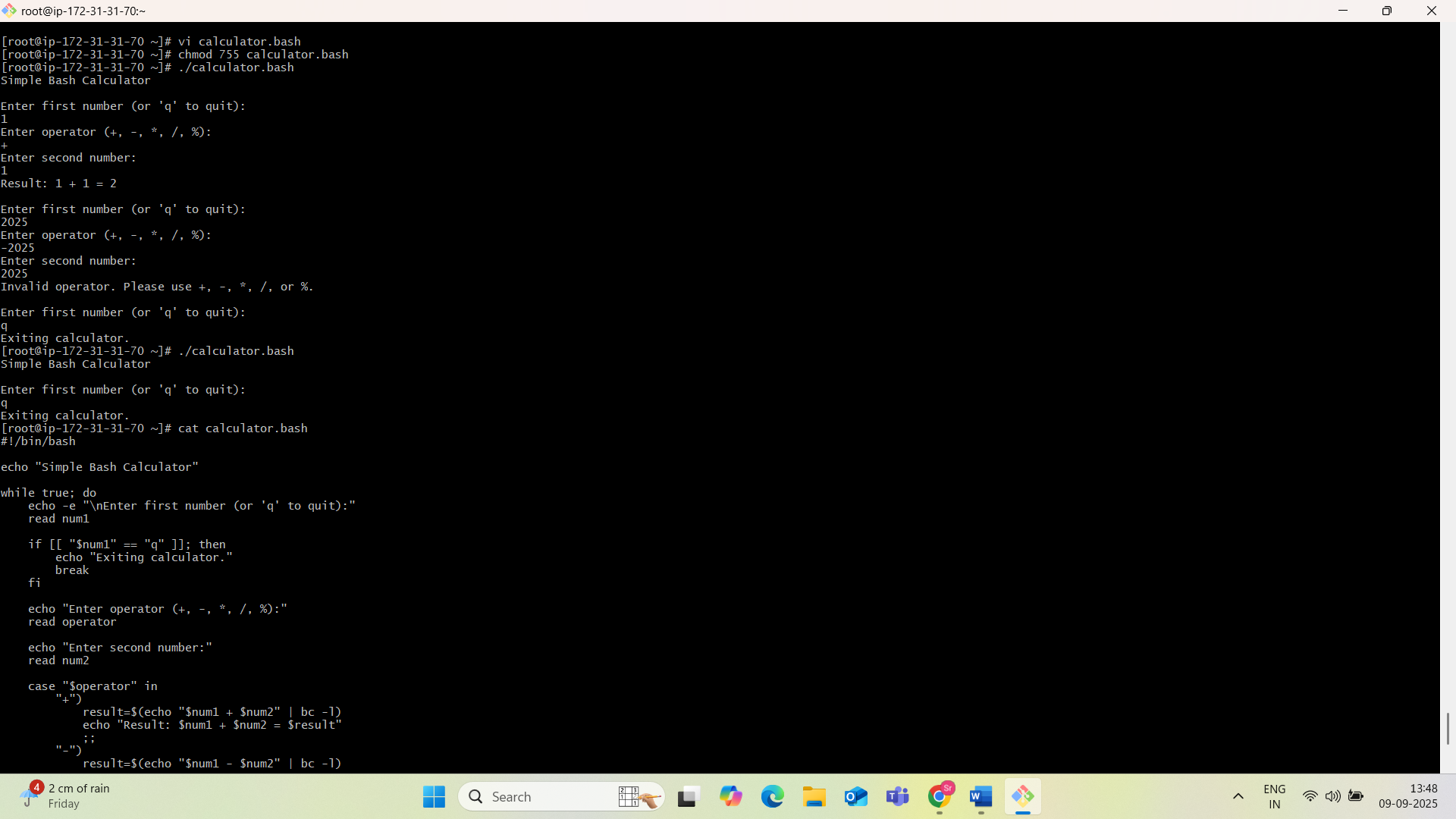
**✅ Steps:**

* Checks the status of nginx.
* If not active, starts it.
* Else, does nothing.

7)**. Bash Script for a Calculator**

*#!/bin/bash*  
  
echo "Simple Bash Calculator"  
  
while true; do  
 echo -e "\nEnter first number (or 'q' to quit):"  
 read num1  
  
 if [[ "$num1" == "q" ]]; then  
 echo "Exiting calculator."  
 break  
 fi  
  
 echo "Enter operator (+, -, \*, /, %):"  
 read operator  
  
 echo "Enter second number:"  
 read num2  
  
 case "$operator" in  
 "+")  
 result=$(echo "$num1 + $num2" | bc -l)  
 echo "Result: $num1 + $num2 = $result"  
 ;;  
 "-")  
 result=$(echo "$num1 - $num2" | bc -l)  
 echo "Result: $num1 - $num2 = $result"  
 ;;  
 "\*")  
 result=$(echo "$num1 \* $num2" | bc -l)  
 echo "Result: $num1 \* $num2 = $result"  
 ;;  
 "/")  
 if (( $(echo "$num2 == 0" | bc -l) )); then  
 echo "Error: Division by zero is not allowed."  
 else  
 result=$(echo "$num1 / $num2" | bc -l)  
 echo "Result: $num1 / $num2 = $result"  
 fi  
 ;;  
 "%")  
 result=$(echo "$num1 % $num2" | bc -l)  
 echo "Result: $num1 % $num2 = $result"  
 ;;  
 \*)  
 echo "Invalid operator. Please use +, -, \*, /, or %."  
 ;;  
 esac  
done





8)**. Bash Script to Check if Directory Exists, If Not Then Create It**

A Bash script to check if a directory exists and create it if it does not can be implemented using the if statement and the -d operator, or more concisely using mkdir -p.

Method 1: Using if [ ! -d ... ] and mkdir

This method explicitly checks for the directory's non-existence before attempting creation.

Method 2: Using mkdir -p

This is a more concise and often preferred method, as mkdir -p in herently handles the existence check and creates parent directories if needed.

Am using method2.

#!/bin/bash

# Define the directory path

DIR="/path/to/your/directory"

# Create the directory if it doesn't exist, and create parent directories if needed

mkdir -p "$DIR"

# Optional: Add a message to indicate status

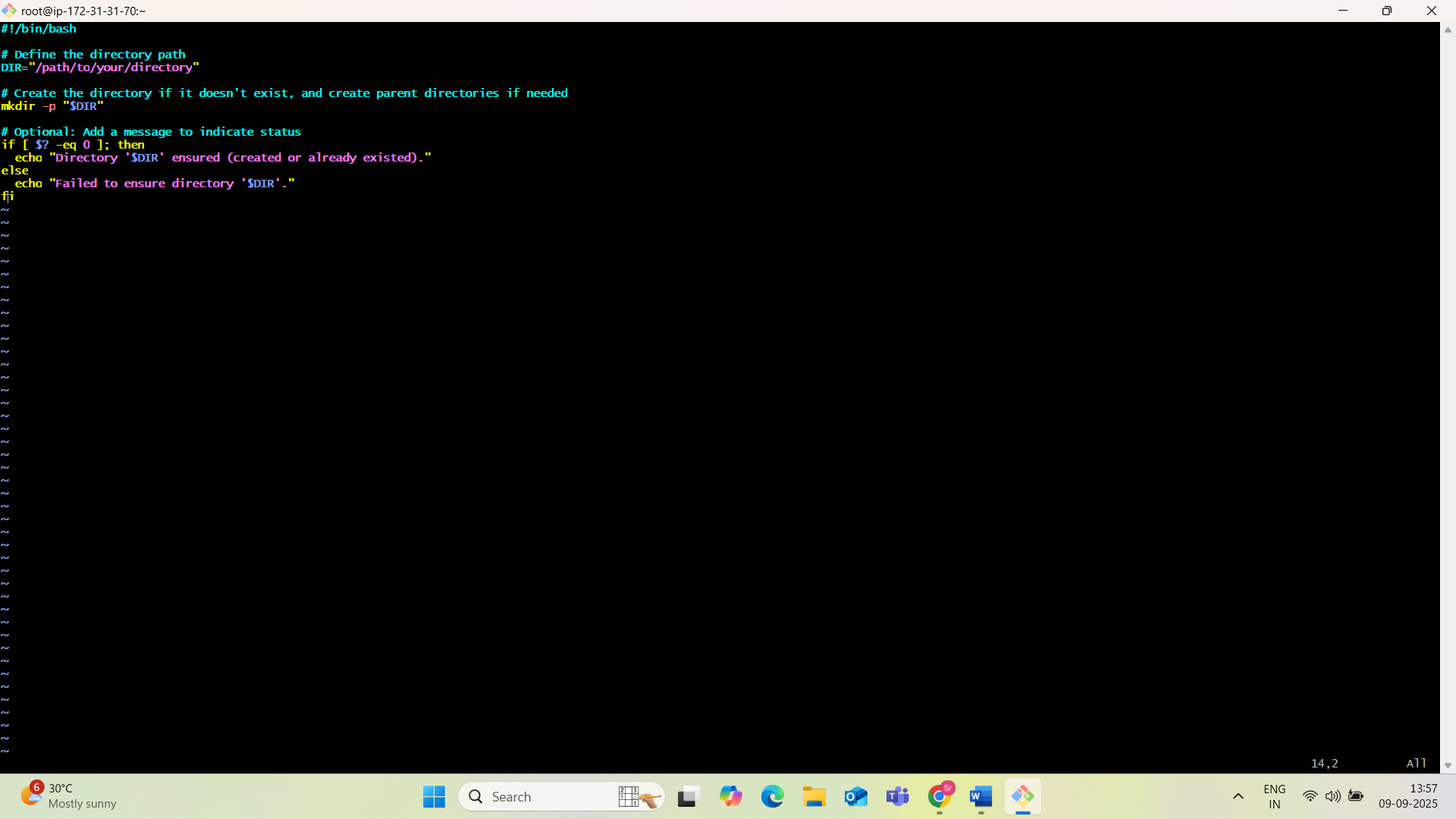
if [ $? -eq 0 ]; then

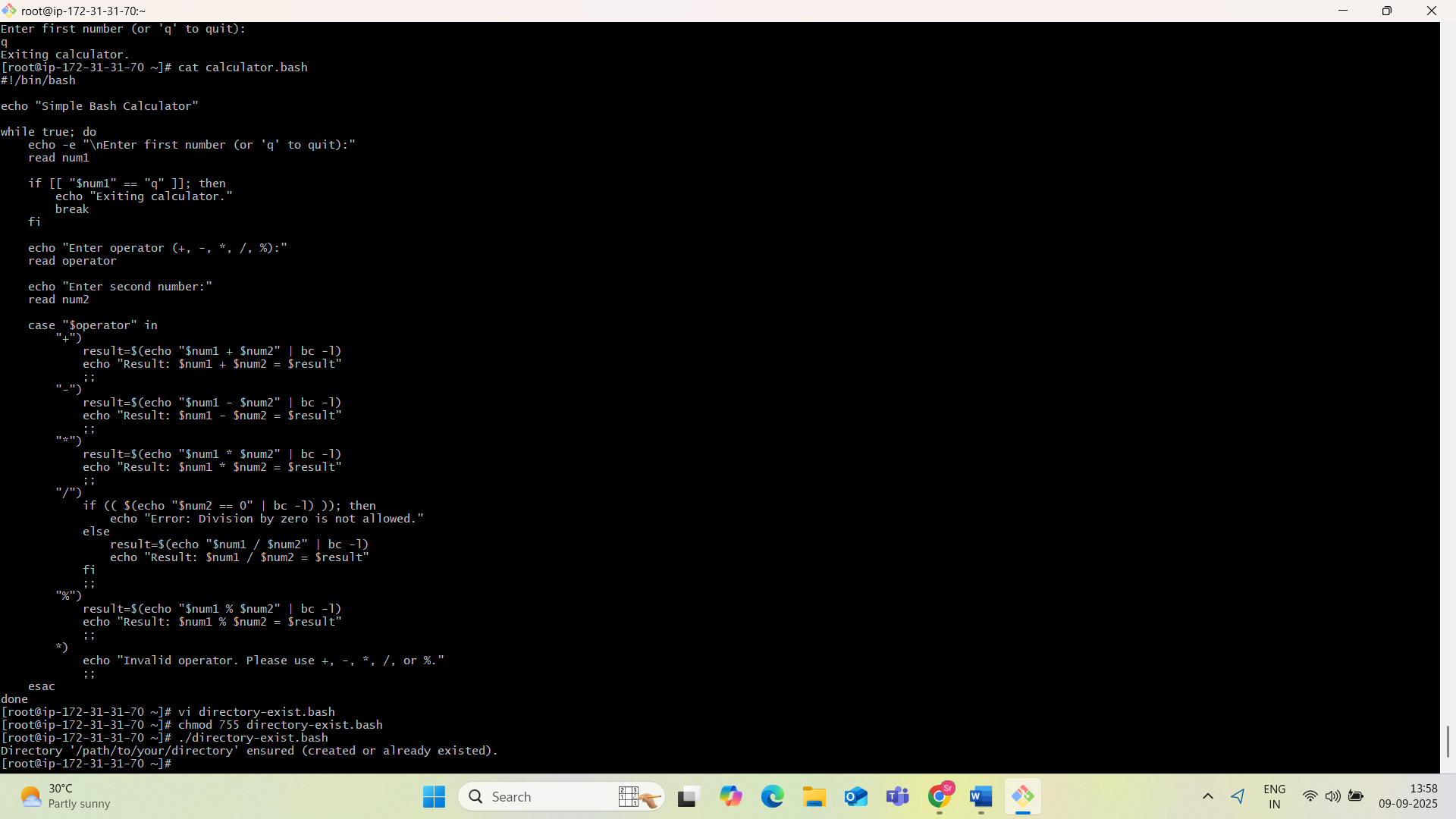
echo "Directory '$DIR' ensured (created or already existed)."

else

echo "Failed to ensure directory '$DIR'."

Fi





9)**. Bash Script to Delete the Last 3 Lines of a File**

To delete the last three lines of a file using a Bash script, the head command can be combined with redirection.

The following script demonstrates this process:

**#!/bin/bash**

**# Check if a filename is provided as an argument**

**if [ -z "$1" ]; then**

**echo "Usage: $0 <filename>"**

**exit 1**

**fi**

**FILE="$1"**

**# Check if the file exists**

**if [ ! -f "$FILE" ]; then**

**echo "Error: File '$FILE' not found."**

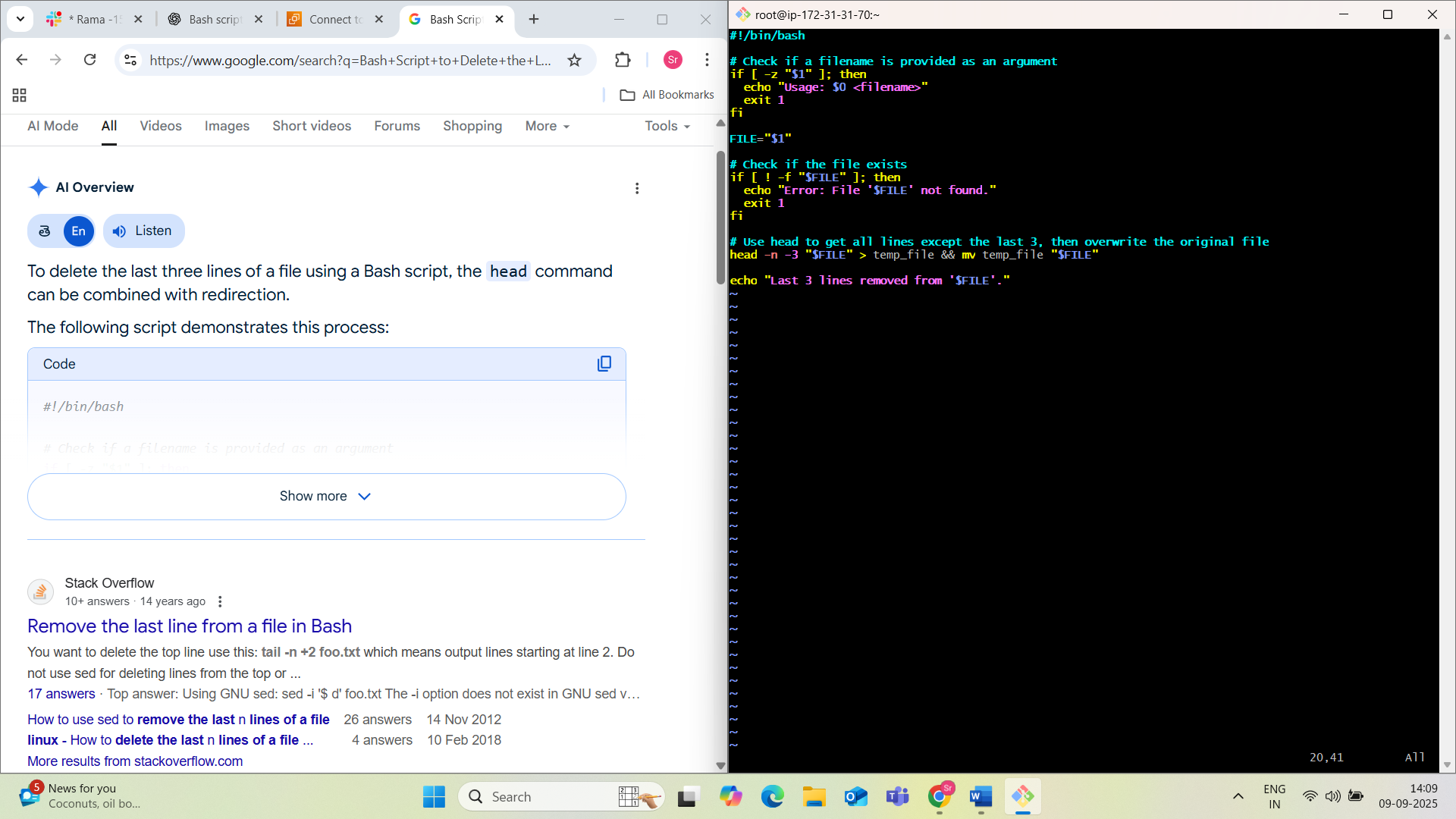
**exit 1**

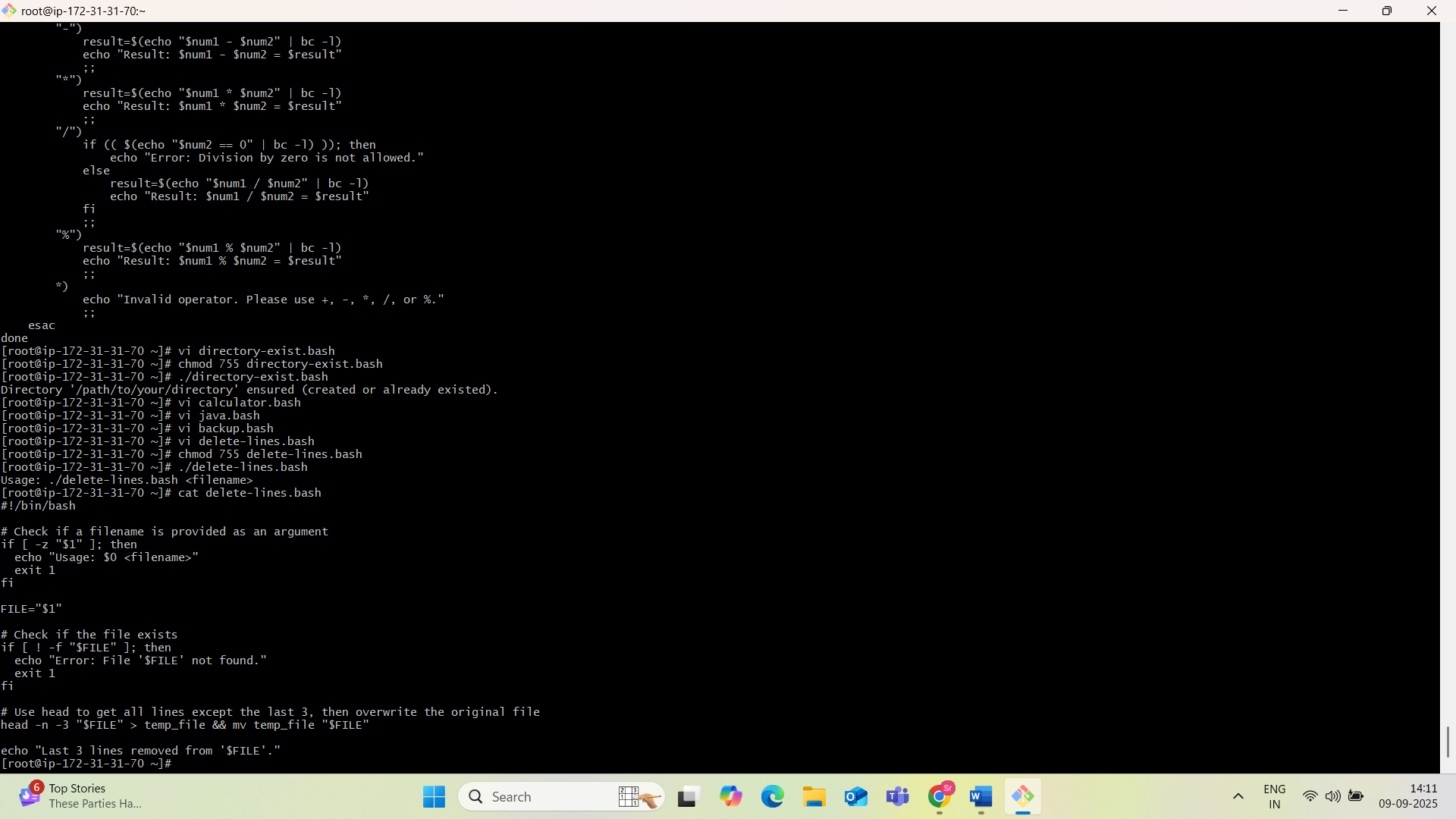
**fi**

**# Use head to get all lines except the last 3, then overwrite the original file**

**head -n -3 "$FILE" > temp\_file && mv temp\_file "$FILE"**

**echo "Last 3 lines removed from '$FILE'."**





10)**. Bash Script to Monitor CPU Usage and Send Email if > 80%**

**#!/bin/bash**

**# Define the email recipient and threshold**

**EMAIL\_RECIPIENT="your\_email@example.com"**

**CPU\_THRESHOLD=80**

**# Get current CPU idle percentage**

**# top -bn1 provides a single snapshot of system processes**

**# grep "Cpu(s)" filters for the CPU line**

**# awk '{print $2}' extracts the idle percentage (second field)**

**CPU\_IDLE=$(top -bn1 | grep "Cpu(s)" | awk '{print $8}' | cut -f 1 -d '.')**

**# Calculate CPU usage**

**CPU\_USAGE=$((100 - CPU\_IDLE))**

**# Get current date and time**

**CURRENT\_DATE=$(date "+%Y-%m-%d %H:%M:%S")**

**# Check if CPU usage exceeds the threshold**

**if [ "$CPU\_USAGE" -ge "$CPU\_THRESHOLD" ]; then**

**SUBJECT="ATTENTION: High CPU Usage Alert on $(hostname)!"**

**MESSAGE\_BODY="CPU usage is currently at $CPU\_USAGE% on $(hostname) at $CURRENT\_DATE. This exceeds the configured threshold of $CPU\_THRESHOLD%."**

**# Add top processes consuming high CPU to the email**

**MESSAGE\_BODY+="\n\nTop 10 processes by CPU consumption:\n"**

**MESSAGE\_BODY+=$(ps aux --sort=-%cpu | head -n 11 | tail -n 10)**

**echo -e "$MESSAGE\_BODY" | mail -s "$SUBJECT" "$EMAIL\_RECIPIENT"**

**else**

**echo "$CURRENT\_DATE: CPU usage is within the threshold ($CPU\_USAGE%)."**

**fi**

(or)

#!/bin/bash

# Set threshold

THRESHOLD=80

# Get CPU usage

CPU=$(top -bn1 | grep "Cpu(s)" | awk '{print 100 - $8}')

# Check condition

if (( ${CPU%.\*} > THRESHOLD )); then

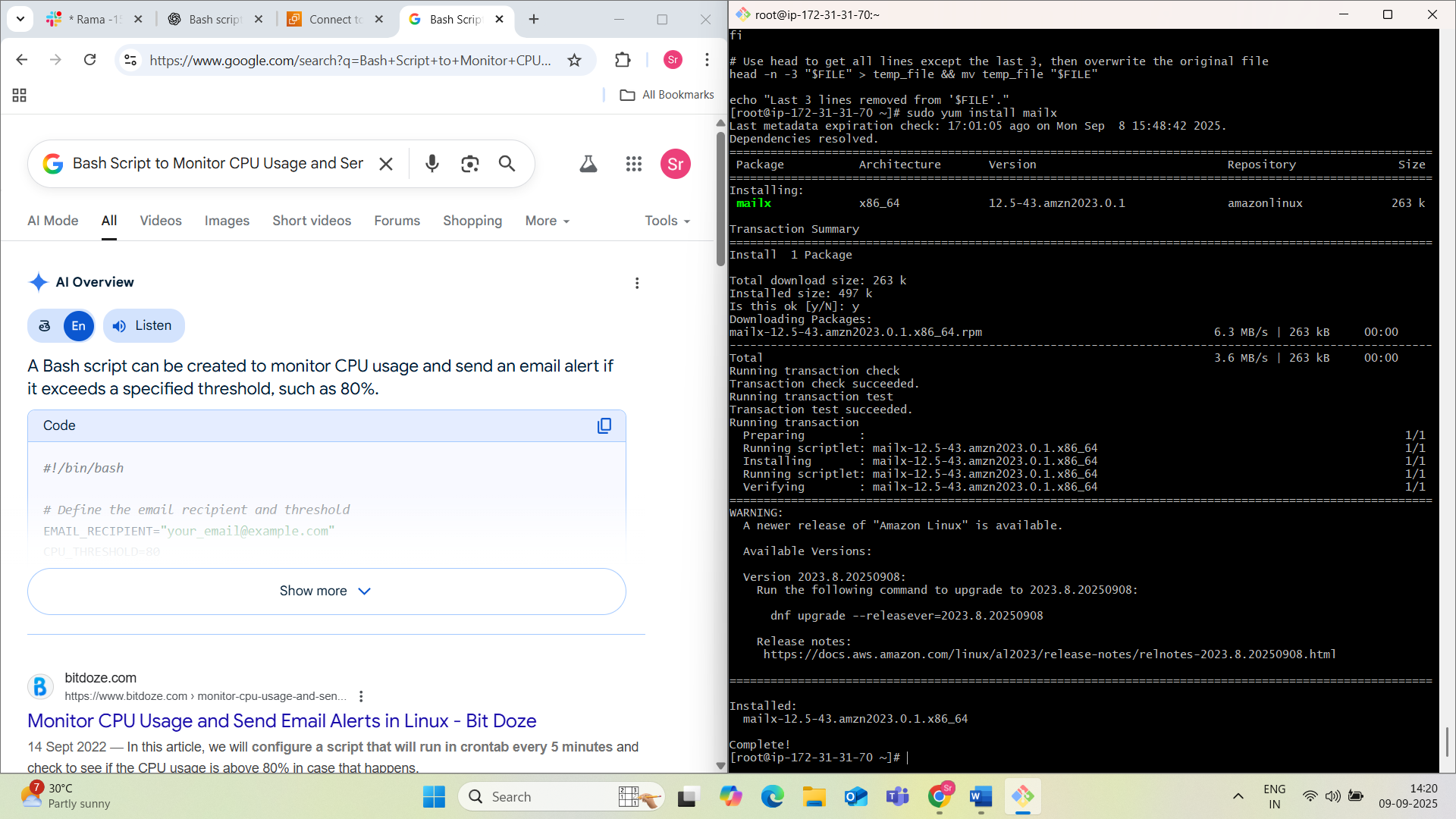
echo "High CPU Usage: $CPU%" | mail -s "CPU Alert" your-email@example.com

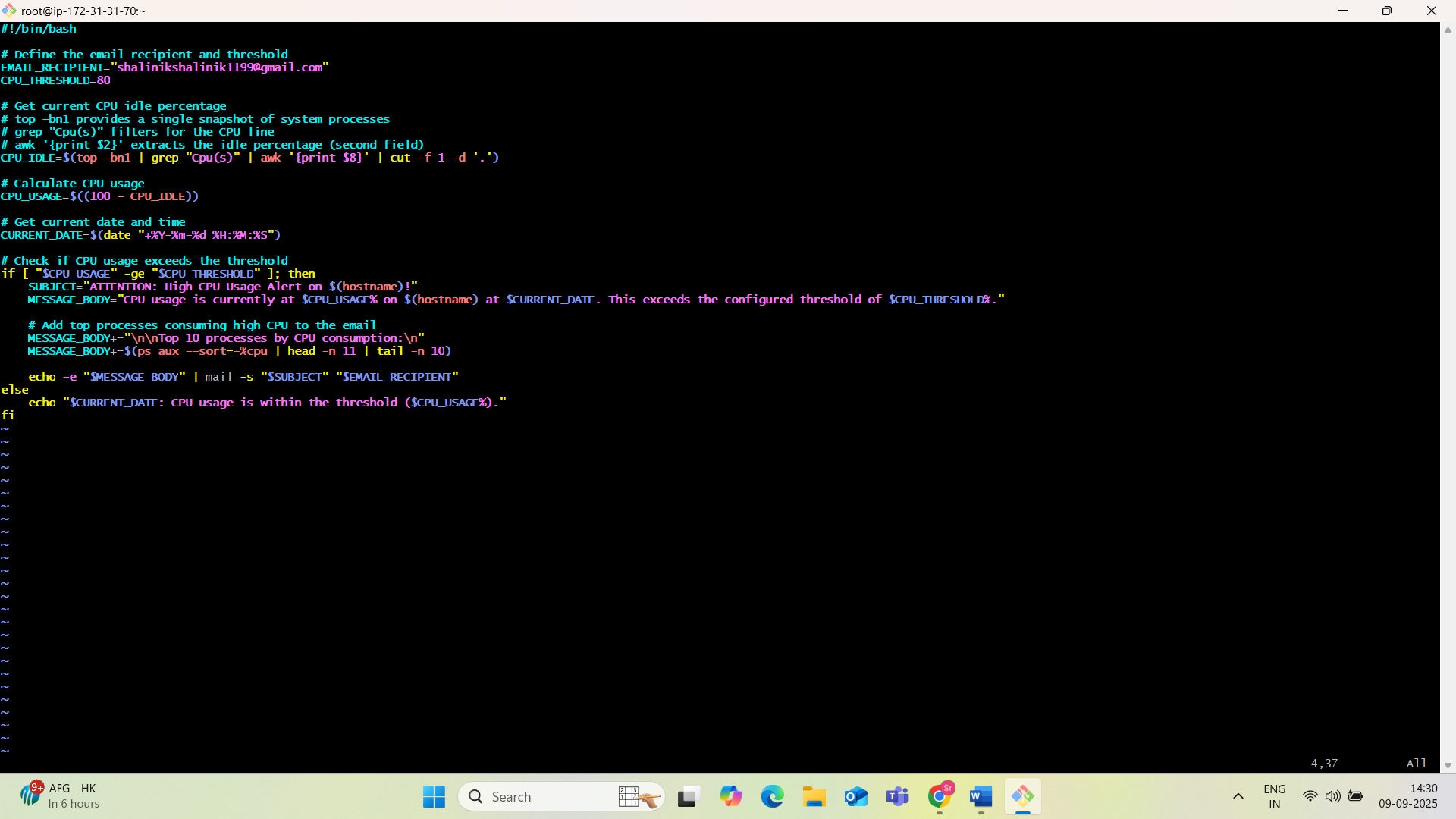
fi

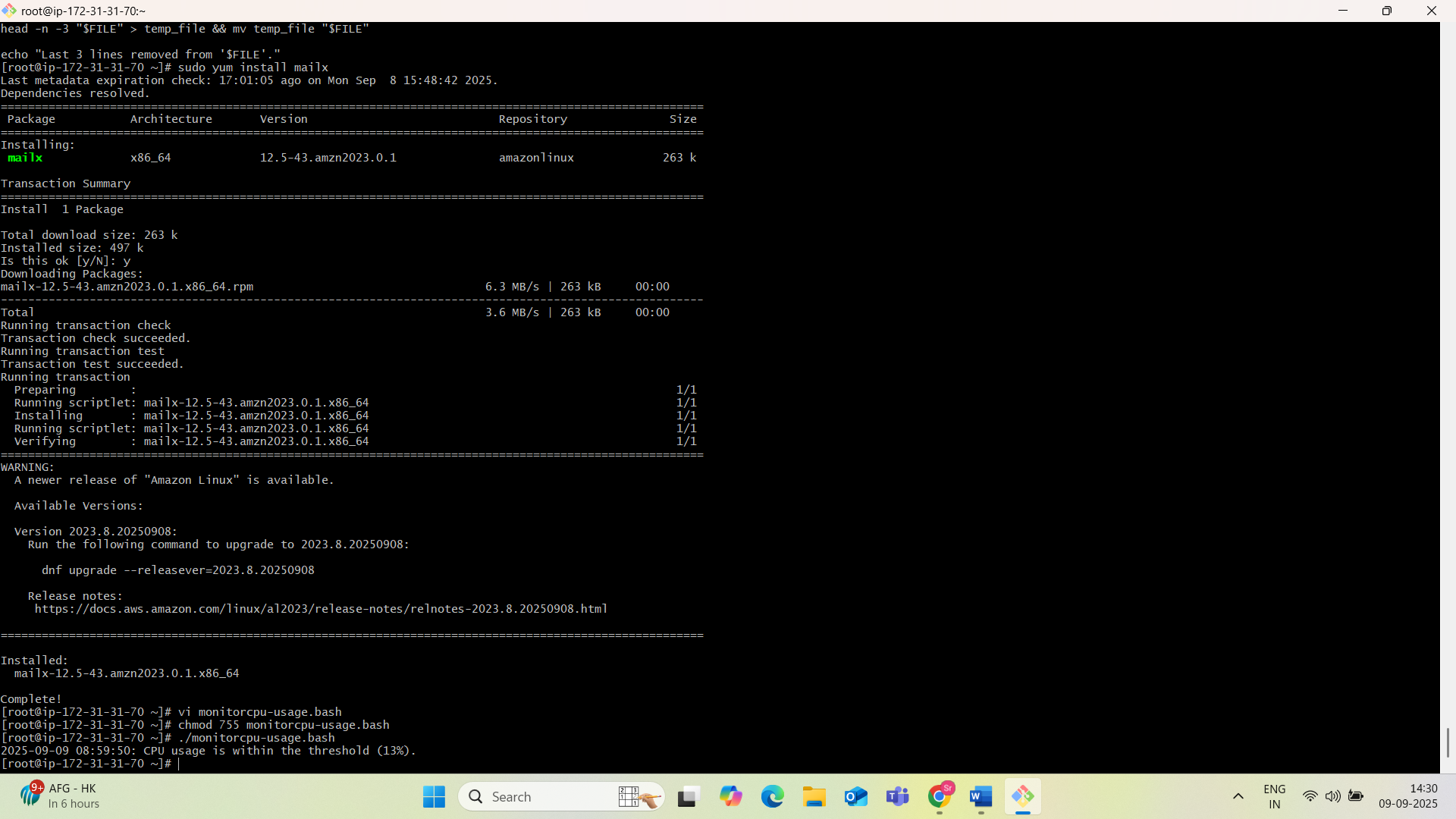
First install mailx:

# Amazon Linux

sudo yum install mailx







11)**. Bash Script to Monitor Disk Space and Send Email if > 80%**

#!/bin/bash

# Define the threshold for disk space usage (in percentage)

THRESHOLD=80

# Define the email recipient and subject

EMAIL\_RECIPIENT="your\_email@example.com"

EMAIL\_SUBJECT="Disk Space Alert on $(hostname)"

# Get disk usage for all mounted filesystems, excluding specific types

# and format the output for easier parsing.

# The 'df -h' command provides human-readable output.

# 'grep -vE' excludes specific filesystem types (e.g., tmpfs, squashfs, loop).

# 'awk' extracts the percentage used and the mount point.

DISK\_USAGE=$(df -h | grep -vE '^Filesystem|tmpfs|squashfs|loop' | awk '{print $5 " " $6}')

# Loop through each line of disk usage output

echo "$DISK\_USAGE" | while read -r line; do

# Extract the usage percentage and mount point

USAGE\_PERCENTAGE=$(echo "$line" | awk '{print $1}' | sed 's/%//')

MOUNT\_POINT=$(echo "$line" | awk '{print $2}')

# Check if the usage percentage is greater than or equal to the threshold

if (( USAGE\_PERCENTAGE >= THRESHOLD )); then

# Compose the email body

EMAIL\_BODY="Disk space on $MOUNT\_POINT is critically low: $USAGE\_PERCENTAGE% used on $(hostname) as of $(date)."

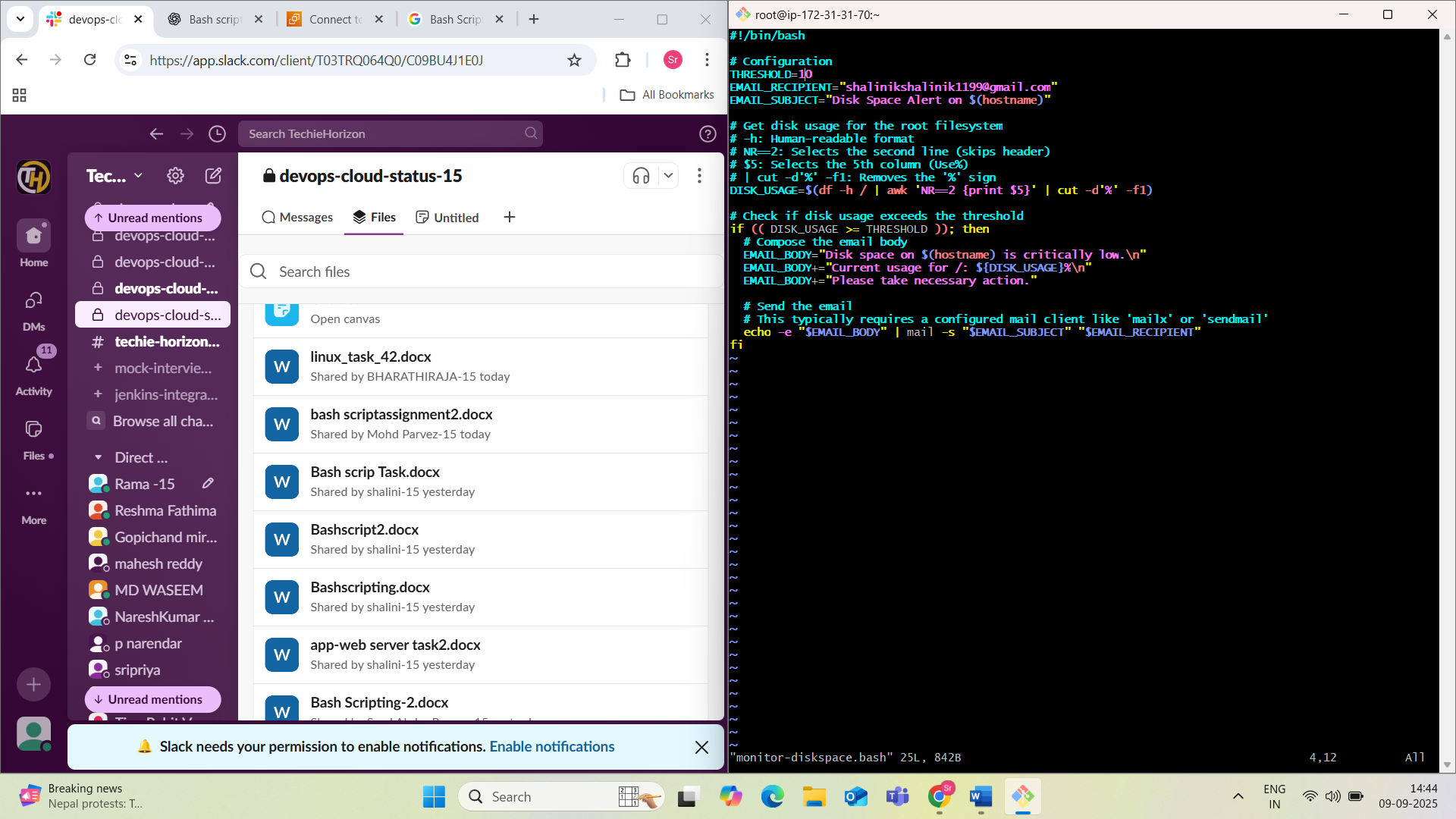
# Send the email using the 'mail' command

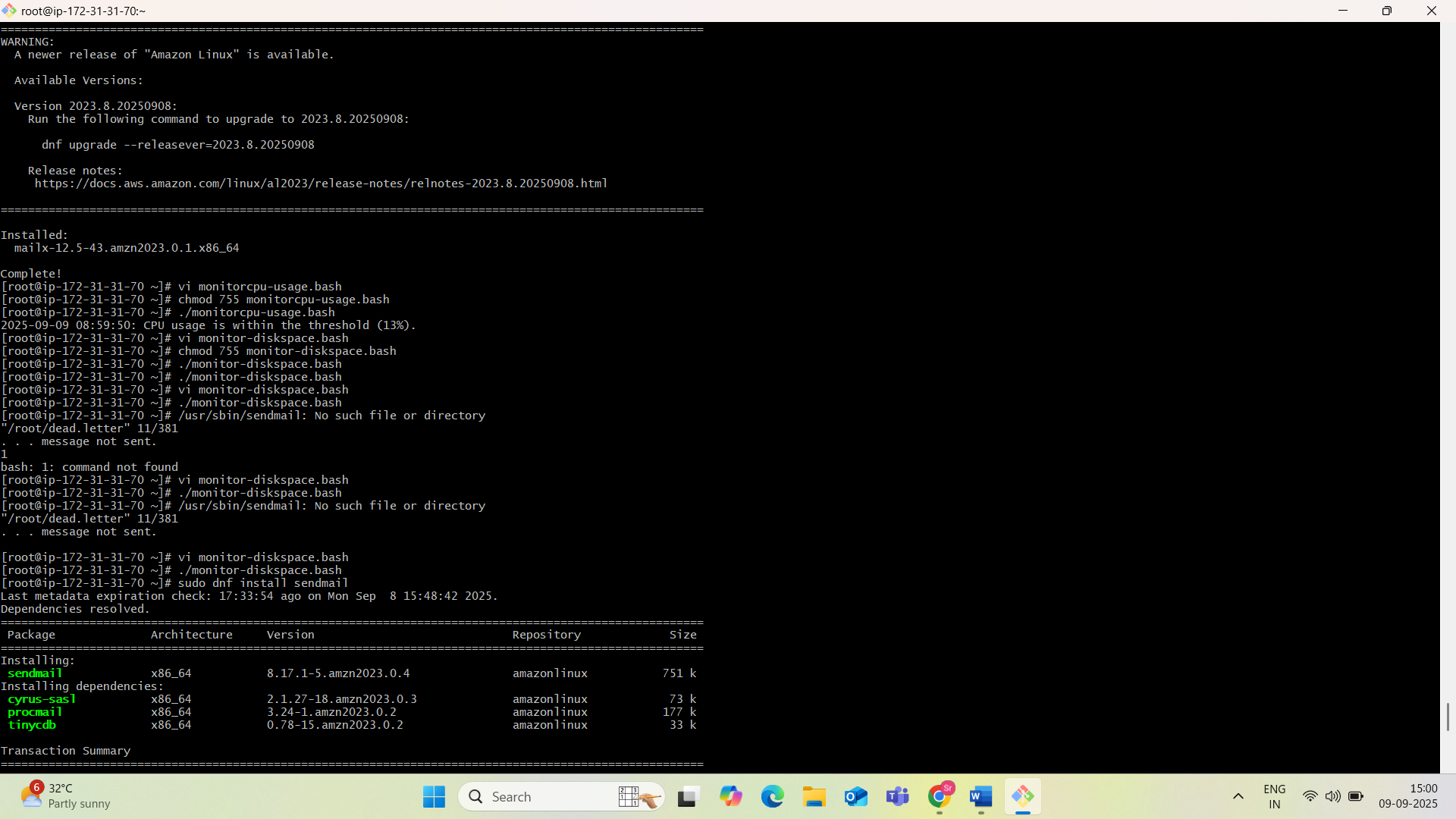
# Ensure 'mailx' or 'mailutils' is installed on your system for this to work.

echo "$EMAIL\_BODY" | mail -s "$EMAIL\_SUBJECT" "$EMAIL\_RECIPIENT"

fi

done





12)**. Bash Script to Monitor Memory and Send Email if > 80%**

#!/bin/bash

# Configuration

THRESHOLD=80 # Memory usage percentage threshold

RECIPIENT\_EMAIL="your\_email@example.com" # Email address to send alerts

SENDER\_EMAIL="server\_alert@example.com" # Sender email address

HOSTNAME=$(hostname)

# Get memory usage

# free -m provides memory in megabytes

# awk processes the output to calculate percentage

MEM\_INFO=$(free -m | awk 'NR==2{printf "%.0f", $3\*100/$2}')

# Check if memory usage exceeds the threshold

if (( MEM\_INFO > THRESHOLD )); then

SUBJECT="Memory Alert on $HOSTNAME: Usage Exceeds $THRESHOLD%"

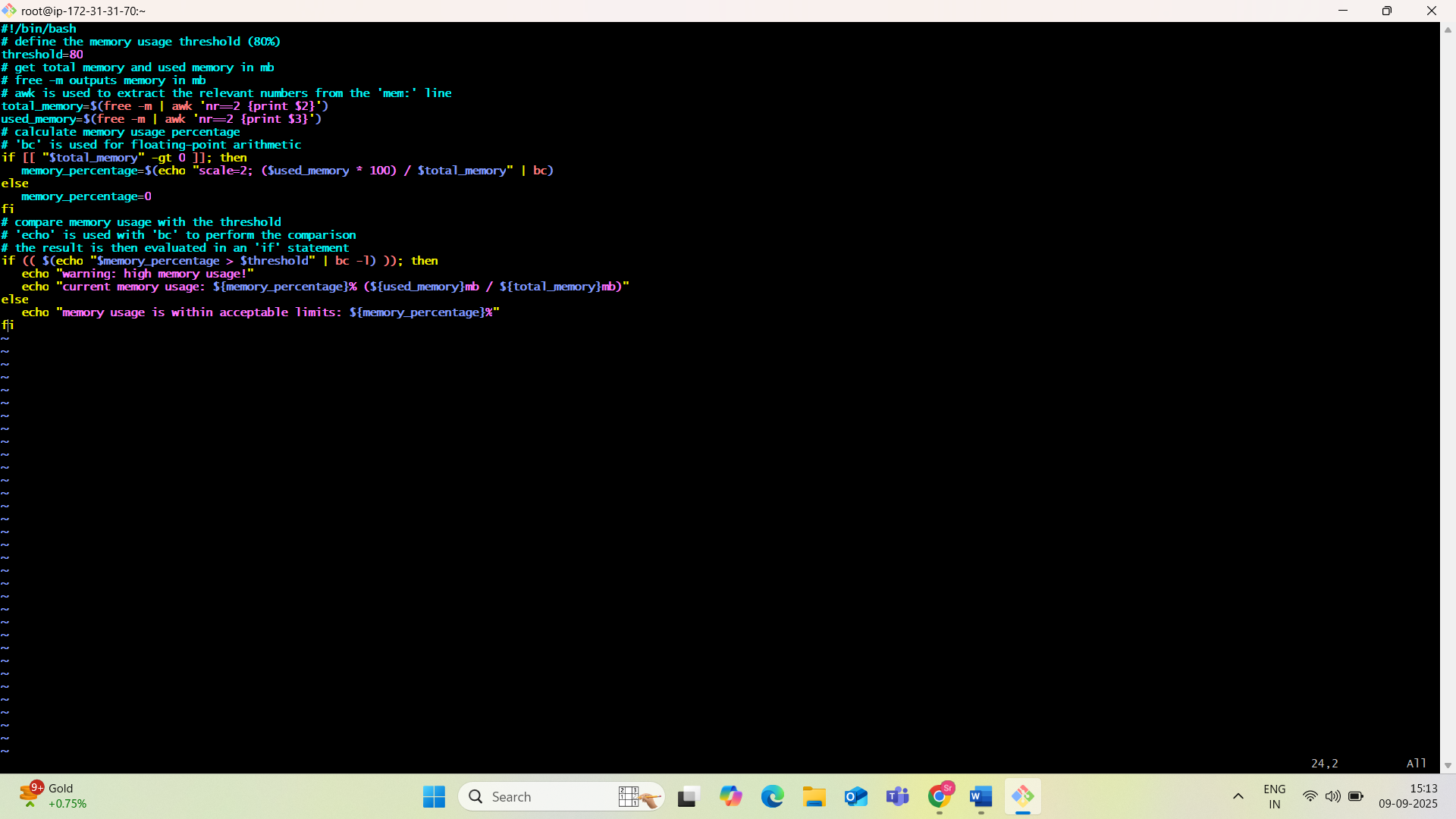
BODY="Current memory usage on $HOSTNAME is $MEM\_INFO%."

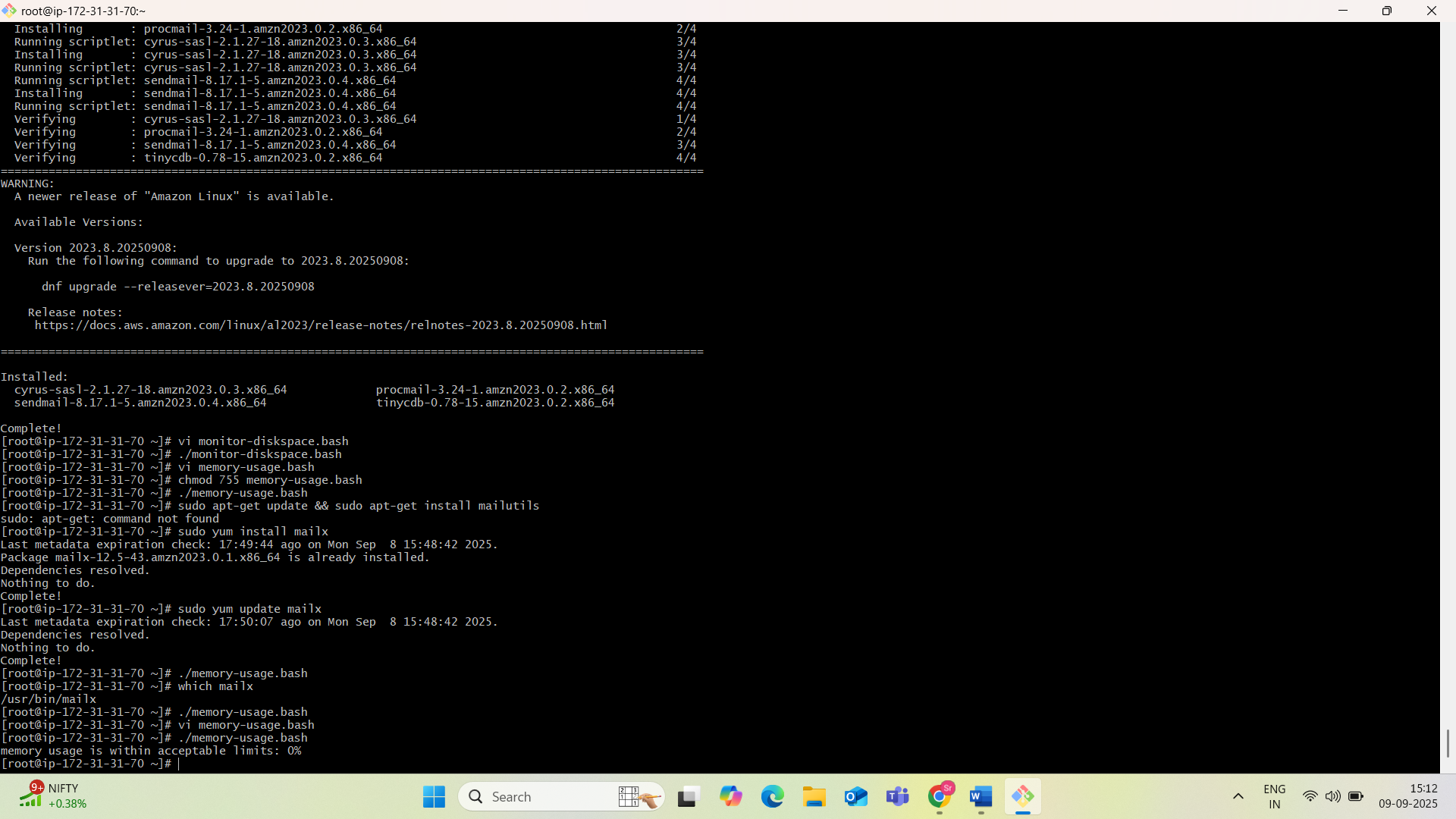
# Send email

# mail command is typically used for sending emails

echo "$BODY" | mail -s "$SUBJECT" -r "$SENDER\_EMAIL" "$RECIPIENT\_EMAIL"

fi





**✅ Email Setup for Scripts**

To make the email sending scripts work, configure your server to send mail using tools like:

* mailx or mailutils
* Set up Postfix or use an SMTP relay (e.g., AWS SES, Gmail SMTP)

**✅ Summary of Scripts**

| **Task** | **Script File** |
| --- | --- |
| Install Nginx | install\_nginx.bash |
| Install Tomcat | install\_tomcat.sh |
| Check Nginx | check\_nginx.sh |
| Calculator | calculator.sh |
| Directory Check | check\_directory.sh |
| Delete Last 3 Lines | delete\_lines.sh |
| Monitor CPU | monitor\_cpu.sh |
| Monitor Disk | monitor\_disk.sh |
| Monitor Memory | monitor\_memory.sh |