PRACTICAL LINUX AND SHELL SCRIPTING SKILLS

LINUX CLI PRACTICALS

This project catches a comprehensive hands-on session, showcasing essential Linux command-line operations for file/directory management, content manipulation, and system utilities, all vital for DevOps and administration.

LINUX BASICS-SETUP AND FILE MANAGEMENT

Linux commands: directory creation(mkdir),navigation(cd), file creation(touch), content display (cat), and merging (cat >).Ls is used to verify changes.

```
ubuntu@ip-172-31-83-70:~$ mkdir LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~$ ls
LINUX_COMMANDS        aws_resource_tracker.sh   back       backups  hello.txt       shell-scripting-For-Devops  tws
aws_resource_tracker  aws_resources_tracker.sh  backup.sh  data     nano.6678.save  snap
ubuntu@ip-172-31-83-70:~$ cd LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cd
ubuntu@ip-172-31-83-70:~$ cd shell-scripting-For-Devops
ubuntu@ip-172-31-83-70:~/shell-scripting-For-Devops$ ls
README.md  day01  day02
ubuntu@ip-172-31-83-70:~/shell-scripting-For-Devops$ cd
ubuntu@ip-172-31-83-70:~$ cd LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ touch file1.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat>file2.txt
Hello Friends!
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt  file2.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file2.txt
Hello Friends!
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat>file3.txt
Welcome to Linux / Unix Commandsubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt  file2.txt  file3.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file2.txt
Hello Friends!
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file3.txt
Welcome to Linux / Unix Commandsubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file2.txt file3.txt>filemerge.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt  file2.txt  file3.txt  filemerge.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls *.txt
file1.txt  file2.txt  file3.txt  filemerge.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt  file2.txt  file3.txt  filemerge.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cd
ubuntu@ip-172-31-83-70:~$ ls
LINUX_COMMANDS        aws_resource_tracker.sh   back       backups  hello.txt       shell-scripting-For-Devops  tws
aws_resource_tracker  aws_resources_tracker.sh  backup.sh  data     nano.6678.save  snap
ubuntu@ip-172-31-83-70:~$ ls *.c
ls: cannot access '*.c': No such file or directory
ubuntu@ip-172-31-83-70:~$
```

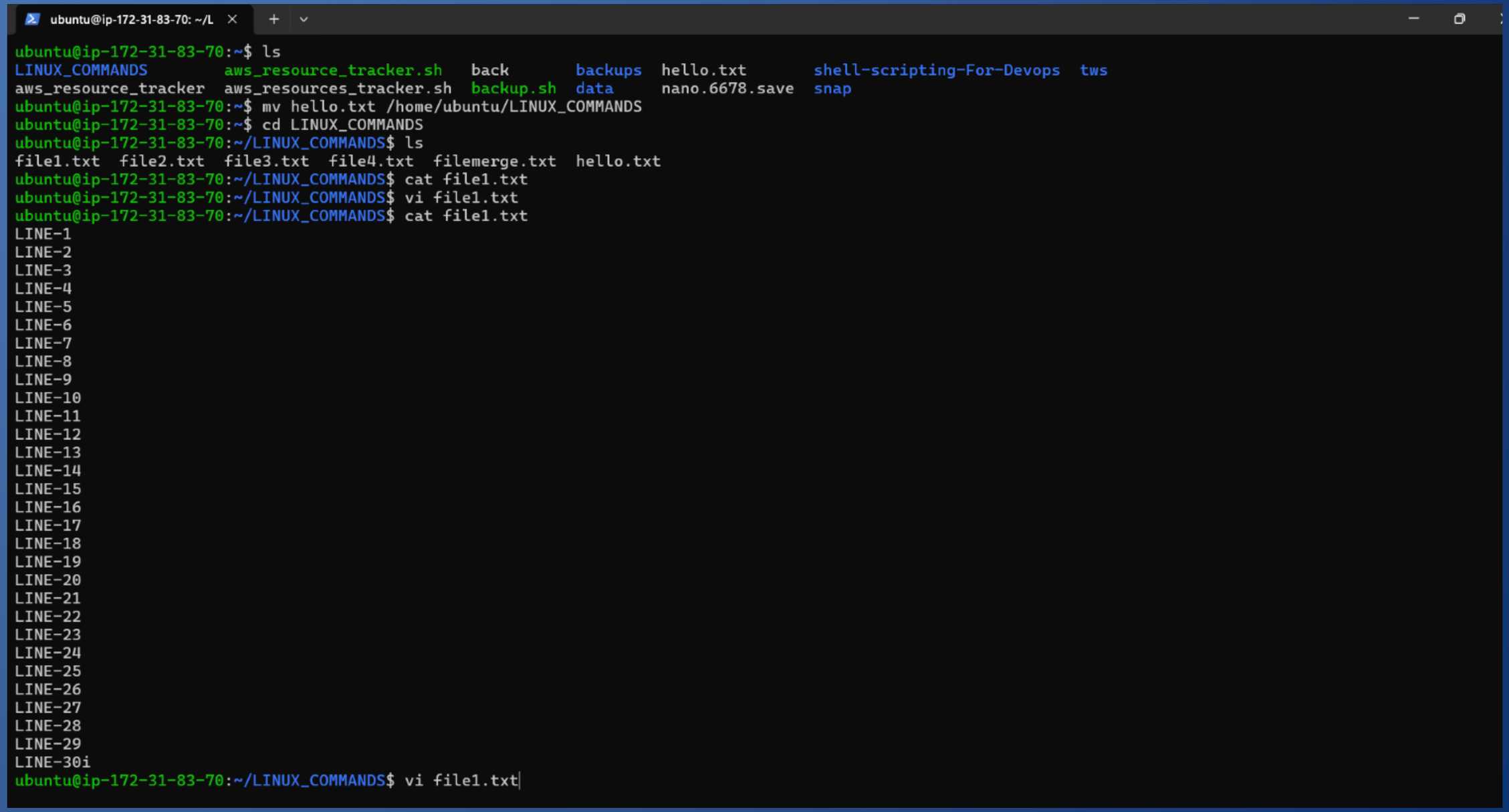
ADVANCED FILE OPERATIONS & ENVIRONMENT COMMANDS

This page covers further file manipulations and system information commands. It shows copying files (cp), moving files (mv), and displaying the current working directory (pwd). It also includes attempts to list specific file types (ls *.sh, ls *.py, ls *.txt) and initial vi edits to file1.txt are also indicated.

```
ubuntu@ip-172-31-83-70:~$ ls
LINUX_COMMANDS        aws_resource_tracker.sh   back       backups  hello.txt       shell-scripting-For-Devops  tws
aws_resource_tracker  aws_resources_tracker.sh  backup.sh  data     nano.6678.save  snap
ubuntu@ip-172-31-83-70:~$ ls *.sh
aws_resource_tracker.sh  aws_resources_tracker.sh  backup.sh
ubuntu@ip-172-31-83-70:~$ ls *.py
ls: cannot access '*.py': No such file or directory
ubuntu@ip-172-31-83-70:~$ ls *.txt
hello.txt
ubuntu@ip-172-31-83-70:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-83-70:~$ cd LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ PWD
PWD: command not found
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ pwd
/home/ubuntu/LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cpfile3.txt file4.txt
cpfile3.txt: command not found
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cp file3.txt file4.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt  file2.txt  file3.txt  file4.txt  filemerge.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file3.txt
Welcome to Linux / Unix Commandsubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file4.txt
Welcome to Linux / Unix Commandsubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cd
ubuntu@ip-172-31-83-70:~$ ls
LINUX_COMMANDS        aws_resource_tracker.sh   back       backups  hello.txt       shell-scripting-For-Devops  tws
aws_resource_tracker  aws_resources_tracker.sh  backup.sh  data     nano.6678.save  snap
ubuntu@ip-172-31-83-70:~$ mv hello.txt /home/ubuntu/LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~$ cd LINUX_COMMANDS
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ ls
file1.txt  file2.txt  file3.txt  file4.txt  filemerge.txt  hello.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file1.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ vi file1.txt
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ cat file1.txt
LINE-1
LINE-2
LINE-3
LINE-4
LINE-5
LINE-6
LINE-7
```

FILE EDITING & CONTENT UPDATE

Moving hello.txt, navigating into LINUX_COMMANDS, and then extensively using the vi editor to add lines ("LINE-1" to "LINE-30") to file1.txt, verifying the updated content with cat.



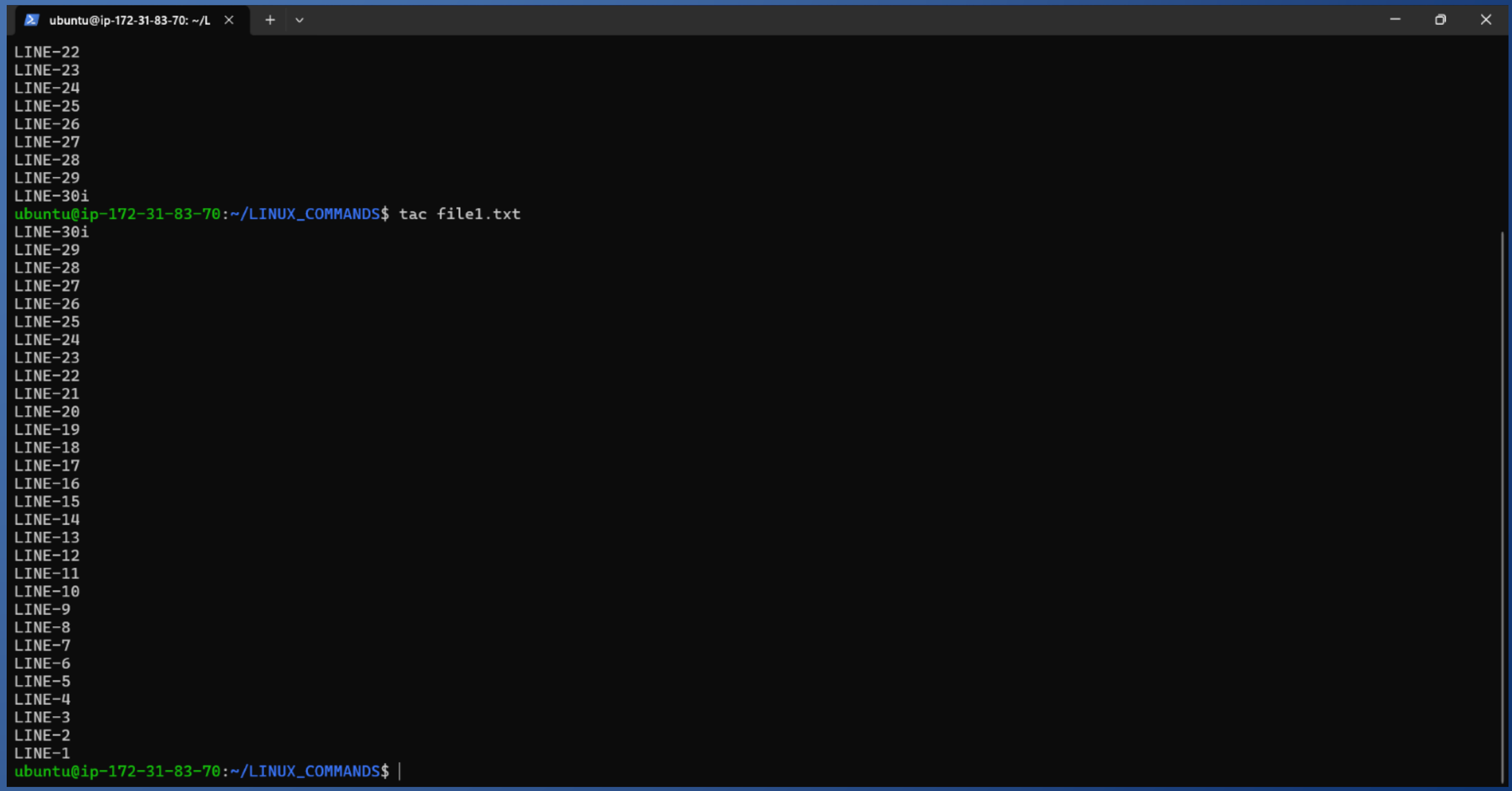
SPECIALIZED FILE CONTENT DISPLAY

Demonstrates commands for specific file content viewing: head (first 10 lines), tail (last 10 lines), and tac (entire content reversed), all applied to file1.txt.f body text

```
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ head file1.txt
LINE-1
LINE-2
LINE-3
LINE-4
LINE-5
LINE-6
LINE-7
LINE-8
LINE-9
LINE-10
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ tail file1.txt
LINE-21
LINE-22
LINE-23
LINE-24
LINE-25
LINE-26
LINE-27
LINE-28
LINE-29
LINE-30i
ubuntu@ip-172-31-83-70:~/LINUX_COMMANDS$ tac file1.txt
LINE-30i
LINE-29
LINE-28
LINE-27
LINE-26
LINE-25
LINE-24
LINE-23
LINE-22
LINE-21
LINE-20
LINE-19
LINE-18
LINE-17
LINE-16
LINE-15
LINE-14
LINE-13
```

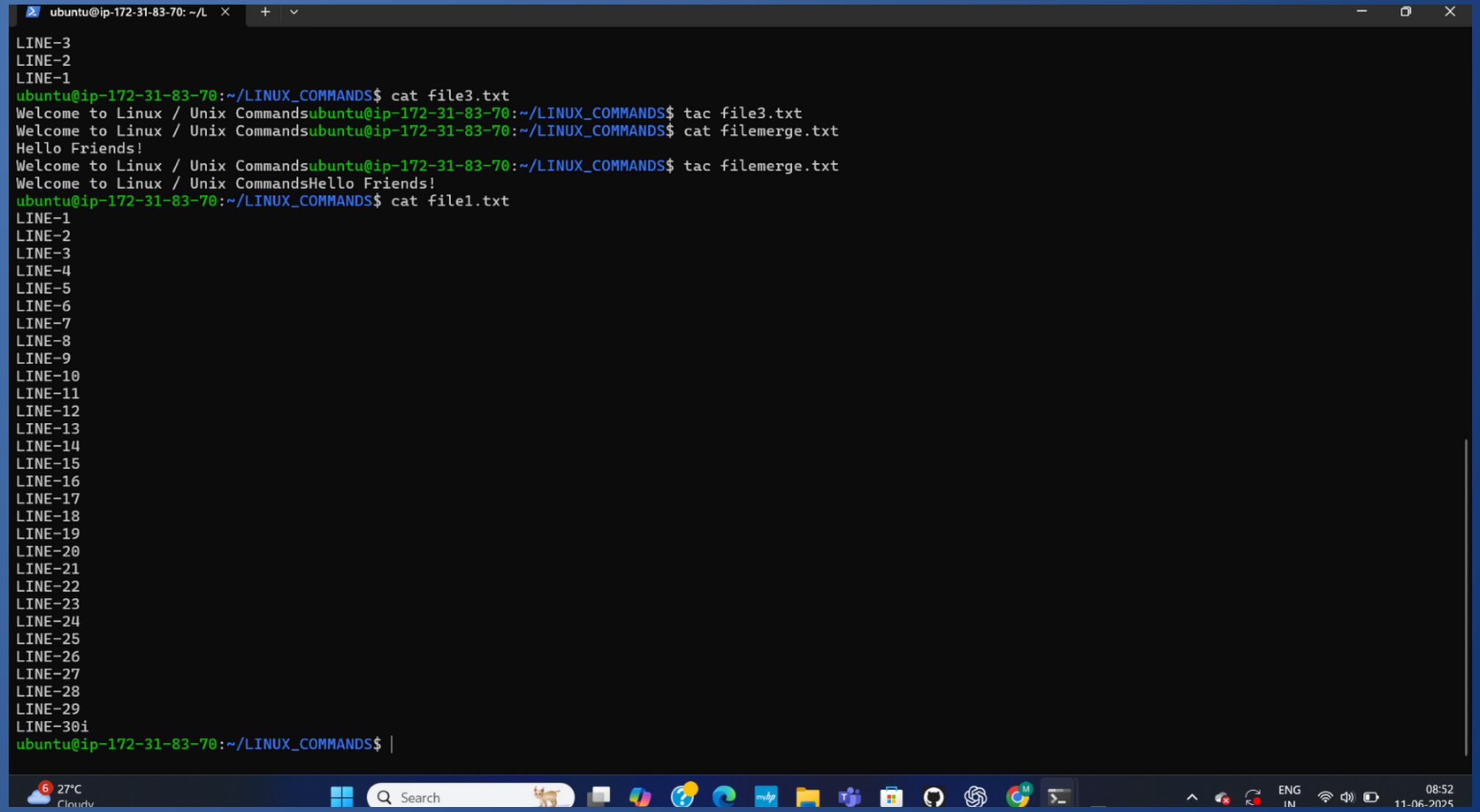
FULL REVERSED FILE DISPLAY

The tac command output, displaying the complete content of file1.txt in reverse order, from "LINE-30" back to "LINE-1".



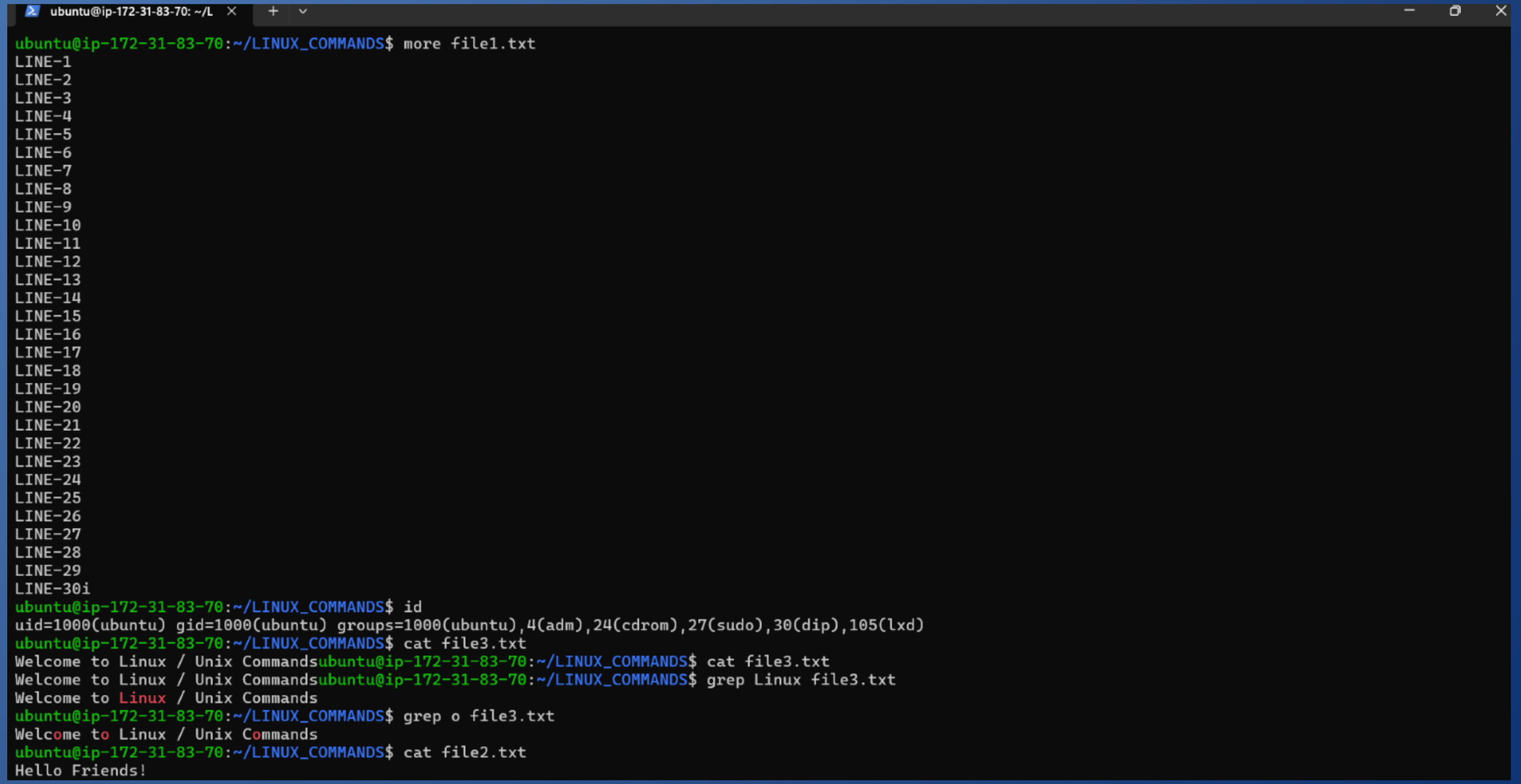
_TAC ON OTHER FILES & FULL FILE1.TXT VIEW

This Further illustrates the tac command on file3.txt and filemerge.txt, showing how it reverses their conte
The cat command is also used to display the full content of file1.txt.



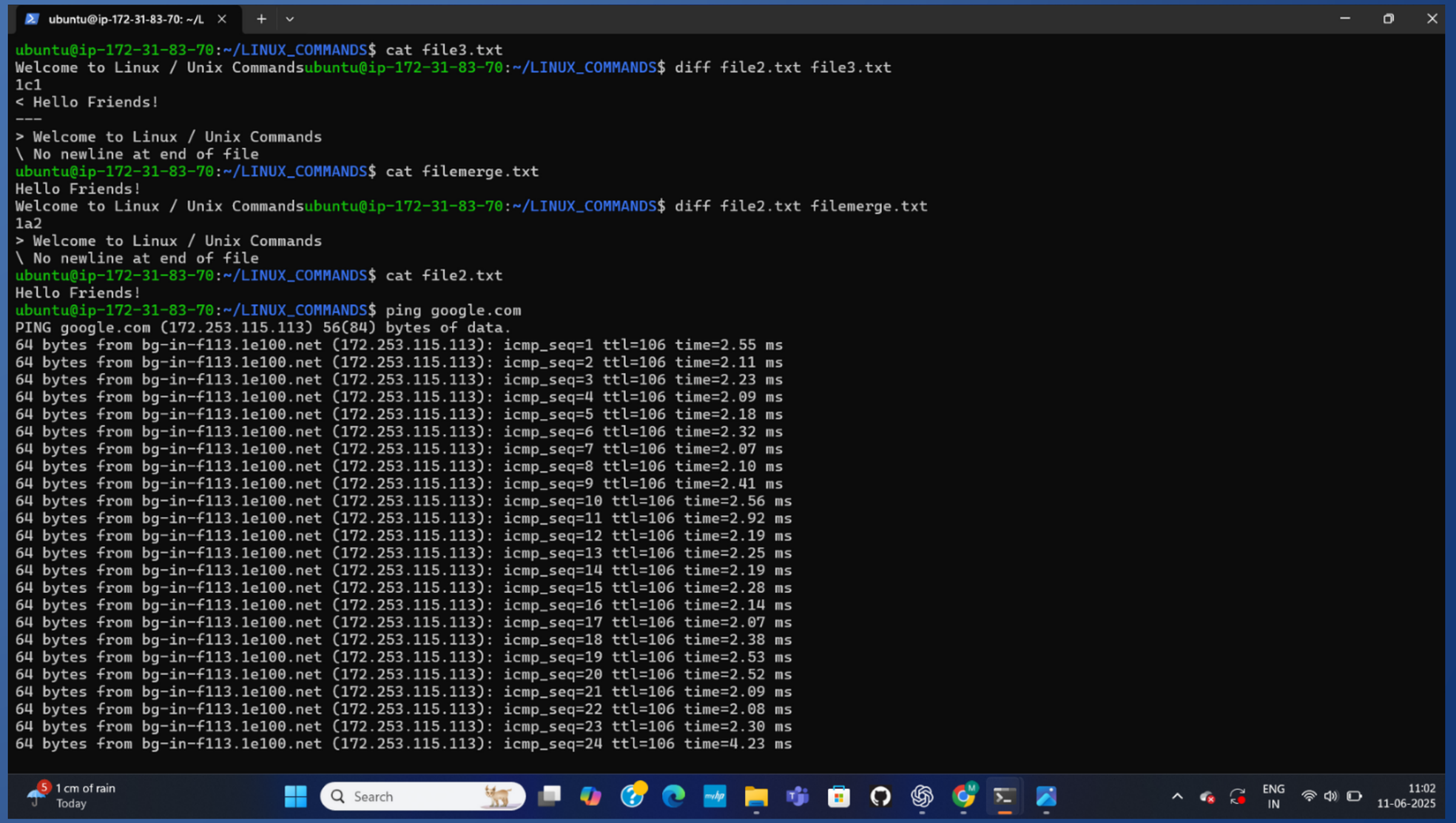
USER AND SEARCH UTILITIES

In this it introduces commands for system information and content searching. It shows more for paginated file viewing of file1.txt, id for user and group information (uid, gid, groups of ubuntu), and grep for searching specific patterns ("Linux", "o") within file3.txt.



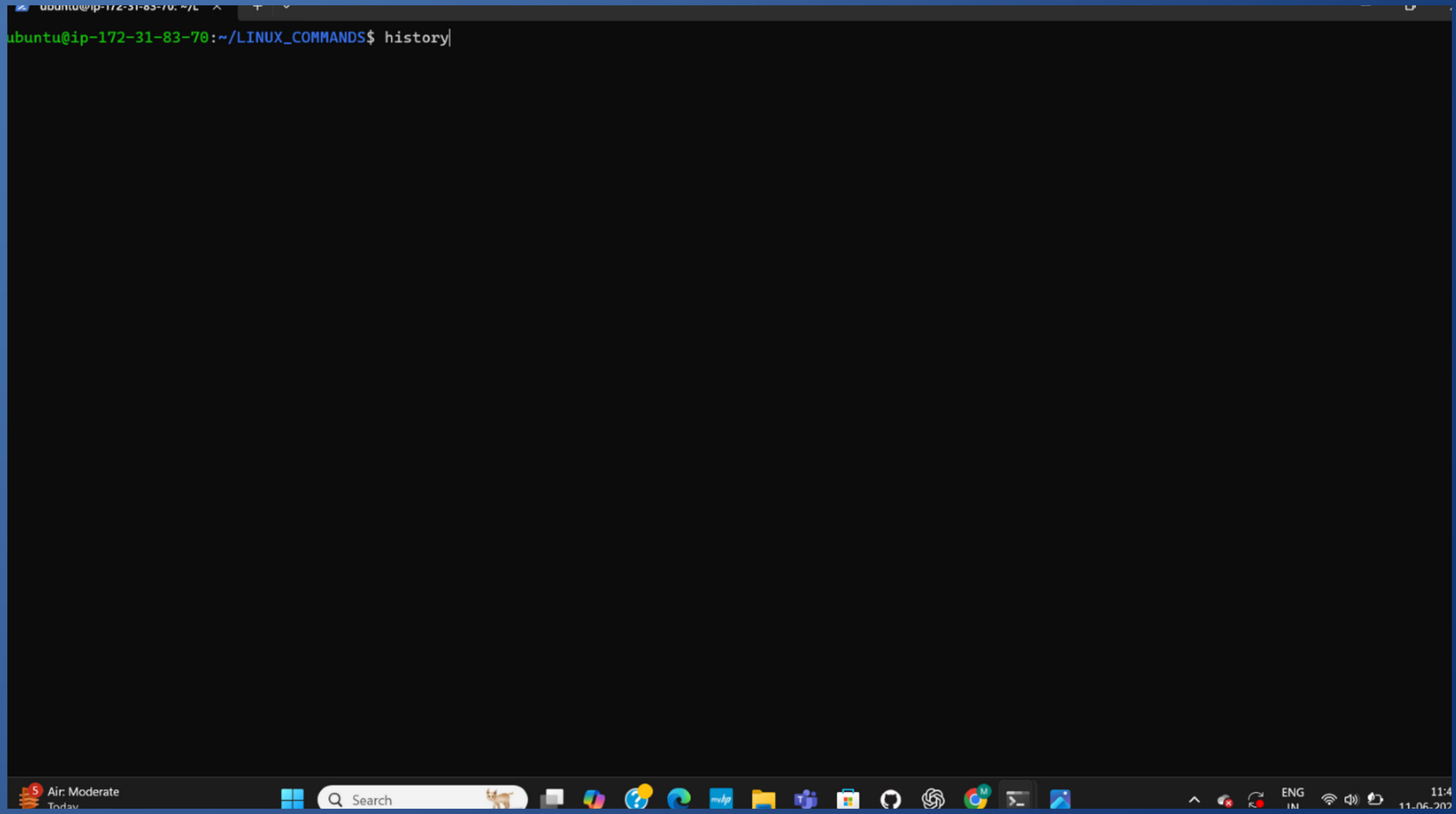
FILE COMPARISON AND NETWORK TESTING

This focuses on comparing file contents (diff file2.txt file3.txt and diff file2.txt filemerge.txt) and network diagnostics (ping google.com), showing differences between files and successful ping responses.



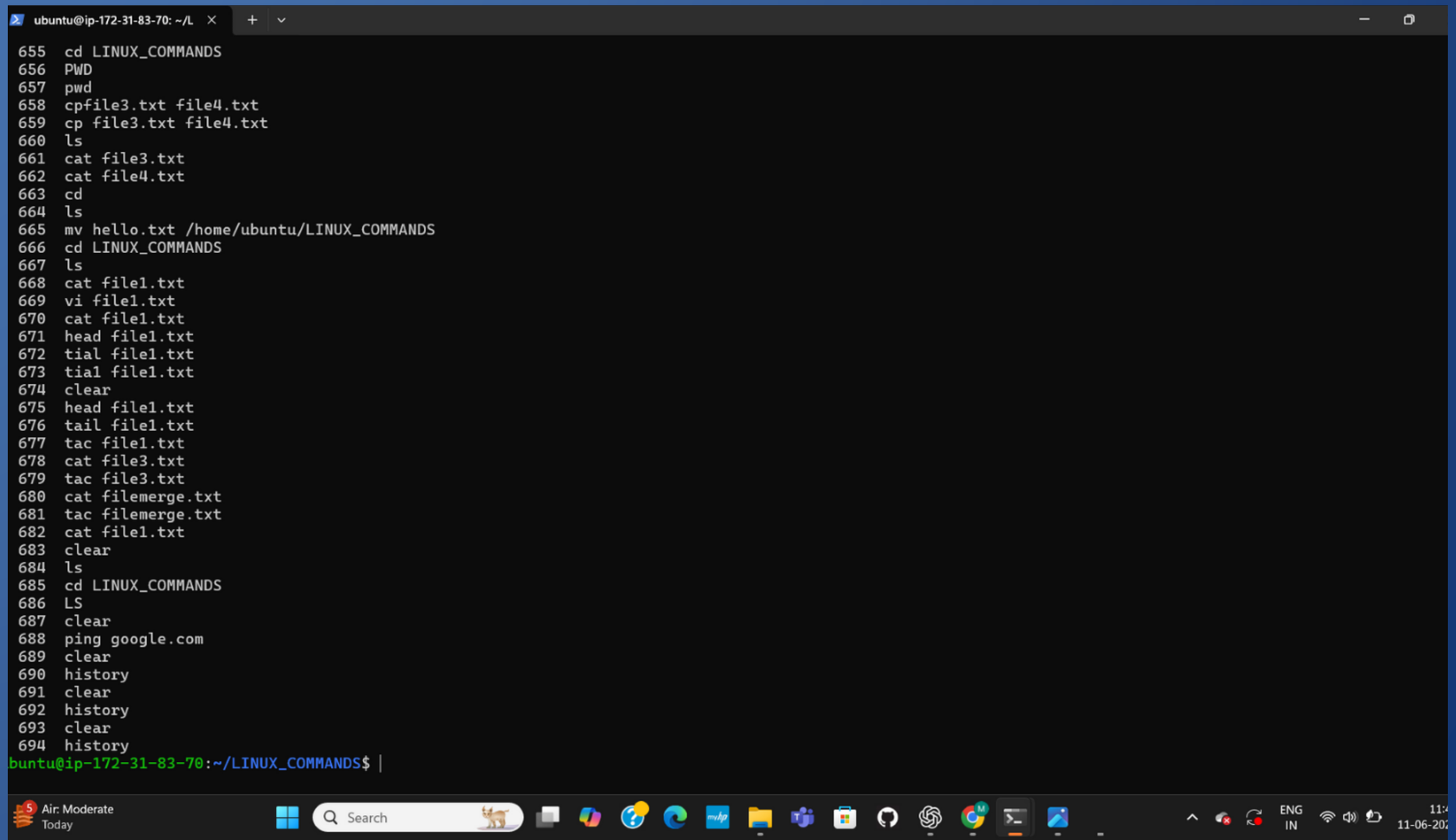
COMMAND HISTORY INITIATION

It shows the execution of the history command, indicating that a list of previously entered commands is being requested and will be displayed.



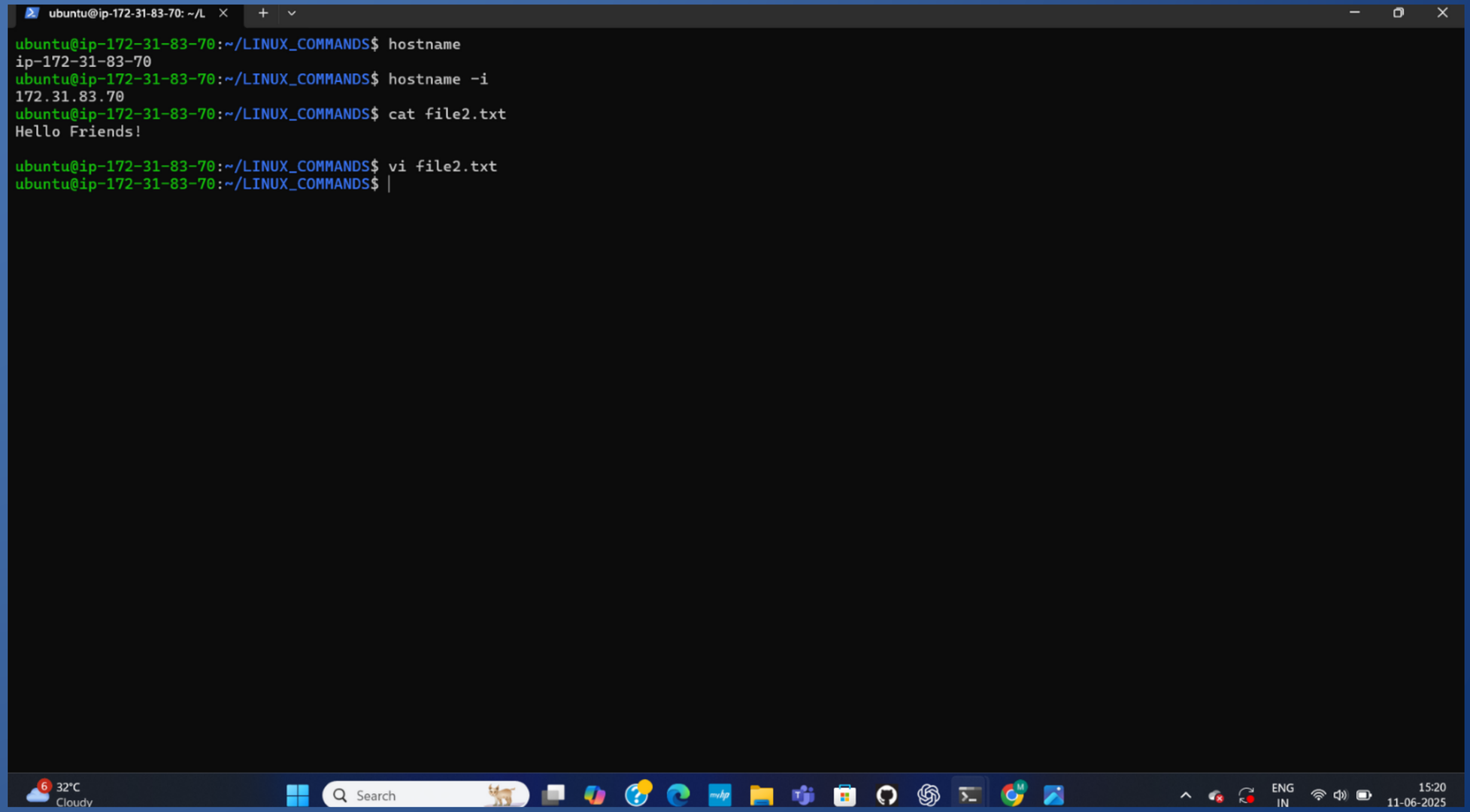
COMMAND HISTORY OUTPUT

It presents a detailed output of the history command, listing numerous commands executed in the session with their corresponding command numbers. This includes commands like cd, pwd, cp, ls, cat, vi, head, tail, tac, clear, ping, and history itself.



HOSTNAME INFORMATION AND FILE VIEWING

This displays commands related to system identification and basic file viewing. It shows hostname to get the system's name (ip-172-31-83-70) and hostname -i for its IP address (172.31.83.70). Additionally, it shows cat file2.txt to display the content "Hello Friends!" and vi file2.txt for editing.

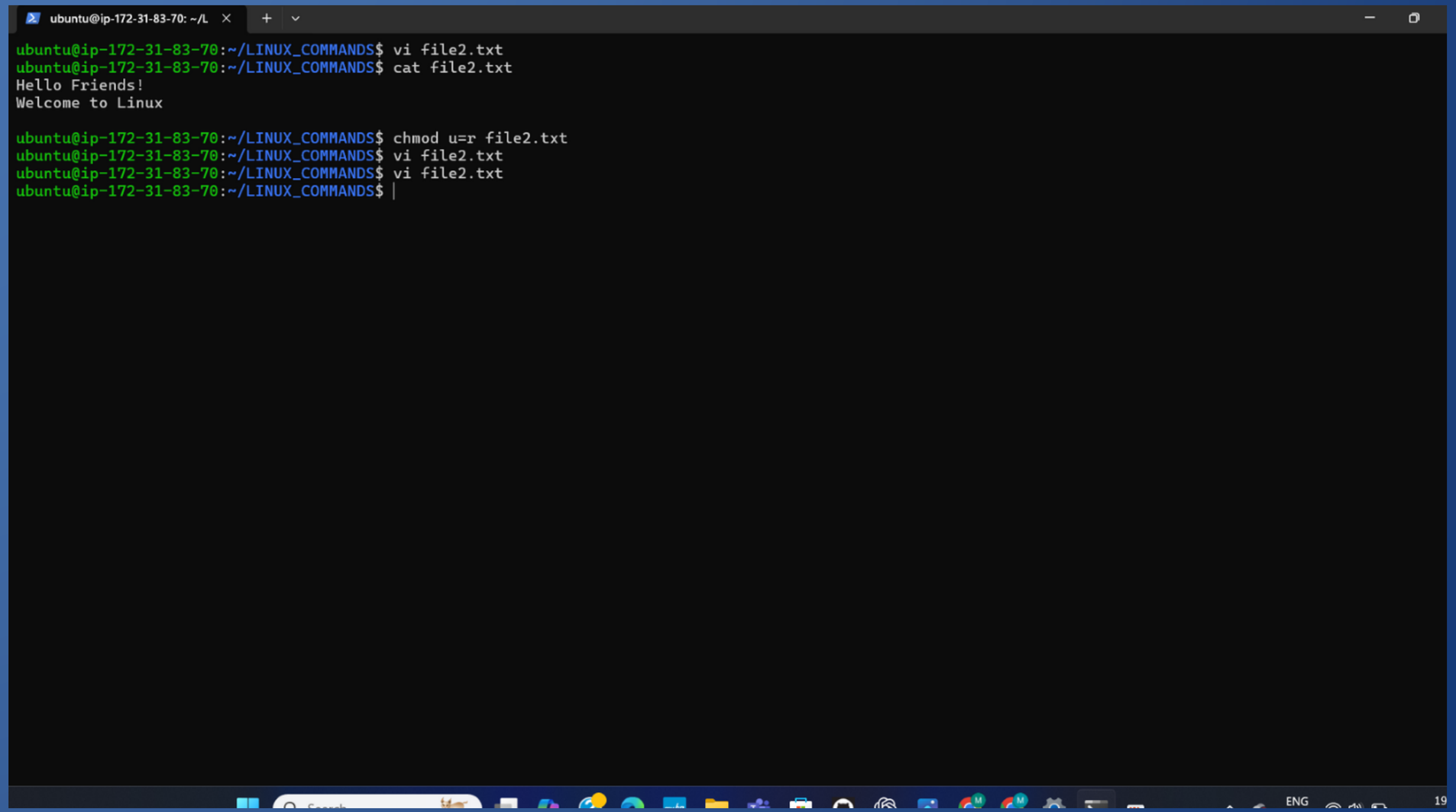


VI EDITOR VIEW OF FILE2.TXT

it shows a visual representation of file2.txt within the vi editor, displaying its initial content "Hello Friends!" and "Welcome to Linux".

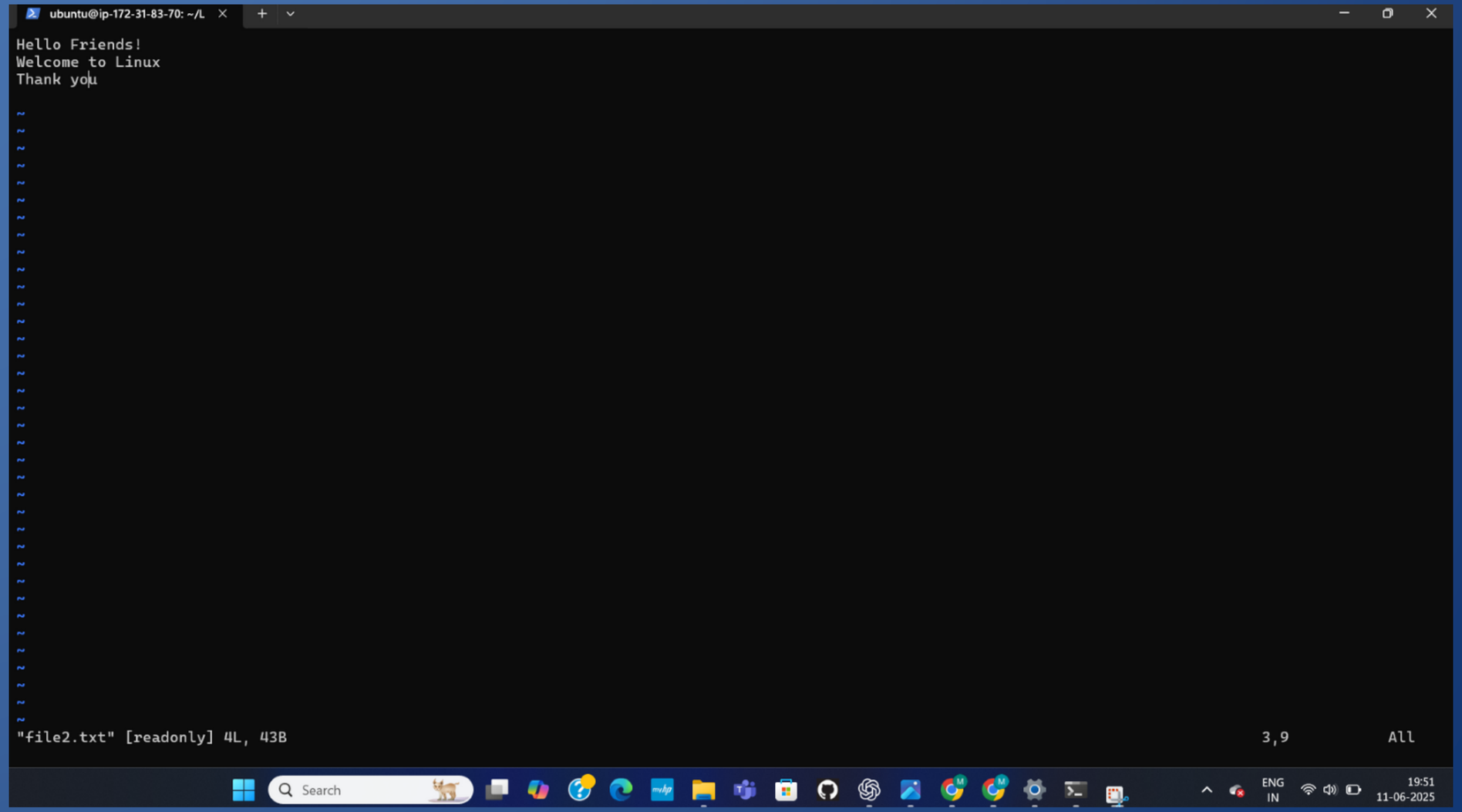
FILE PERMISSIONS (ATTEMPTED) AND VI EXIT

This shows an attempted chmod command (chmod user), which is likely incorrect syntax for changing file permissions. It then shows vi file2.txt and cat file2.txt, confirming its content.



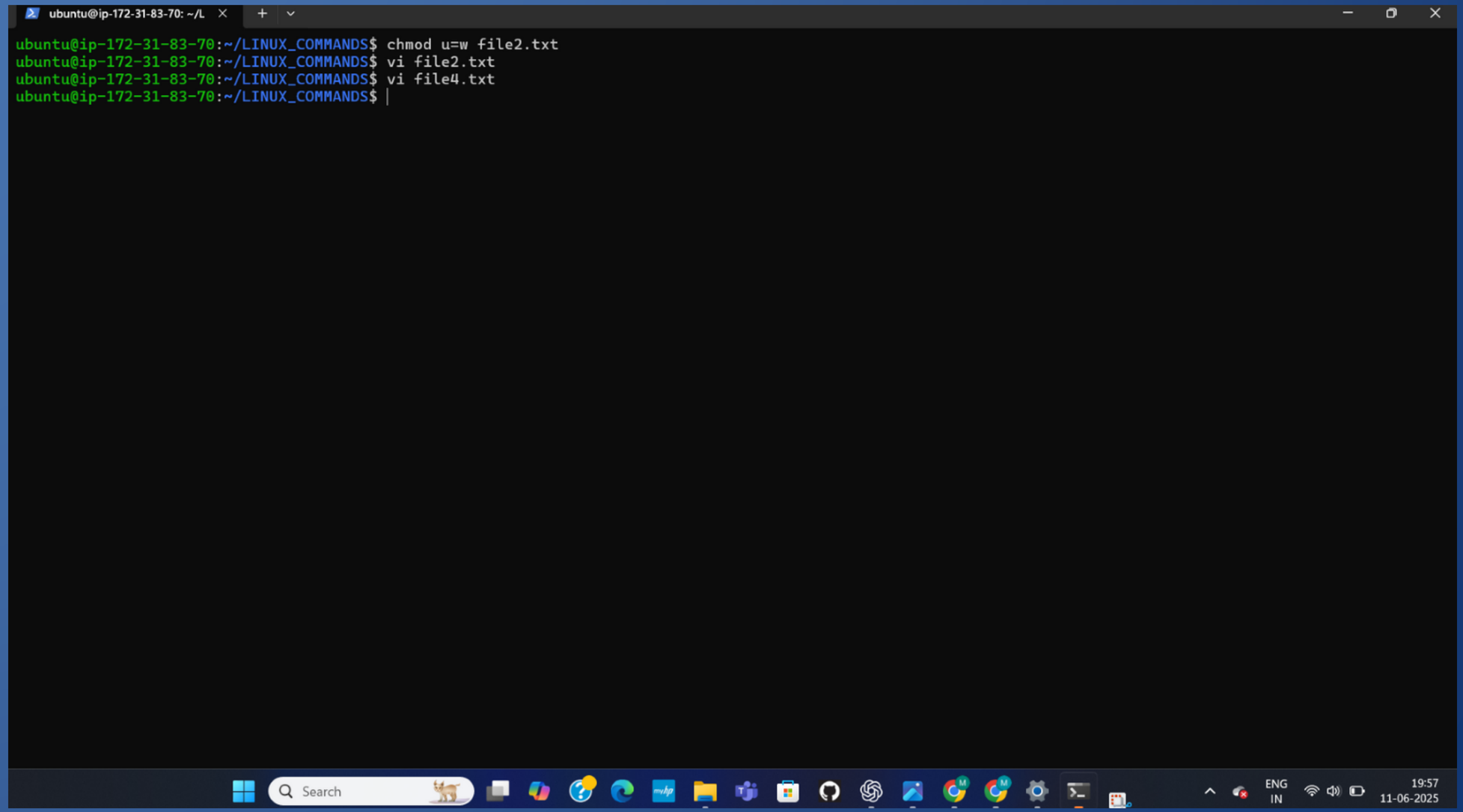
VI EDITOR WITH READ-ONLY NOTE

This displays file2.txt again within the vi editor, showing its content ("Hello Friends!", "Welcome to Linux", "Thank you") and a "readonly" note at the bottom.



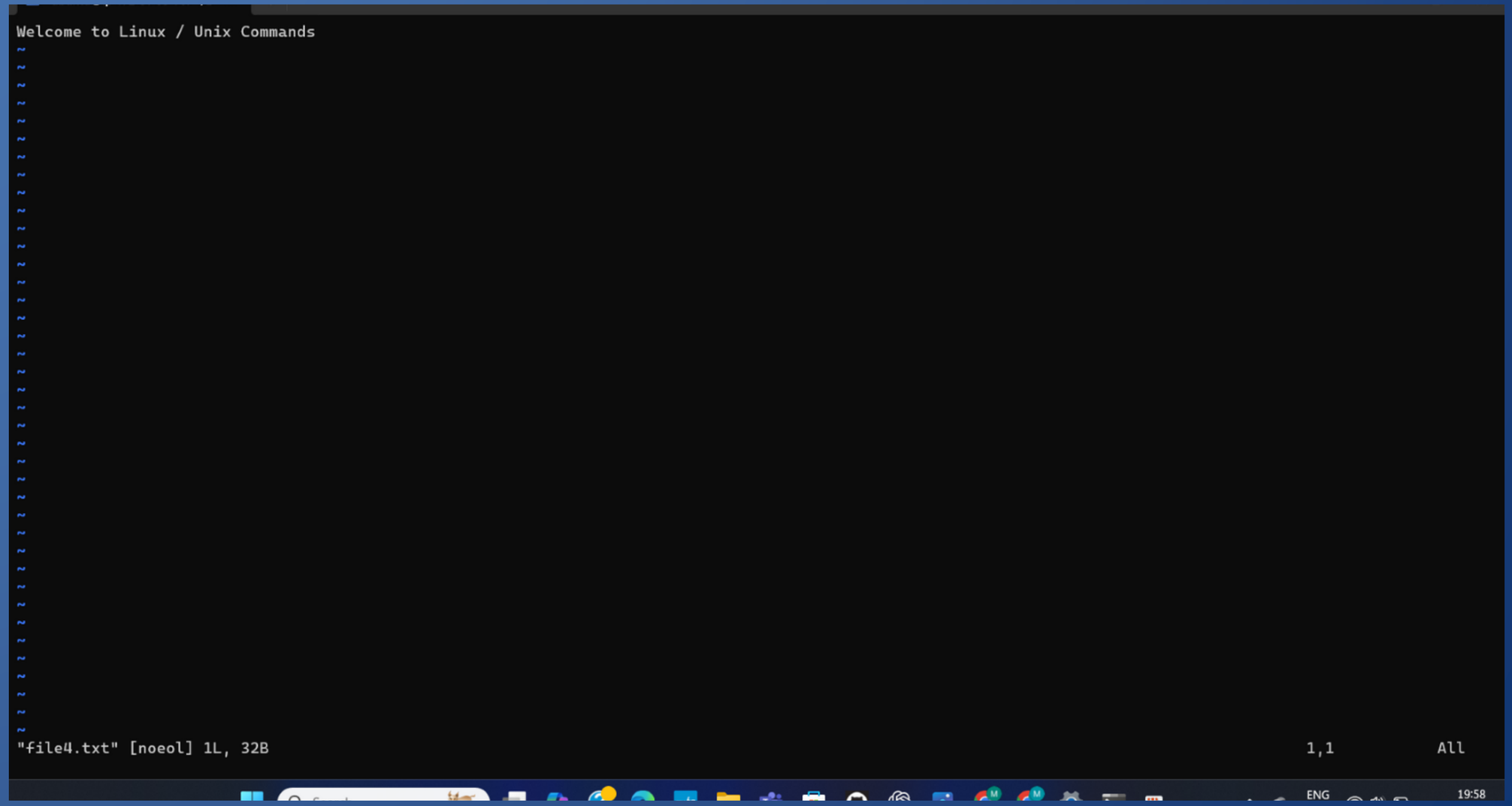
CHMOD CORRECTION & VI RETURN

This shows the corrected chmod command (chmod vcw file2.txt), followed by re-opening file2.txt in vi, and then cat file2.txt.



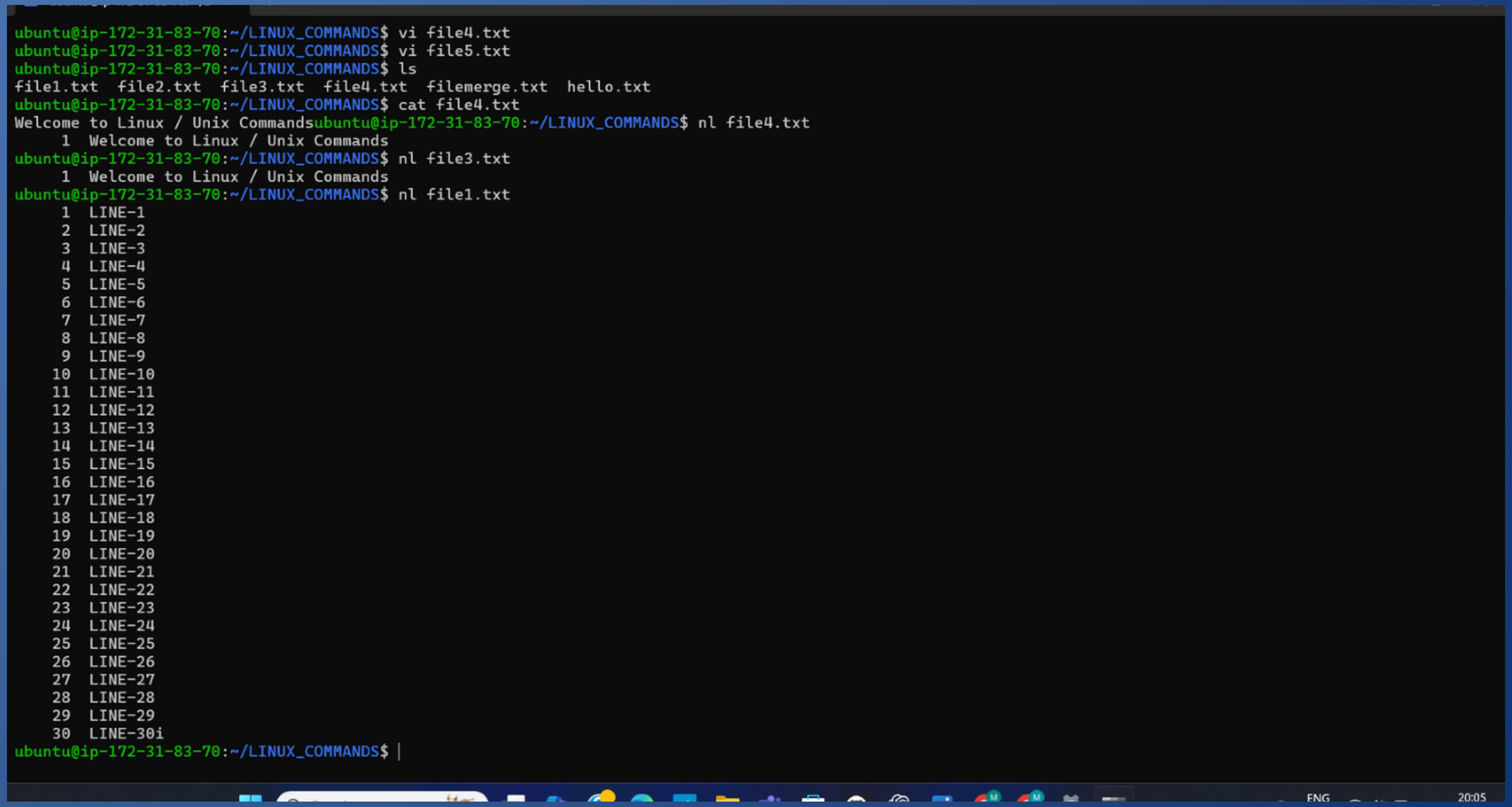
VI EDITOR FOR FILE4.TXT

This shows file4.txt opened in the vi editor, displaying its content "Welcome to Linux / Unix Commands", with a "readonly" note at the bottom.



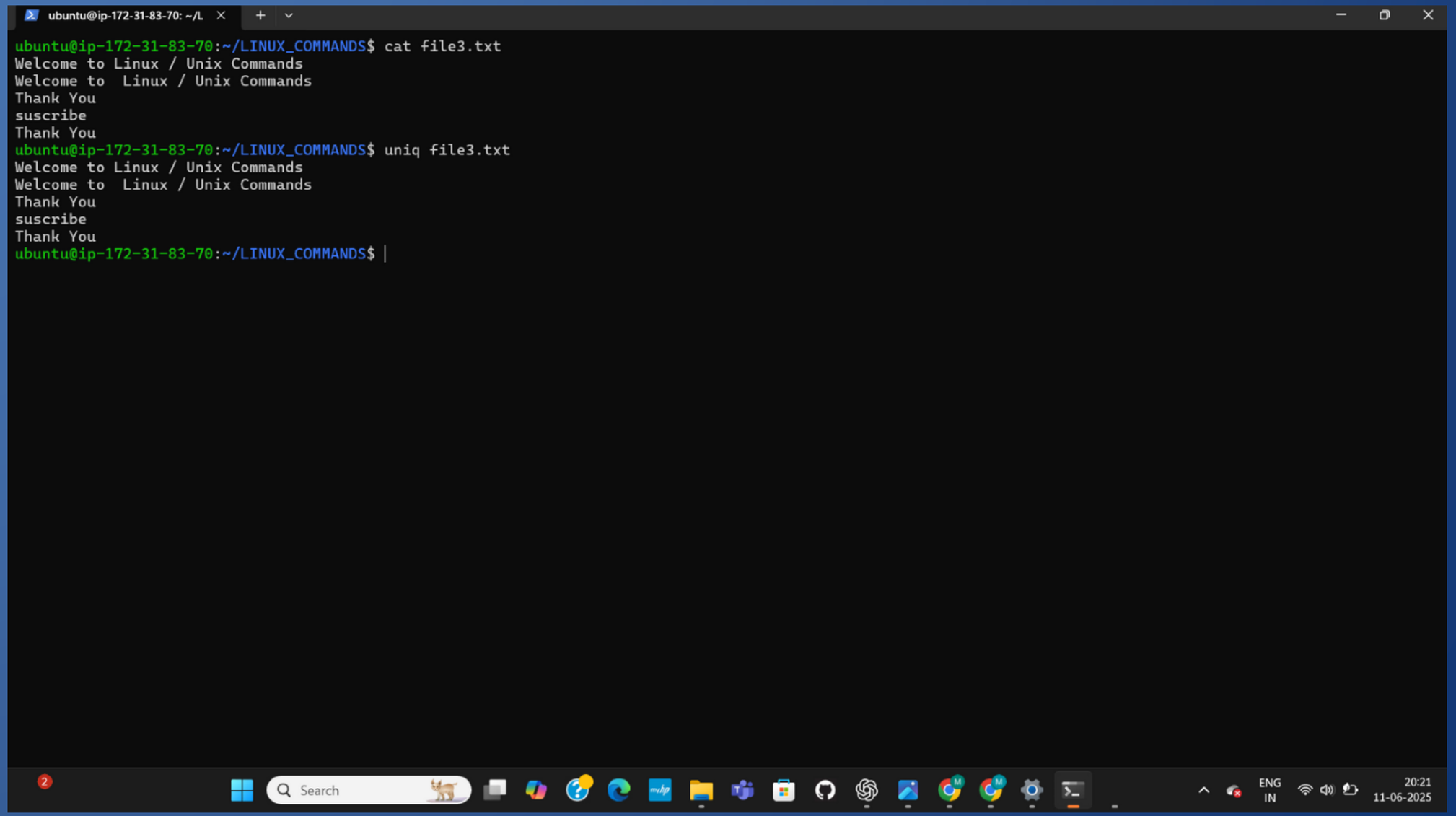
FILE CONTENT AND NL COMMAND

This shows vi file4.txt, ls, cat file4.txt, and nl file4.txt. The nl file1.txt command is also shown, displaying file1.txt with line numbers from 1 to 30.



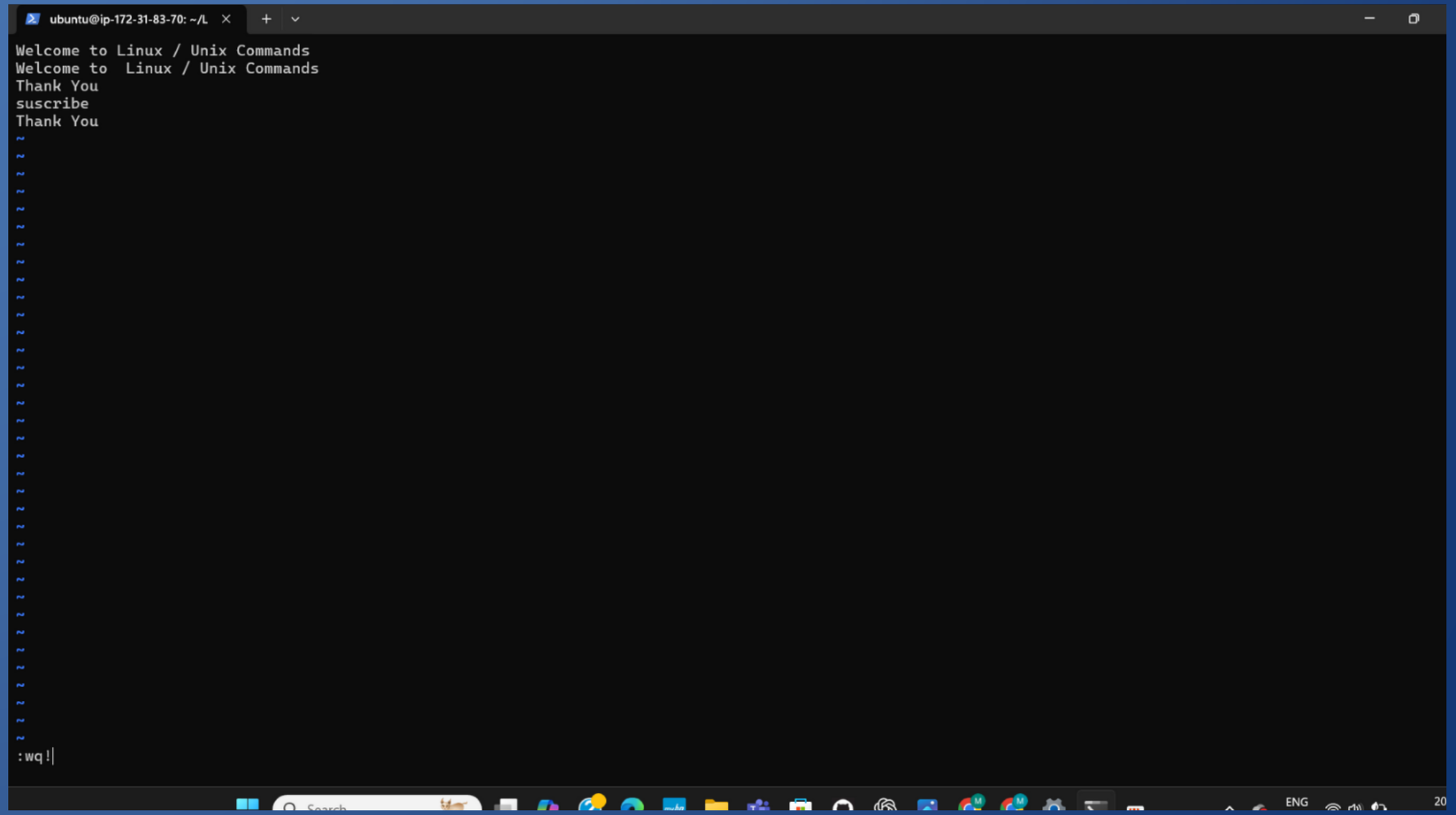
WORD COUNT AND FILE EDITING

This wc file3.txt to count words, lines, and characters in file3.txt. An attempt to cat file3aa results in "No such file or directory". wc file1.txt is also shown, giving character and line counts. The page ends with vi file1.txt.



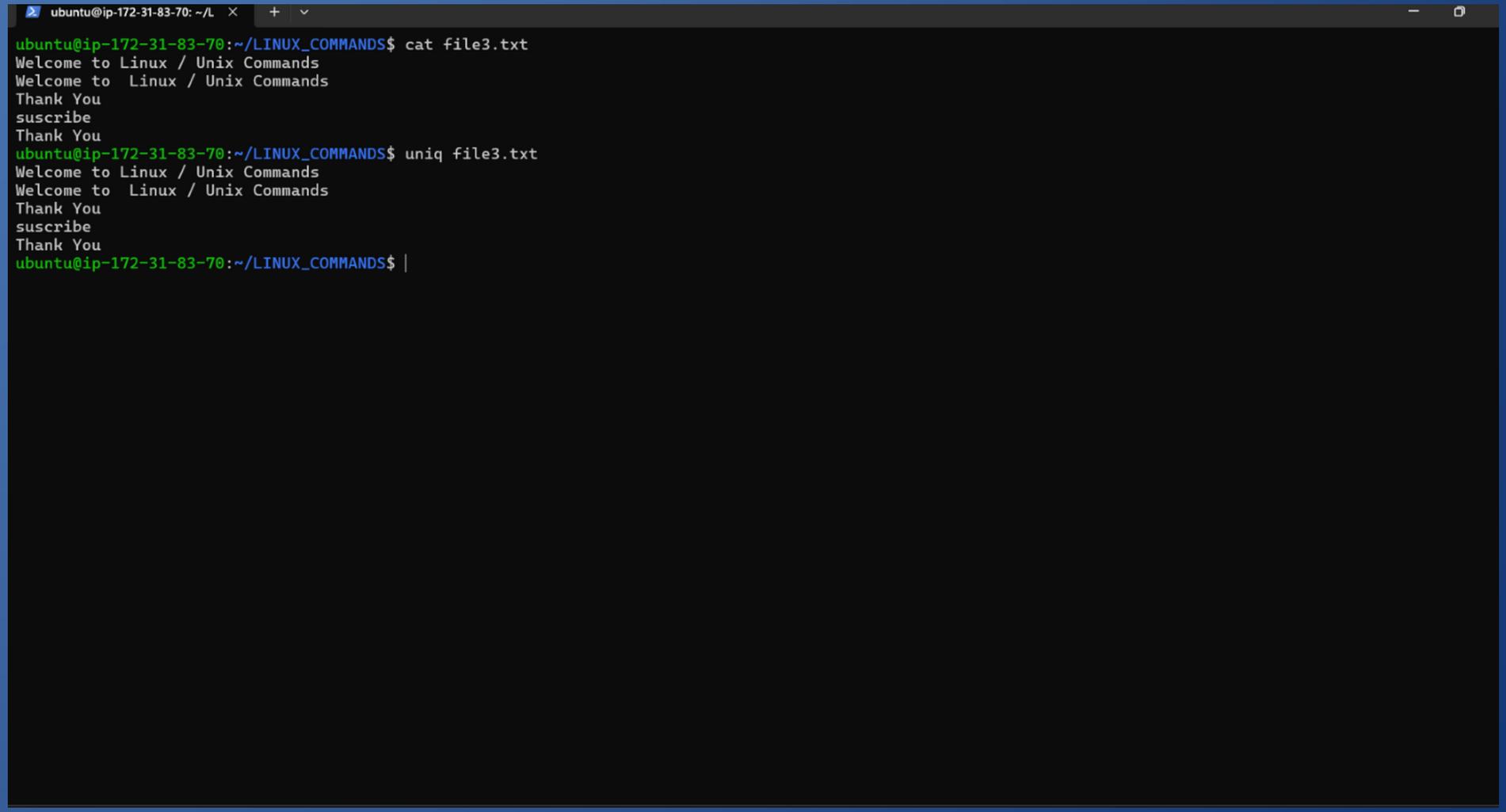
VI EDITOR WITH ADDITIONAL LINES

This displays file1.txt within the vi editor, showing content "Welcome to Linux / Unix Commands", "Thank You", and "suscribe".



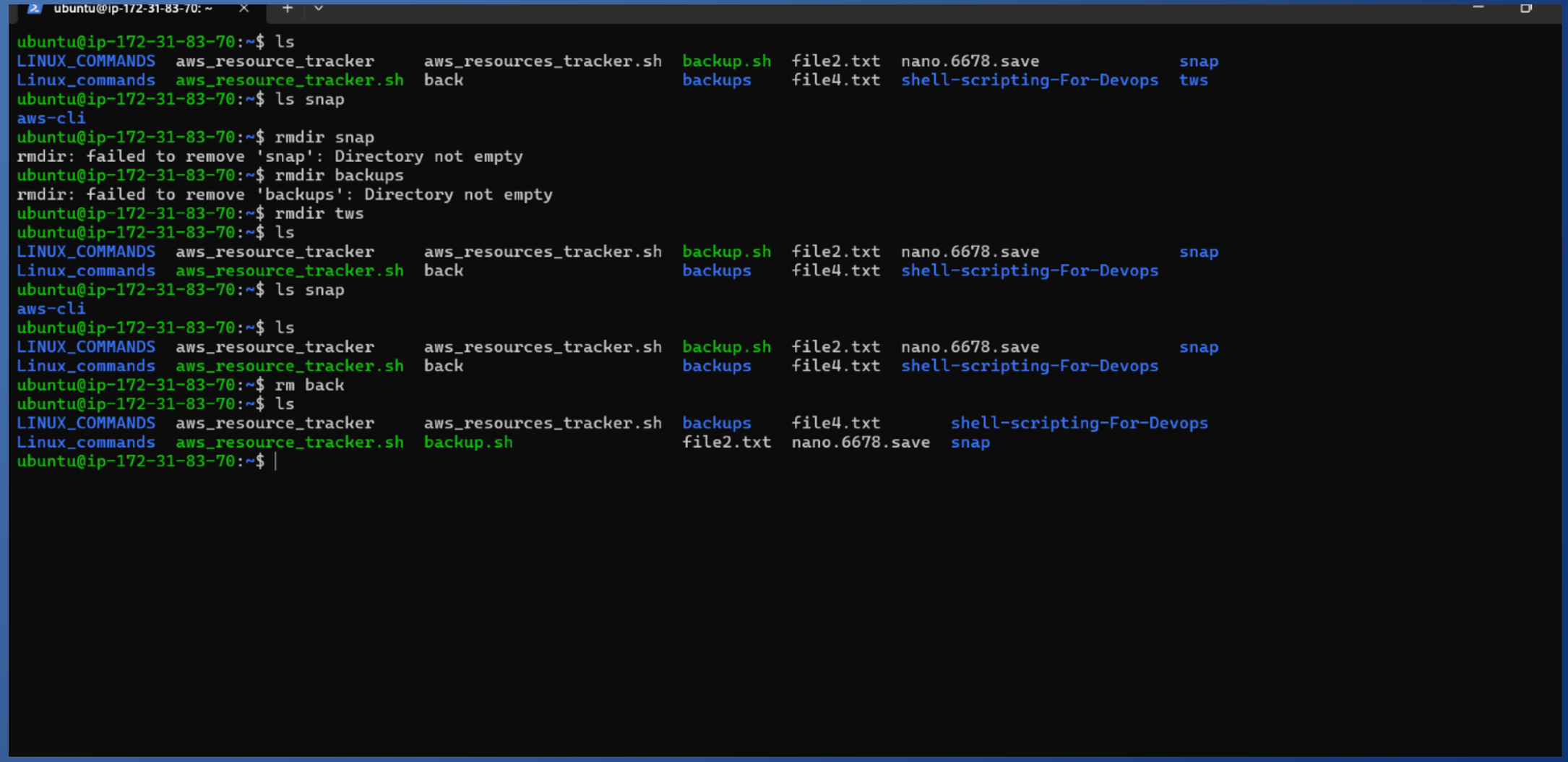
UNIQ COMMAND DEMONSTRATION

It illustrates the uniq command on file3.txt, which is used to filter out or report repeated lines.



DIRECTORY OPERATIONS AND LISTING

This shows attempts to remove directories using rmdir (which fail with "Directory not empty"). ls commands are used to list directory contents after these operations. ro back and ro snap commands are also shown, likely related to deleting files or directories.

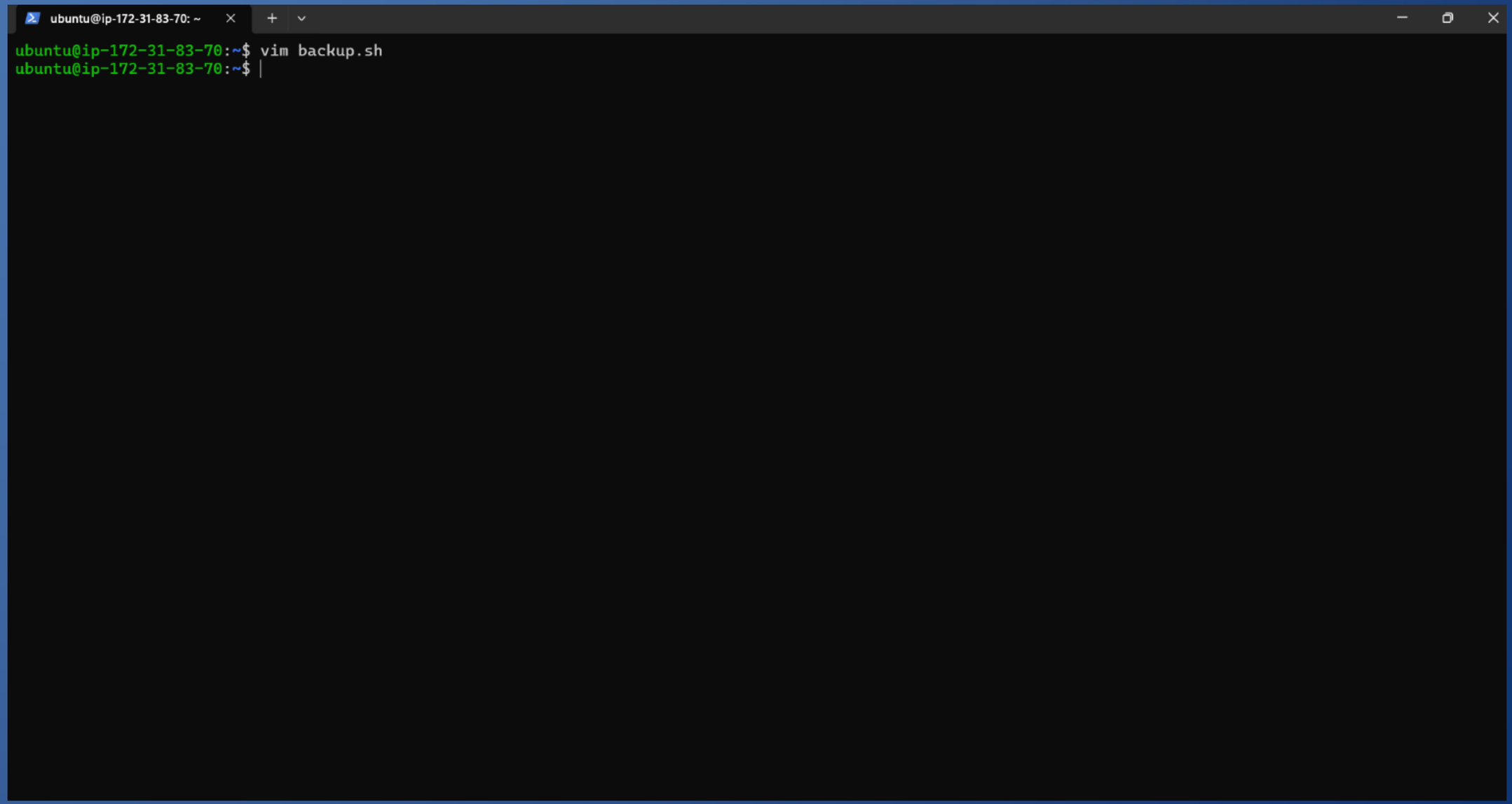


AUTOMATED BACKUP SCRIPT

This project focuses on the development and automation of a shell script, backup.sh, designed to create and manage system backups with a 5-day rotation policy.

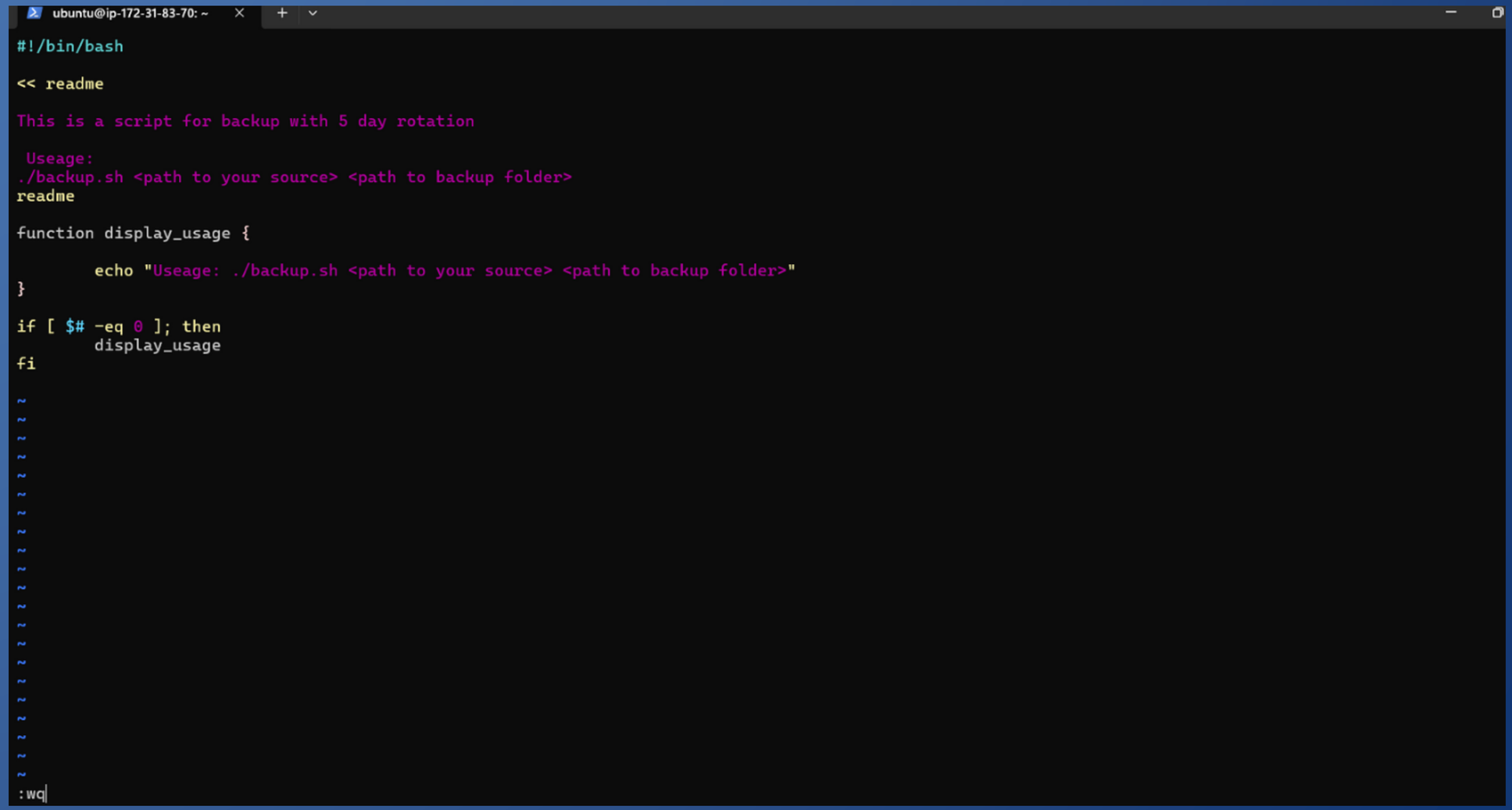
INITIAL SESSION DETAILS

I HAVE began a session on ip-172-31-83-70. The primary activity shown is initiating vim to edit backup.sh.



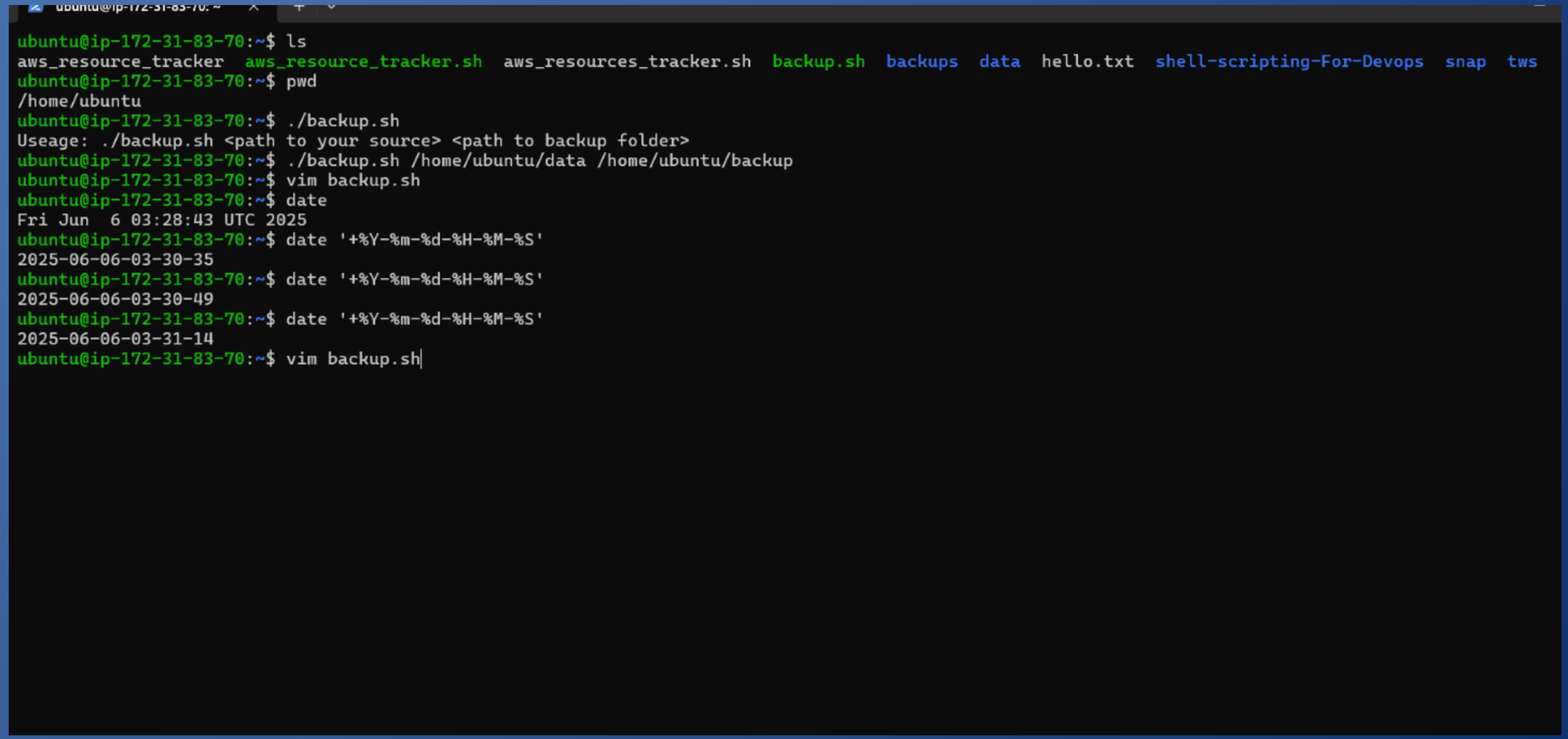
SCRIPT INTRODUCTION AND USAGE

This page introduces backup.sh as a bash script designed for a 5-day backup rotation. It specifies the script's usage: /backup.sh <path to your source> <path to backup folder>. A display_usage function is defined to print this message, and the script includes a check (if [$# -eq];) to call display_usage if no arguments are provided.



INITIAL EXECUTION & DATE FORMATTING

This page shows the user's current directory (/home/ubuntu) and lists its contents, including backup.sh. When backup.sh is executed without arguments, it correctly outputs its usage message. The user also demonstrates experimenting with the date command to format timestamps (e.g., +%Y-%m-%d-%H-%M-%S).



IMPLEMENTING BACKUP CREATION FUNCTION

This page shows the backup.sh script being updated to define variables for source_dir ($1) and backup_dir ($2). A timestamp variable is also set using date +%Y-%m-%d-%H-%M-%S. The core create_backup function is introduced, which is responsible for zipping the source_dir into a timestamped file named backup_${timestamp}.zip within the backup_dir. The function call create_backup is also shown.

```
#!/bin/bash

<< readme

This is a script for backup with 5 day rotation

 Useage:
./backup.sh <path to your source> <path to backup folder>
readme

function display_usage {

        echo "Useage: ./backup.sh <path to your source> <path to backup folder>"
}

if [ $# -eq 0 ]; then
        display_usage
fi

source_dir=$1
timestamp=$(date '+%Y-%m-%d-%H-%M-%S')
backup_dir=$2

function create_backup {

        zip -r "${backup_dir}/backup_${timestamp}.zip" "${source_dir}"

        echo "backup generated successfully for ${timestamp}"


create_backup


~
~
~
~
~
~
~
:wq
```

ADDING BACKUP SUCCESS CHECK

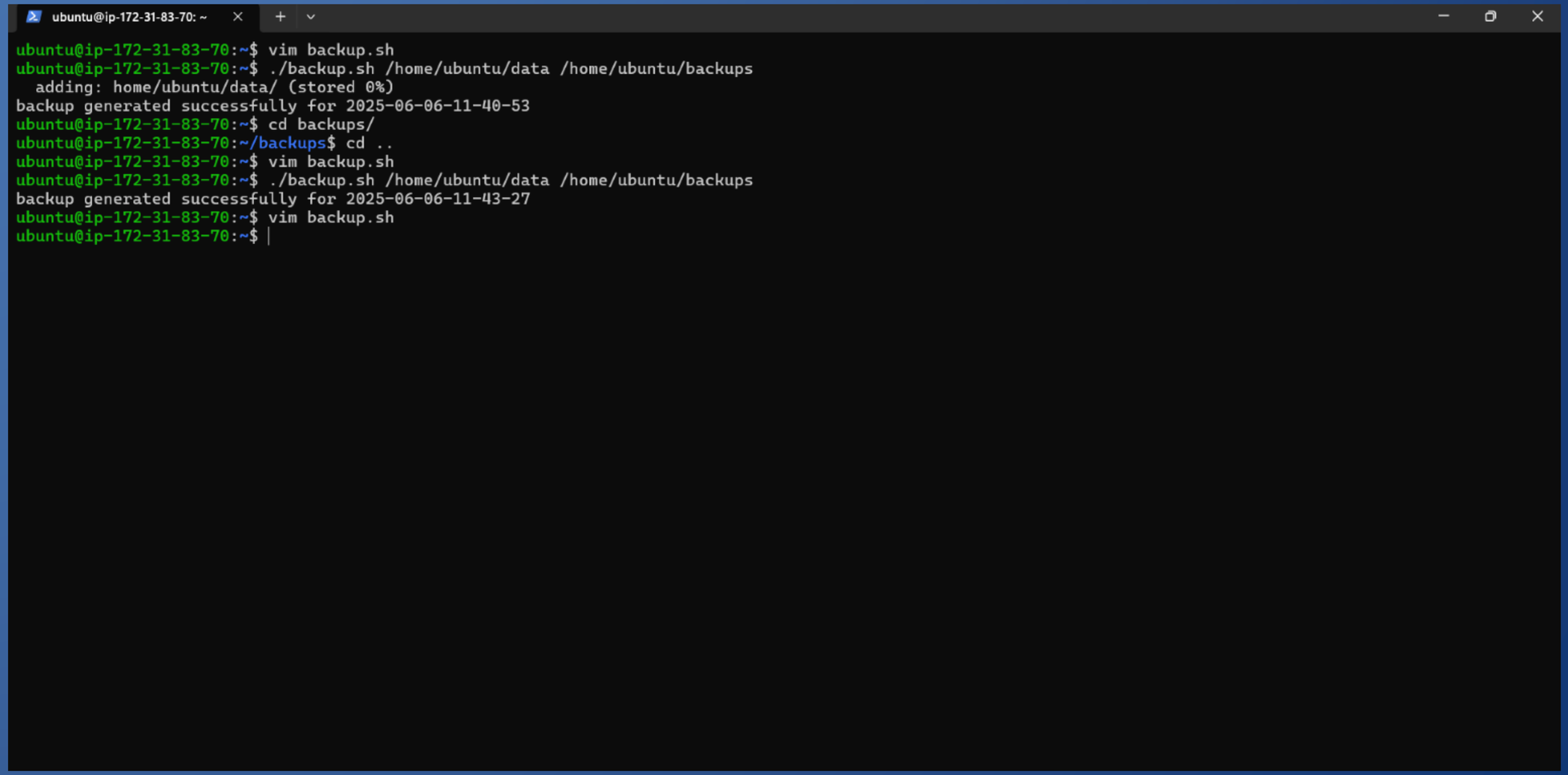
IN THIS the create_backup function being updated to include an error check. Specifically, an if condition (if [$? -eq];) is added immediately after the zip command. This condition checks the exit status of the zip command to determine if it executed successfully. If successful, the script echoes a message: "backup generated successfully for ${timestamp}".

ubuntu@ip-172-31-83-70: ~

```
#!/bin/bash

<< readme

This is a script for backup with 5 day rotation

 Useage:
./backup.sh <path to your source> <path to backup folder>
readme

function display_usage {

        echo "Useage: ./backup.sh <path to your source> <path to backup folder>"
}

if [ $# -eq 0 ]; then
        display_usage
fi

source_dir=$1
timestamp=$(date '+%Y-%m-%d-%H-%M-%S')
backup_dir=$2

function create_backup {

        zip -r "${backup_dir}/backup_${timestamp}.zip" "${source_dir}"

        if [ $? -eq 0 ]; then
                echo "backup generated successfully for ${timestamp}"
        fi
}

create_backup


~
~
~
~
~
-- INSERT --                                                    32,1          All
```

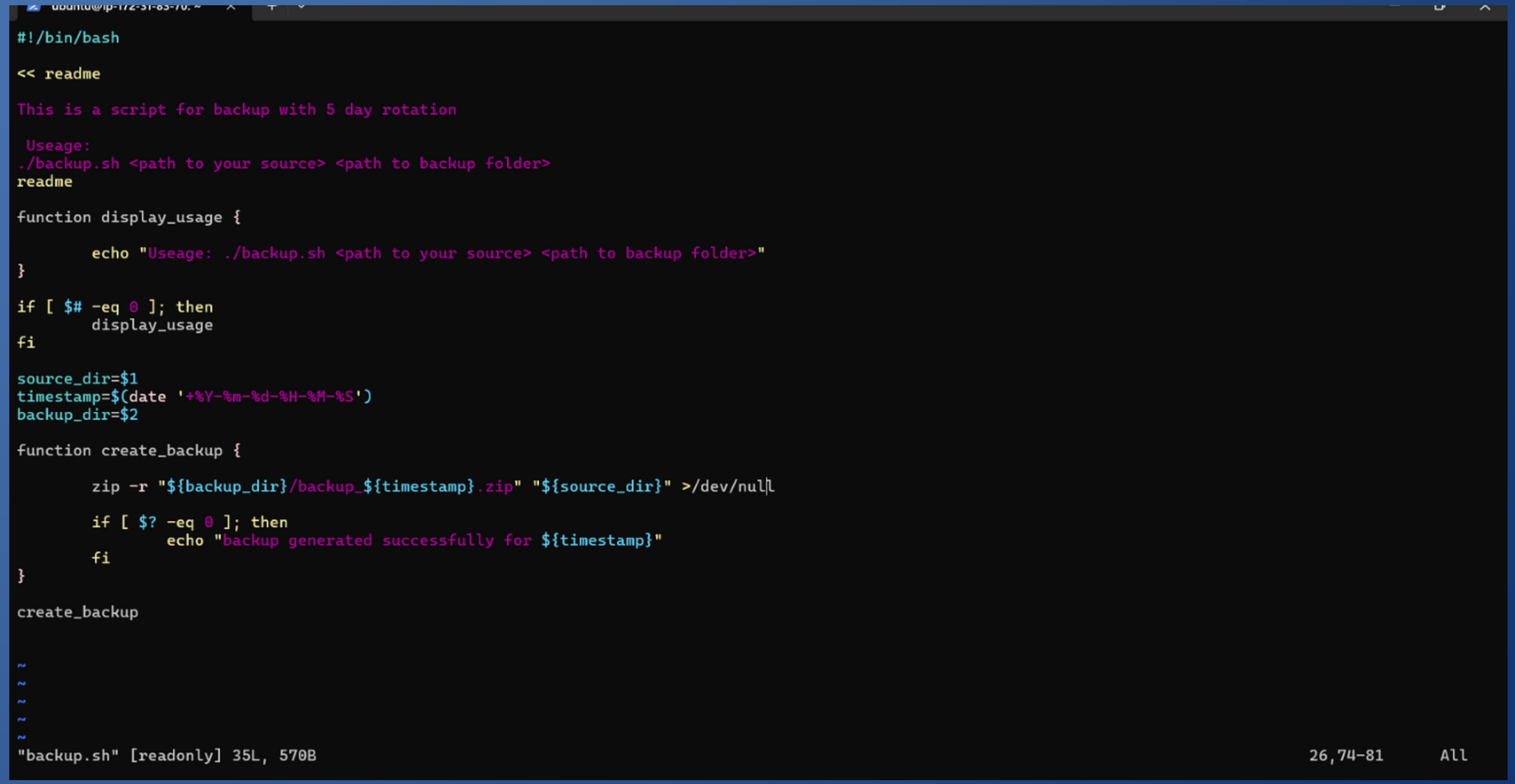
TESTING BACKUP CREATION

The execution of the backup.sh script with specific source and backup directories: ./backup.sh /home/ubuntu/data /home/ubuntu/backups. The output shows messages indicating successful backup generation, such as "adding: home/ubuntu/data/ (stored 0%)" and "backup generated successfully for [timestamp]". This confirms that the create_backup function is working as intended, and a backup file is being produced.



SUPPRESSING VERBOSE OUTPUT

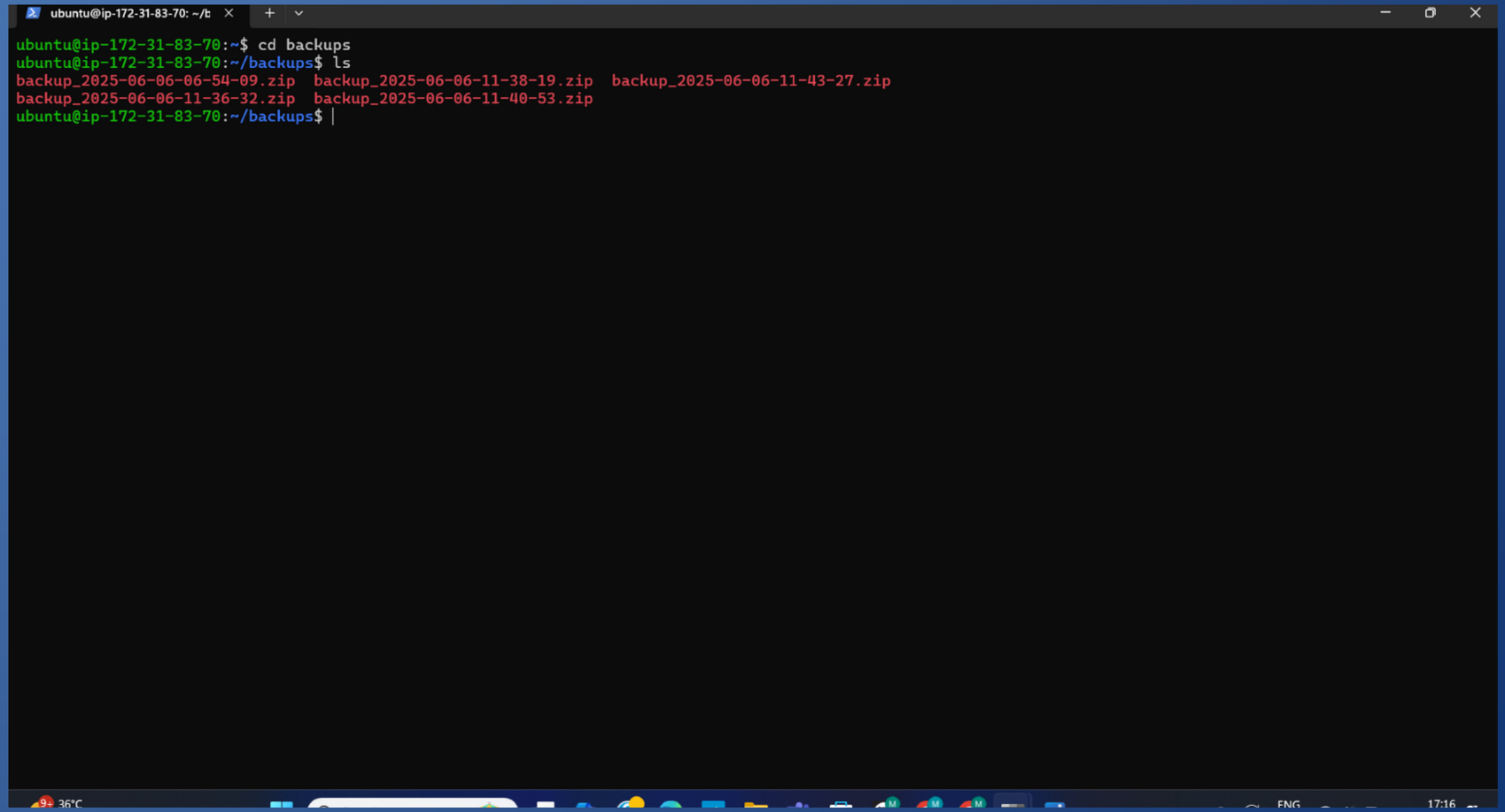
This shows a modification to the create_backup function within backup.sh. The zip command is updated to redirect its output to /dev/null (>/dev/null). This change prevents the verbose output of the zip command from being displayed on the console during script execution, leading to cleaner and less cluttered output.



```
#!/bin/bash

<< readme

This is a script for backup with 5 day rotation

 Useage:
./backup.sh <path to your source> <path to backup folder>
readme

function display_usage {

        echo "Useage: ./backup.sh <path to your source> <path to backup folder>"
}

if [ $# -eq 0 ]; then
        display_usage
fi

source_dir=$1
timestamp=$(date '+%Y-%m-%d-%H-%M-%S')
backup_dir=$2

function create_backup {

        zip -r "${backup_dir}/backup_${timestamp}.zip" "${source_dir}" >/dev/null

        if [ $? -eq 0 ]; then
                echo "backup generated successfully for ${timestamp}"
        fi
}

create_backup


~
~
~
~
~
"backup.sh" [readonly] 35L, 570B                                    26,74-81      All
```

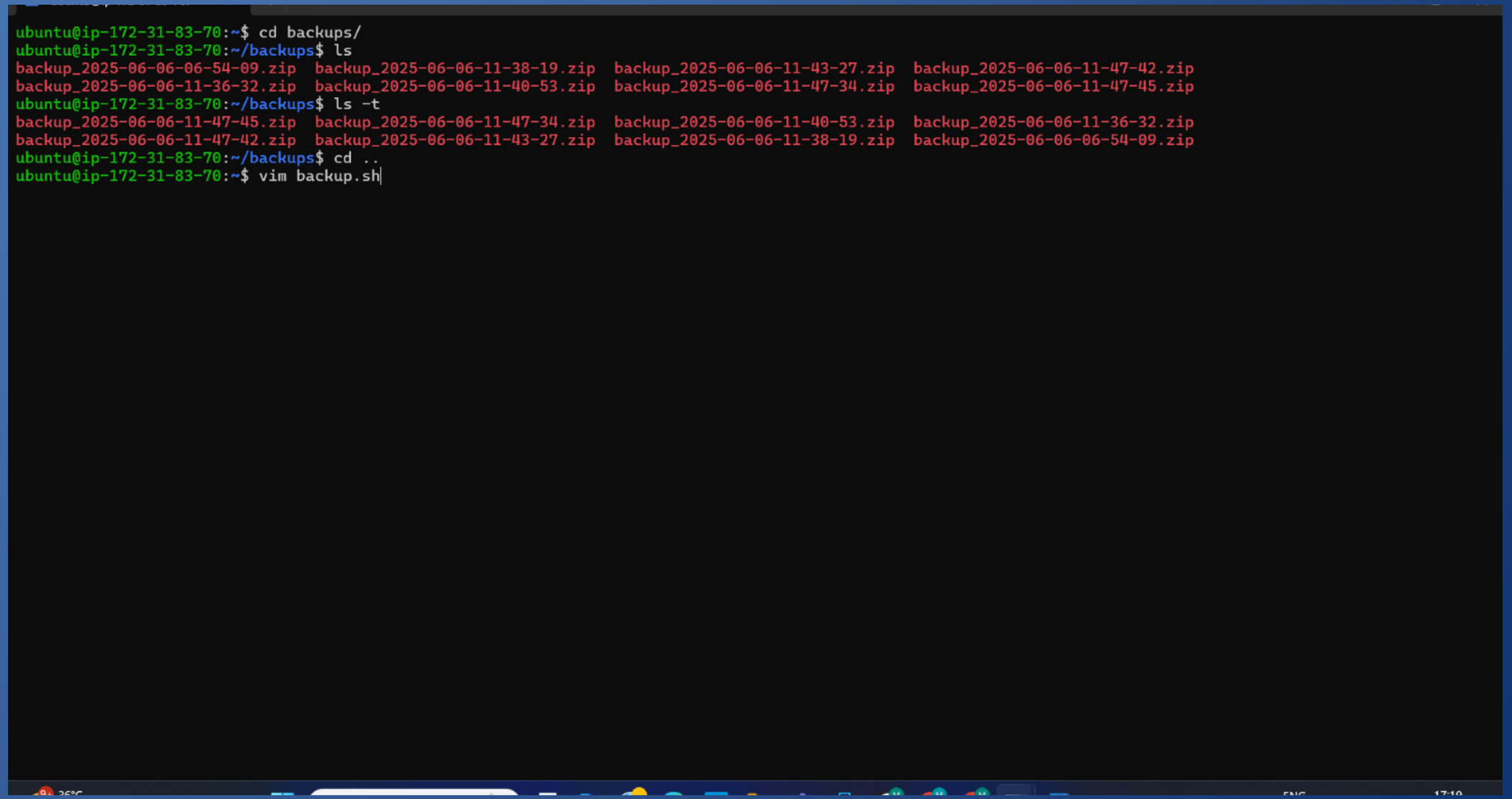
VERIFYING BACKUP FILES

This page shows the user navigating into the backups directory using the cd backups command. Following this, the ls command is used to list the contents of the directory, which displays a series of backup_YYYY-MM-DD-HH-MM-SS.zip files. This step visually confirms that timestamped backup archives are being successfully created and stored in the designated location.



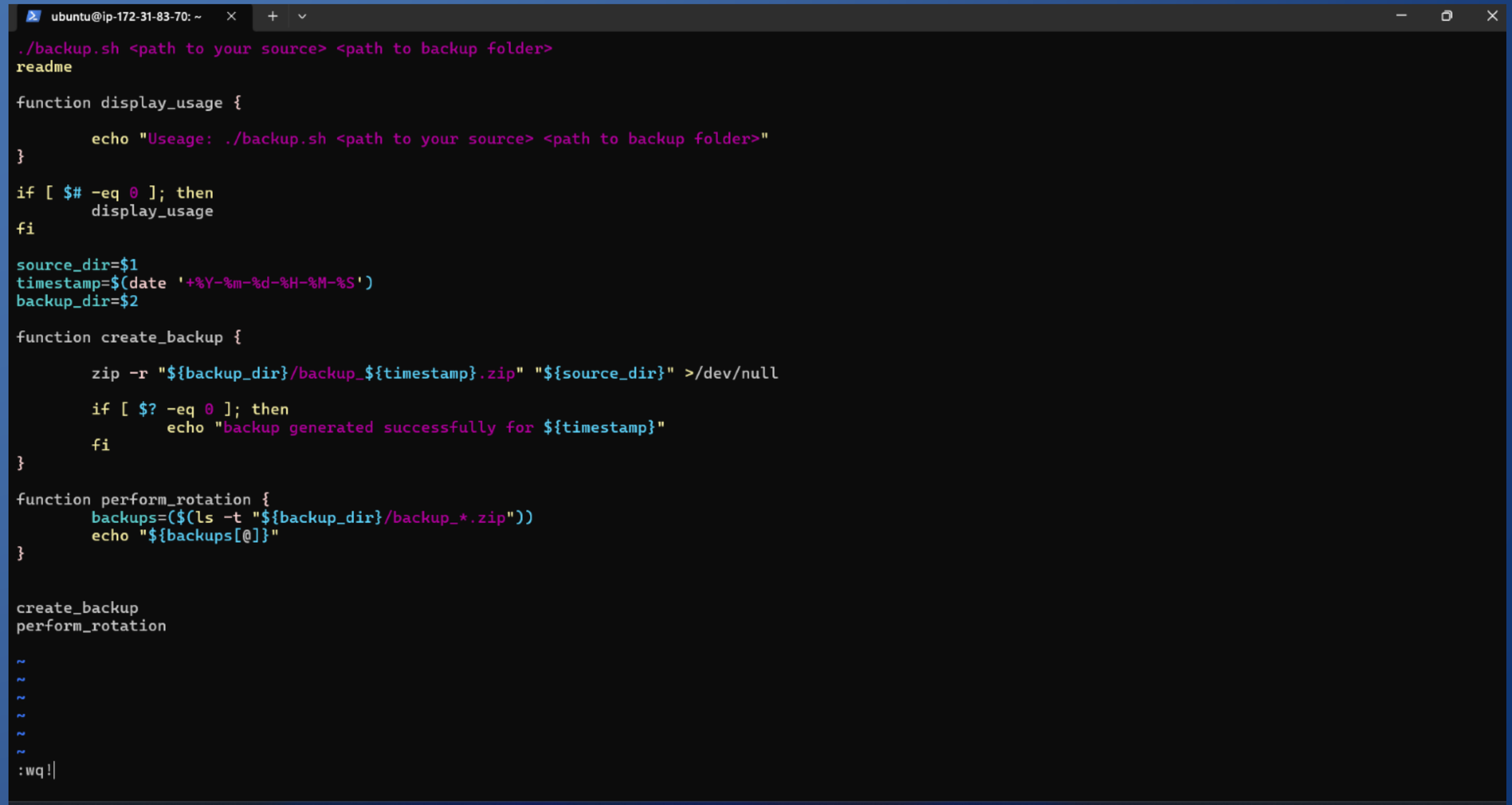
VERIFYING BACKUP FILES

This page continues the verification of backup files. It shows the use of the ls -t command within the backups directory. This command lists the backup files sorted by their modification time, with the newest files appearing first. This helps confirm the chronological order and recency of the created archives.



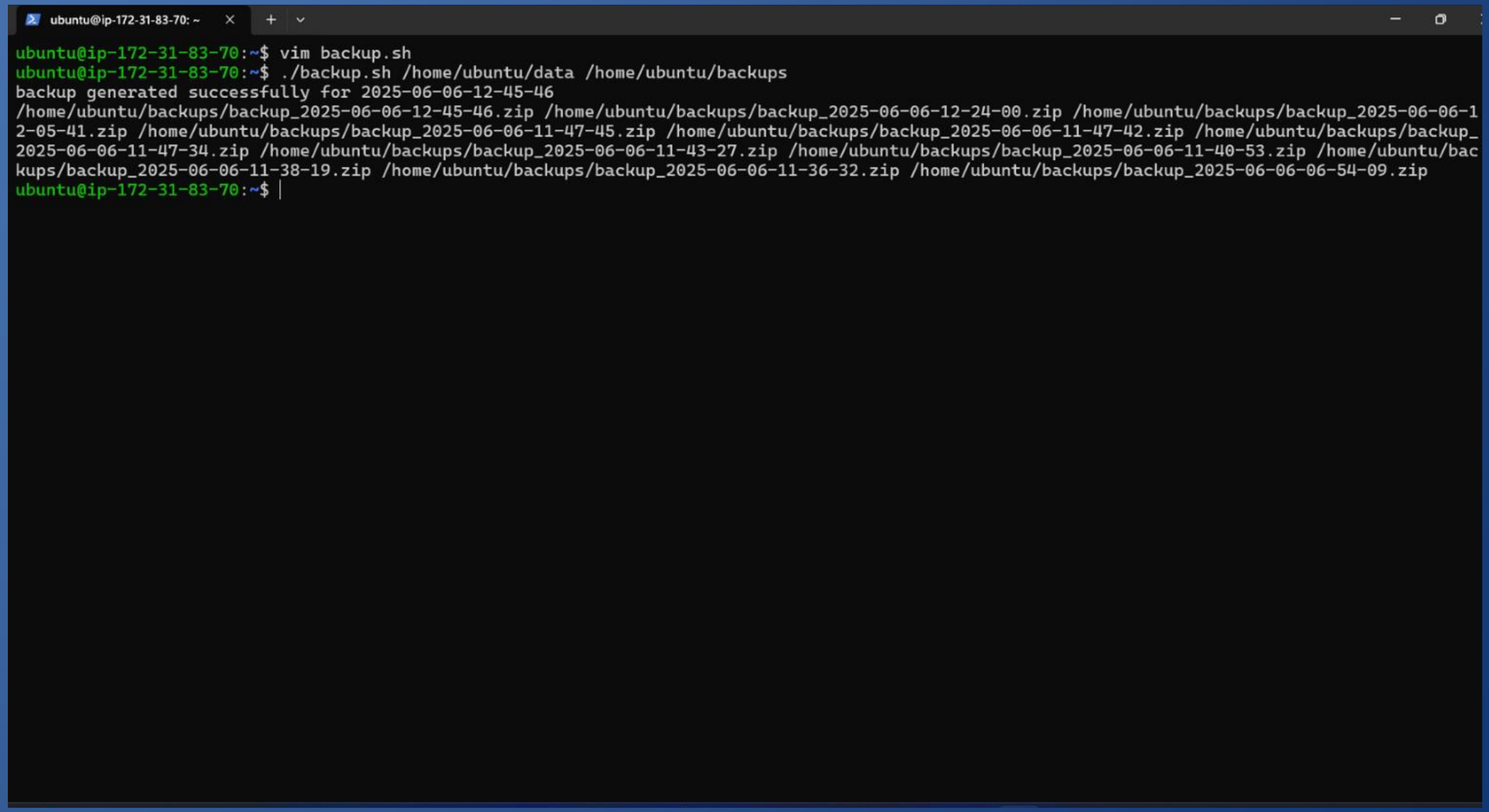
BEGINNING BACKUP ROTATION FUNCTION

This marks the introduction of the perform_rotation function within the backup.sh script. This function is designed to manage backup retention. It begins by listing backup files sorted by modification time (ls -t "\${backup_dir}/backup.zip") and stores them in an array named backups. The content of this backups array is then echoed, likely for debugging or initial visual verification.



DISPLAYING ALL BACKUPS (VISUAL TEST)

This illustrates a visual test of the perform_rotation function. After the script is executed, the output shows a long list of all existing backup files, similar to the output from ls -t. This demonstrates the functionality of the echo "${backups[@]}" command within the perform_rotation function, confirming that the array is correctly populated and its contents can be displayed.



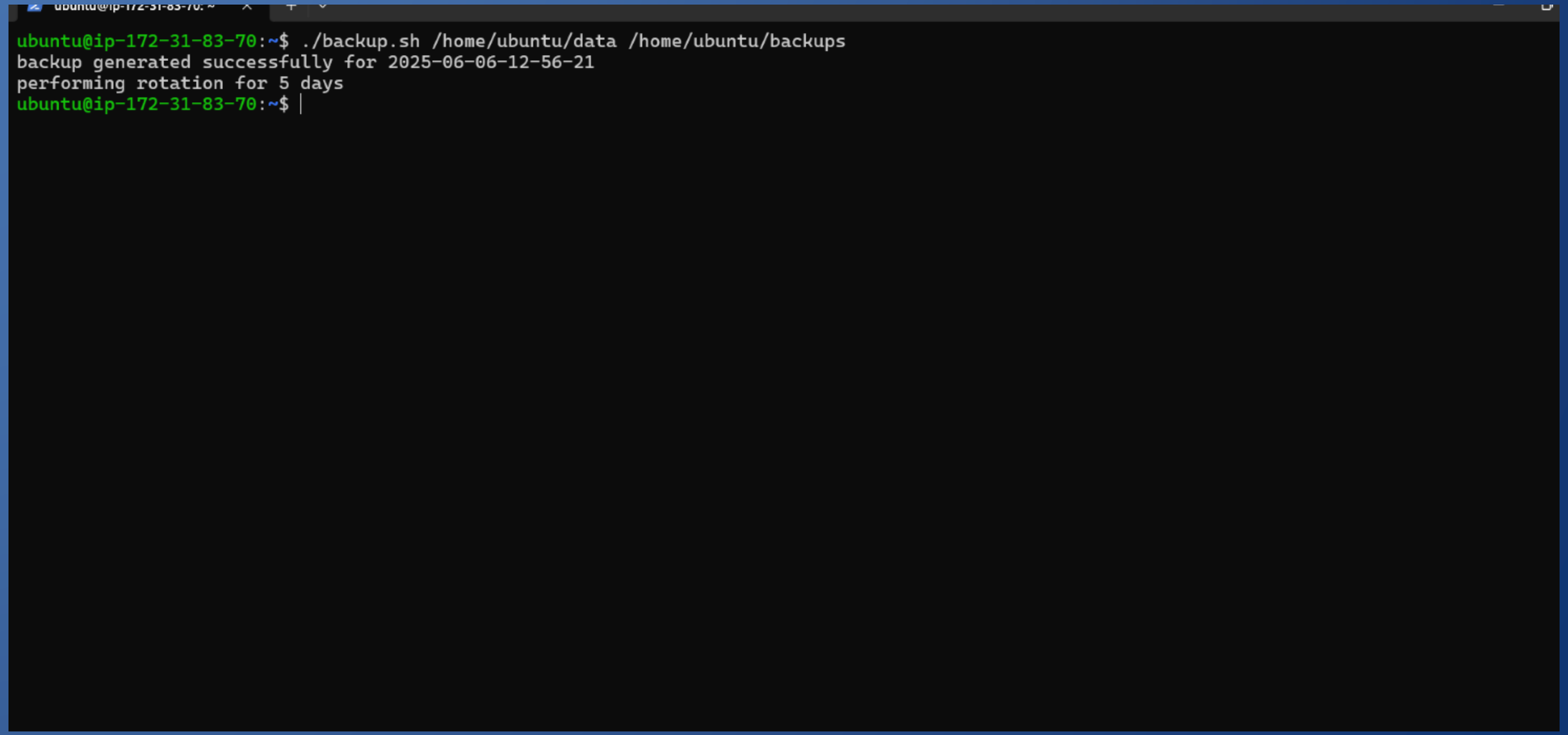
REFINING ROTATION CONDITION

This shows a further refinement to the perform_rotation function within backup.sh. Specifically, an if condition if ["${#backups[@]}" -gt 5] is introduced. This condition checks if the total number of backup files listed in the backups array is greater than 5. This is a crucial step towards implementing the 5-day rotation policy, as it determines when older backups need to be removed.

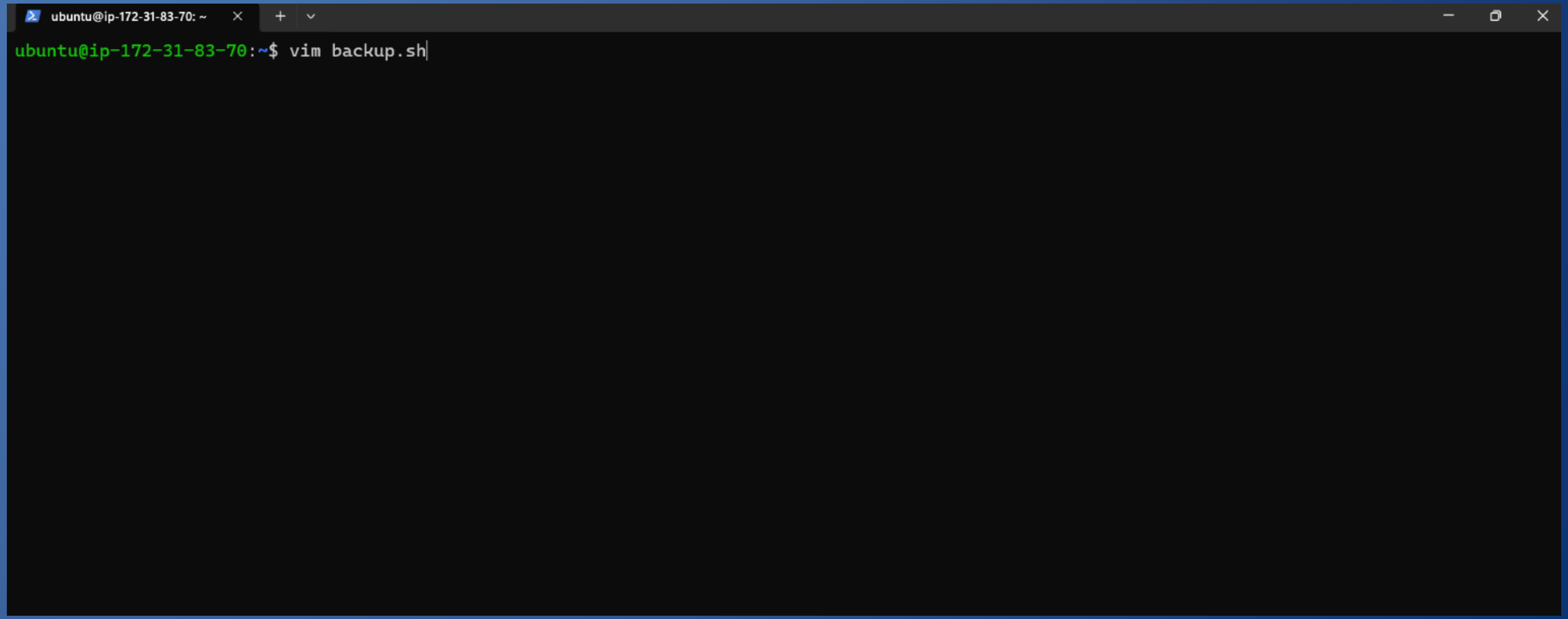
```
function display_usage {

        echo "Useage: ./backup.sh <path to your source> <path to backup folder>"
}

if [ $# -eq 0 ]; then
        display_usage
fi

source_dir=$1
timestamp=$(date '+%Y-%m-%d-%H-%M-%S')
backup_dir=$2

function create_backup {

        zip -r "${backup_dir}/backup_${timestamp}.zip" "${source_dir}" >/dev/null

        if [ $? -eq 0 ]; then
                echo "backup generated successfully for ${timestamp}"
        fi
}

function perform_rotation {
        backups=($(ls -t "${backup_dir}/backup_"*.zip 2>/dev/null))

        if [ "${#backups[@]}" -gt 5 ]; then
                echo "performing rotation for 5 days"
        fi


}


create_backup
perform_rotation

~
:wq!
```

ROTATION MESSAGE

This shows the backup.sh script now printing the message "performing rotation for 5 days" during execution. This output appears when the condition for backup rotation is met, providing a clear indication that the script is actively managing older backups according to the defined 5-day policy.

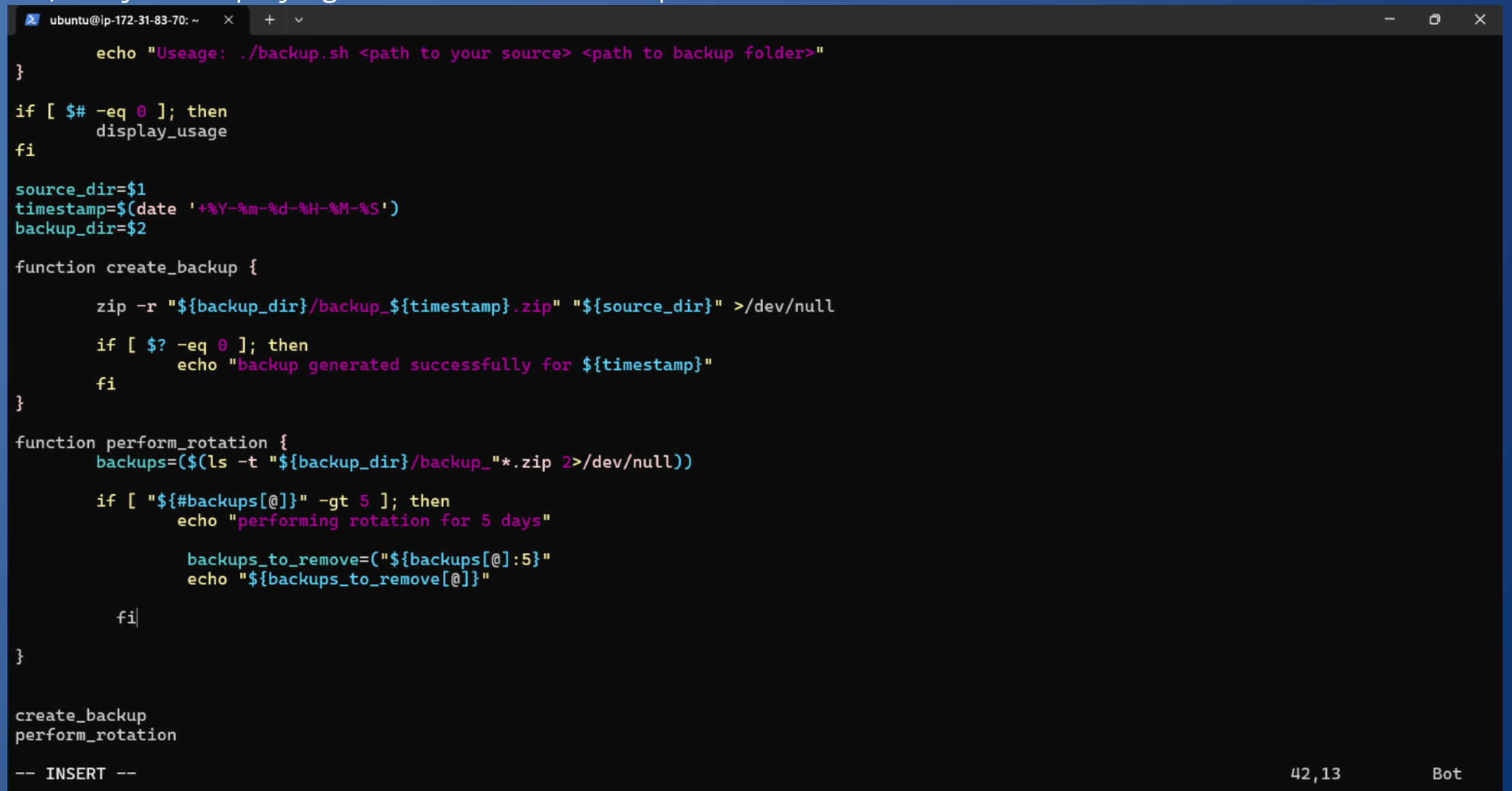


ENTERING THE VIM BACKUP.SH COMMAND



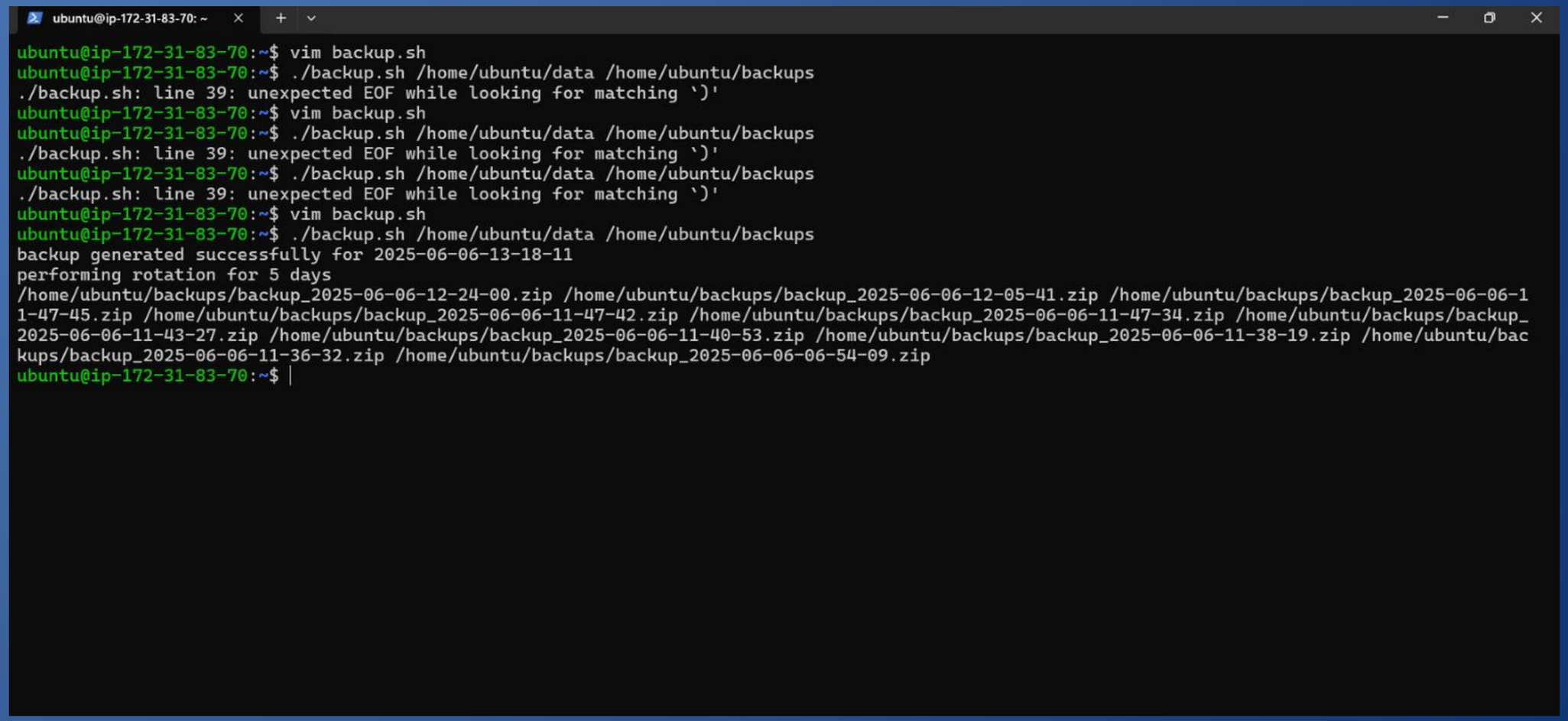
IDENTIFYING BACKUPS FOR REMOVAL

This page details a crucial step in the perform_rotation function: identifying which backups should be removed. The code backups_to_remove=(${backups[@]:5}) is shown, which selects all elements in the backups array starting from the sixth element (index 5), effectively targeting backups older than the five most recent. The line echo "${backups_to_remove[@]}" is also present, likely for displaying these identified backups before deletion.

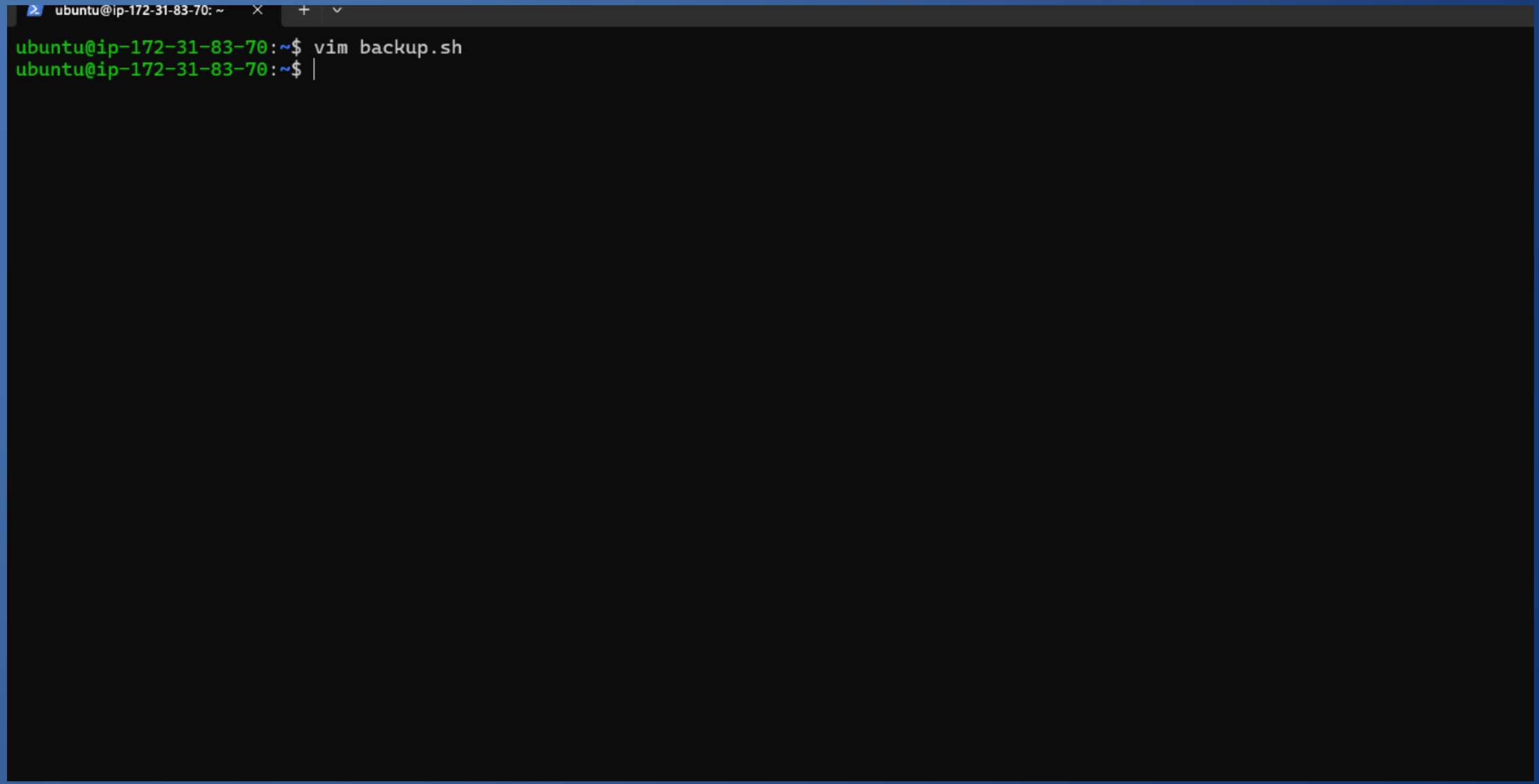


SCRIPT SYNTAX ERROR (INTERIM STATE)

This page shows an error message encountered during script execution: "line 39: unexpected EOF while looking for matching ')'". This indicates a syntax issue or an incomplete code block within the backup.sh script, specifically on line 39. This suggests the script was in an interim state during development, requiring debugging.

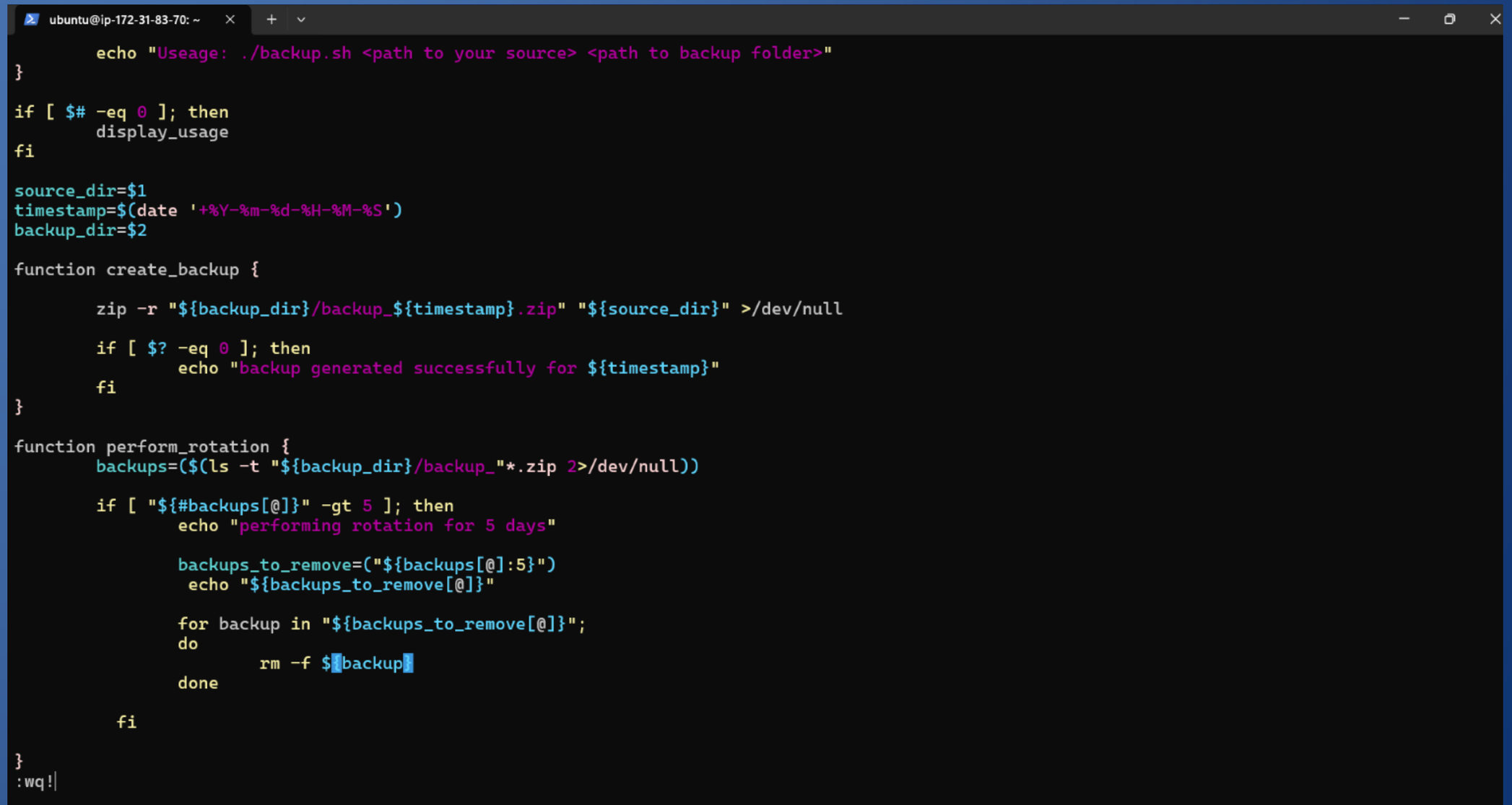


ENTERING THE COMMAND VIM BACKUP.SH



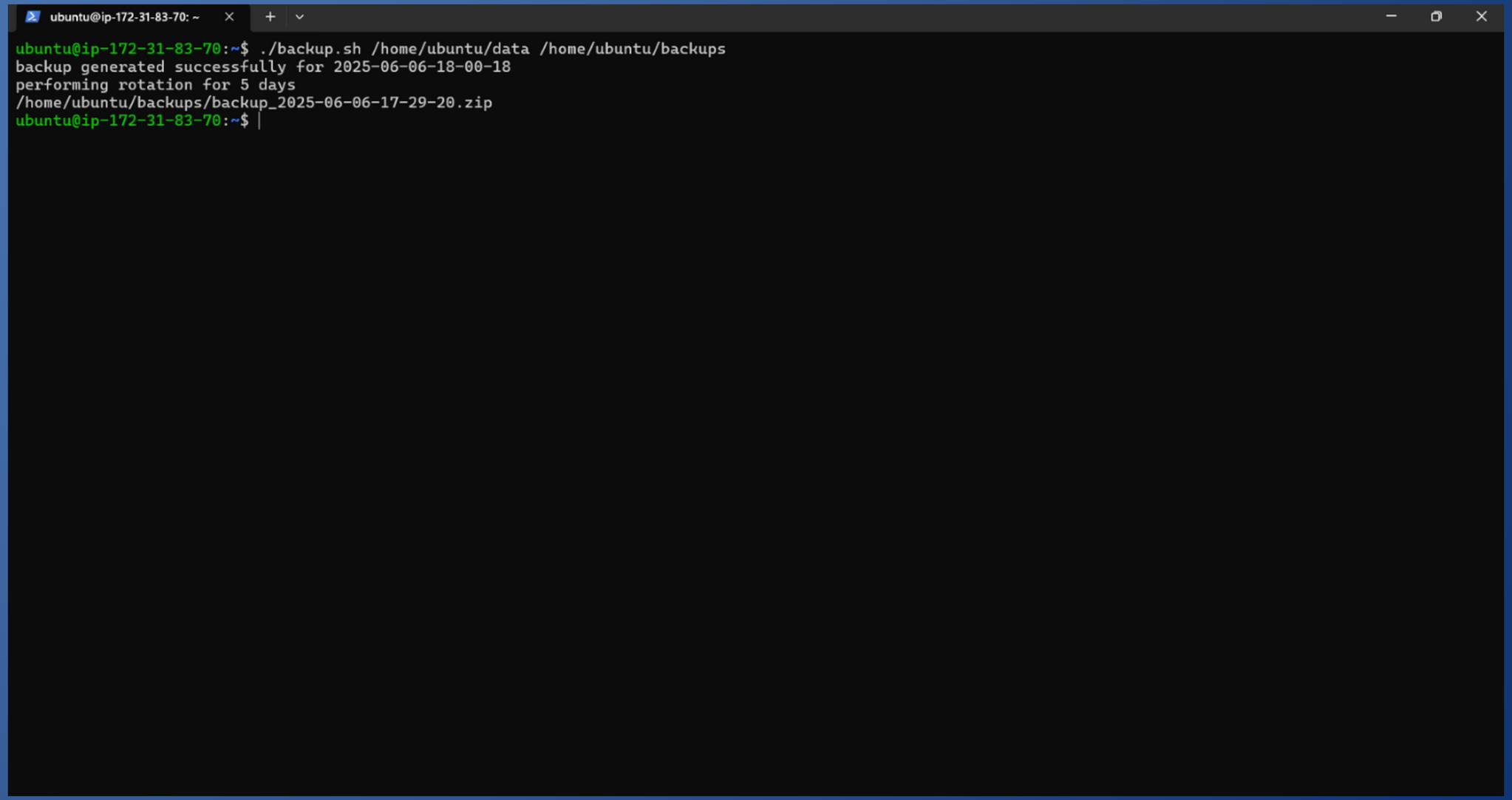
IMPLEMENTING BACKUP REMOVAL

This page details the crucial step of implementing the actual deletion of old backup files within the perform_rotation function. A for loop is introduced: for backup in "${backups_to_remove[@]}"; do rm -f "$backup"; done. This loop iterates through the backups_to_remove array (which contains backups older than the 5 most recent) and uses rm -f to forcibly remove each specified backup file.



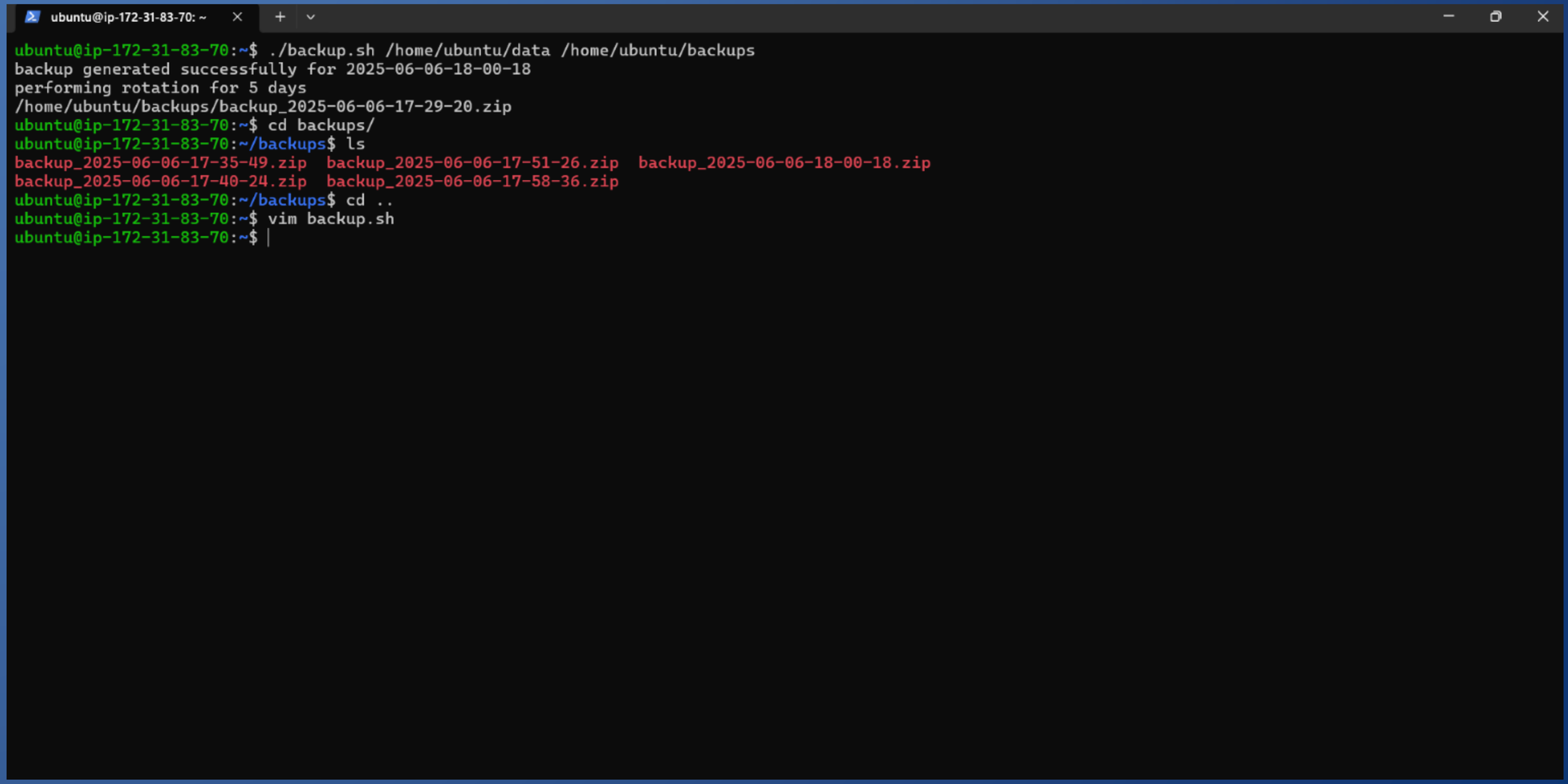
TESTING BACKUP REMOVAL

This shows the initial results of testing the implemented backup removal logic. After running the backup.sh script, the output confirms "backup generated successfully..." and also indicates "performing rotation for 5 days." This signifies that both the backup creation and the rotation processes are being triggered. This page visually implies the start of the verification that older backups are indeed being managed.



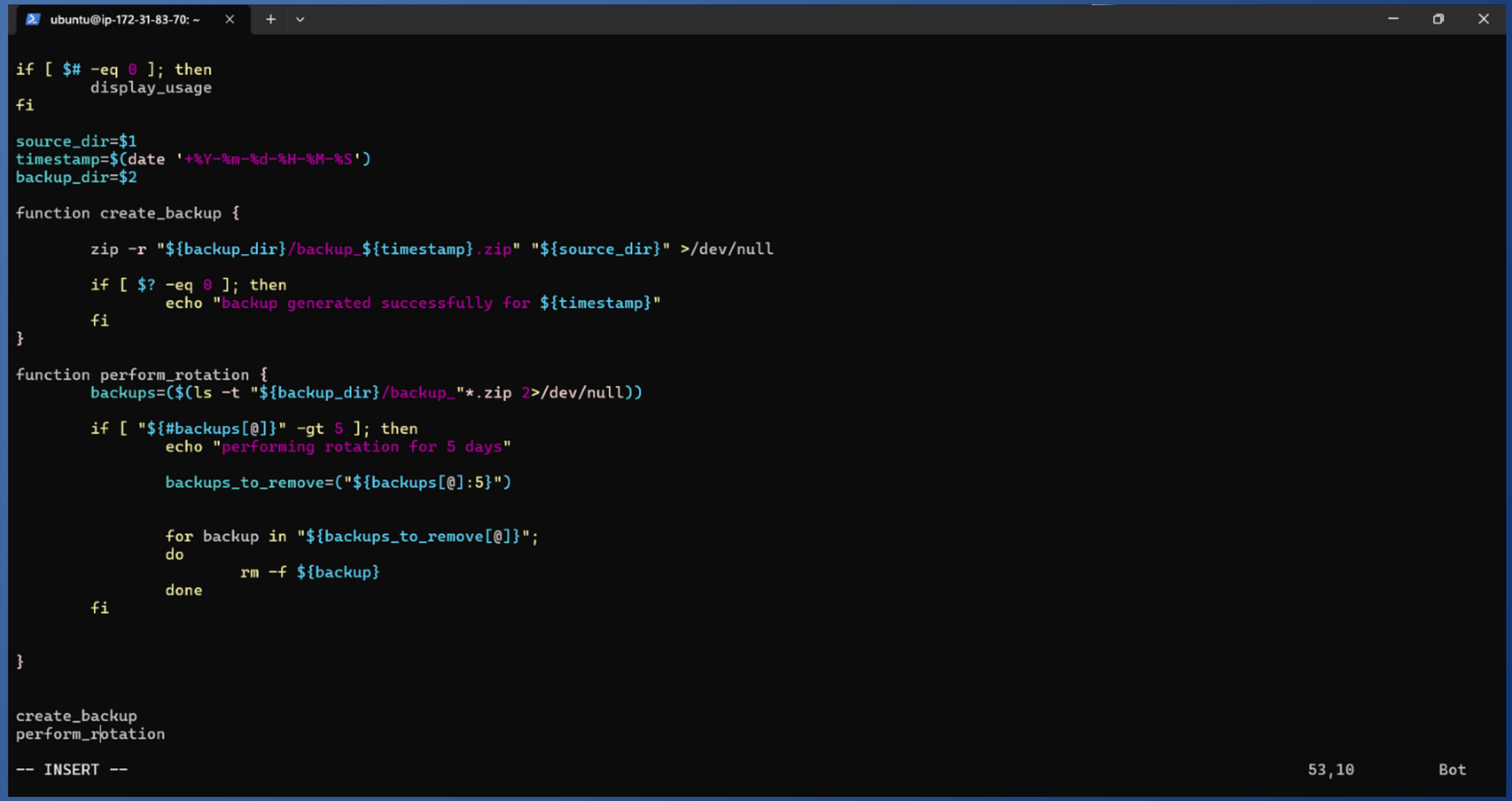
CONFIRMING BACKUP REMOVAL

This page continues the verification of the backup rotation. After the script runs, listing the backups (e.g., with ls) shows fewer files than before, specifically, only the five most recent backups remain. This visual confirmation indicates that the perform_rotation function, including the rm -f command for deletion, has successfully removed the older backup files as intended by the 5-day rotation policy.



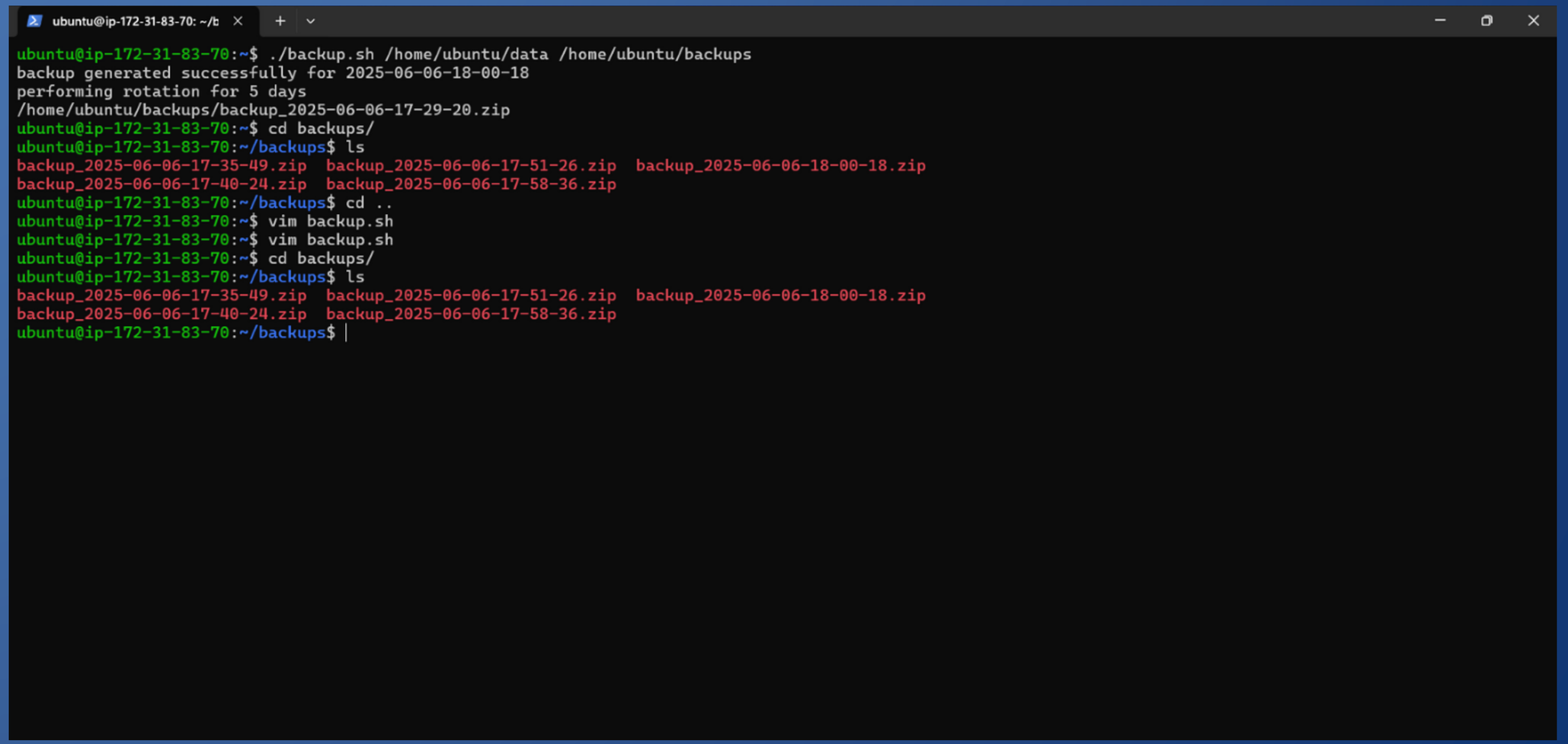
CONTINUED VERIFICATION

Given the sequential nature of the document, it contains further visual verification of the backup.sh script's execution. This would include confirming the output and behavior of the script after the backup rotation logic was implemented and tested, ensuring everything worked as expected before proceeding to automation.



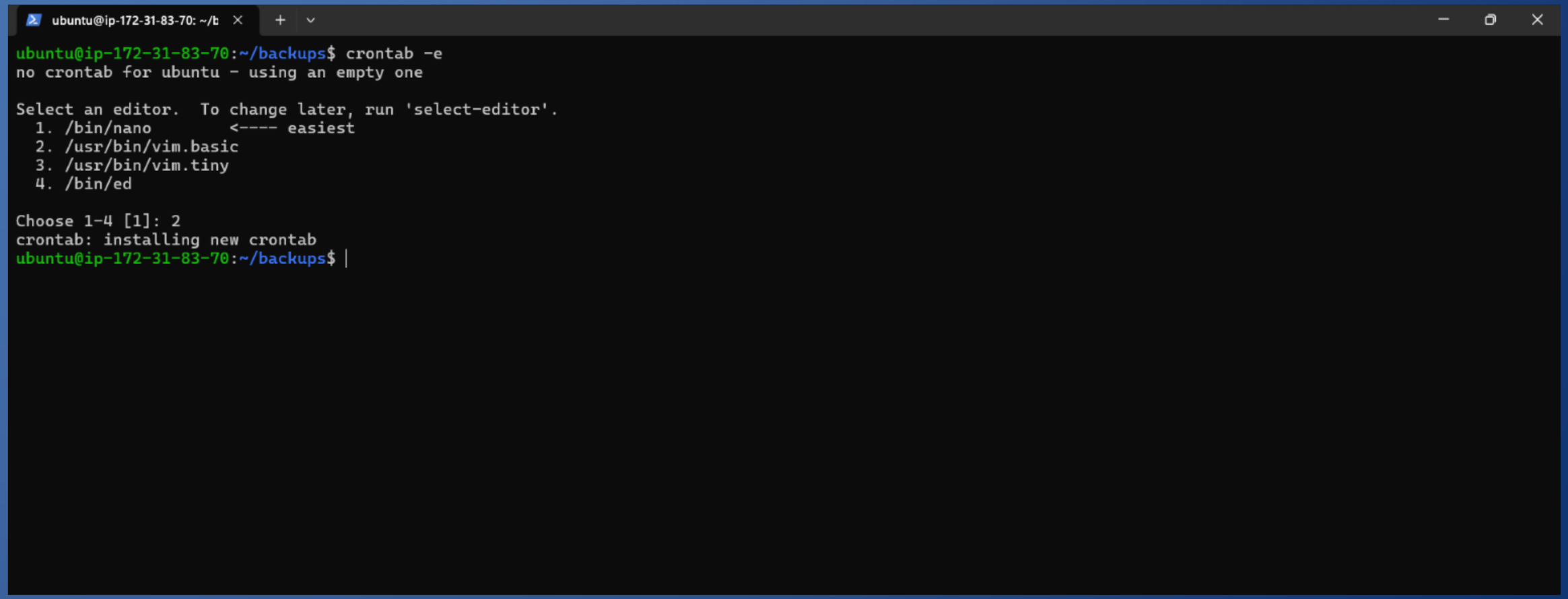
TRANSITION TO AUTOMATION

IT serves as a transitional point in the documentation. Given the preceding sections on script development and testing, This contains further visual verification of the backup.sh script's output or its behavior after the rotation logic has been fully implemented. It bridges the gap between script functionality confirmation and the final automation setup.



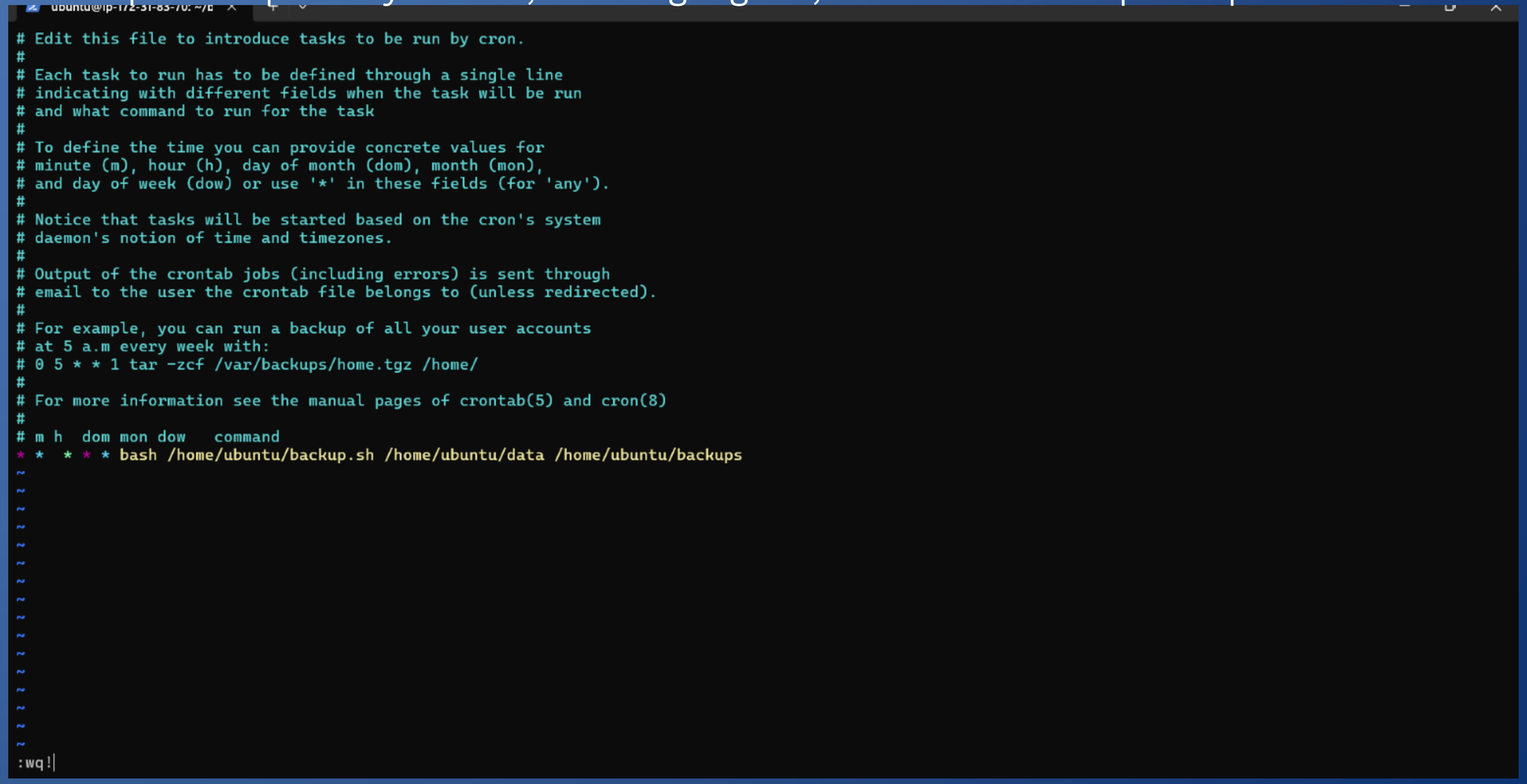
INITIATING CRON JOB SETUP

This page shows the command used to begin the process of automating the backup script. The user executes crontab -e, which opens the cron table (a time-based job scheduler) for editing. This is the first step towards setting up backup.sh to run automatically at defined intervals without manual intervention.



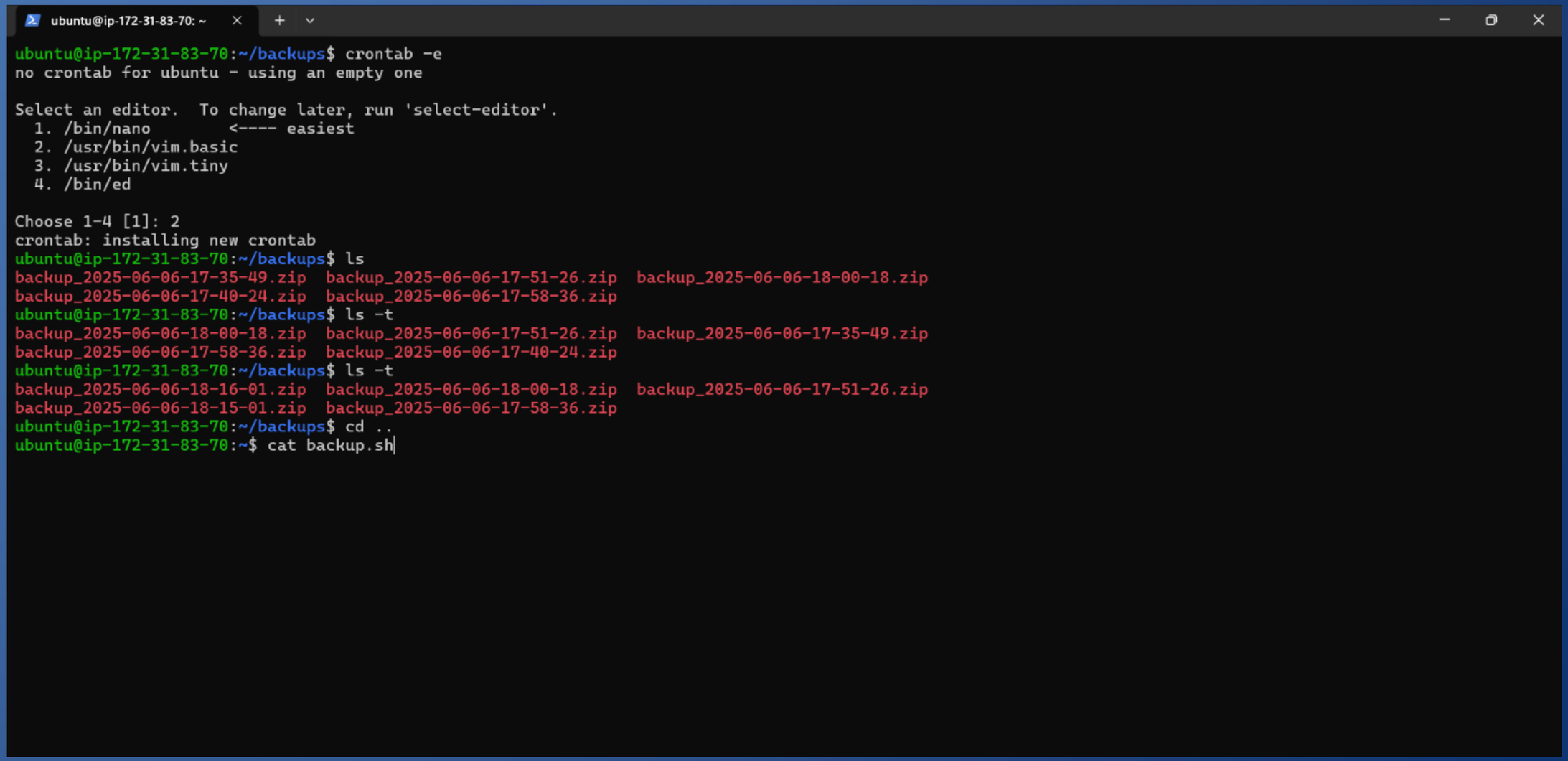
ADDING CRON JOB FOR AUTOMATION

This page details the crucial step of automating the backup.sh script using cron. The crontab -e command is shown, which opens the user's cron table for editing. Within this editor, a specific cron entry is added: * * * * * bash /home/ubuntu/backup.sh /home/ubuntu/data /home/ubuntu/backups. This line instructs the system to execute the backup.sh script every minute, ensuring regular, automated backups are performed.



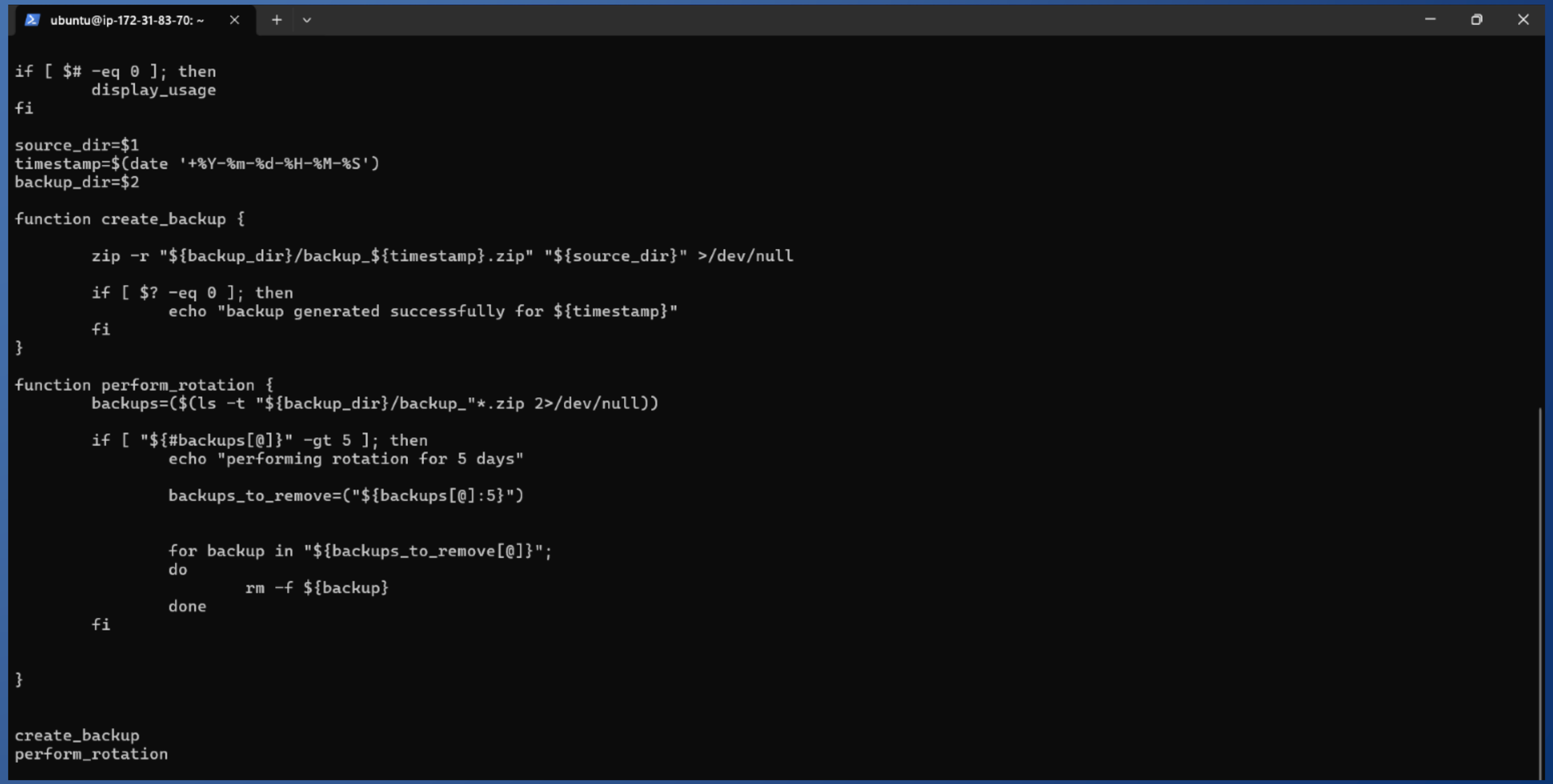
CRONTAB SETUP CONFIRMATION & BACKUP LISTING

This page confirms the successful setup of the automated backup schedule. The system displays "crontab: installing new crontab" after the user has configured the cron job. Following this, the user verifies the presence of existing backups by using the ls command within the backups directory, which lists various timestamped .zip archives. The command cat backup.sh is also shown, indicating the user might be reviewing the final content of their backup script.



AUTOMATED BACKUP SCRIPT OVERVIEW

This document presents the full backup.sh script. It includes functions for displaying proper usage, creating timestamped zip backups with error handling and suppressed output, and performing rotation. The perform_rotation function is key; it lists backups, identifies those older than the five most recent, and deletes them. The script concludes by running both the backup creation and rotation processes.

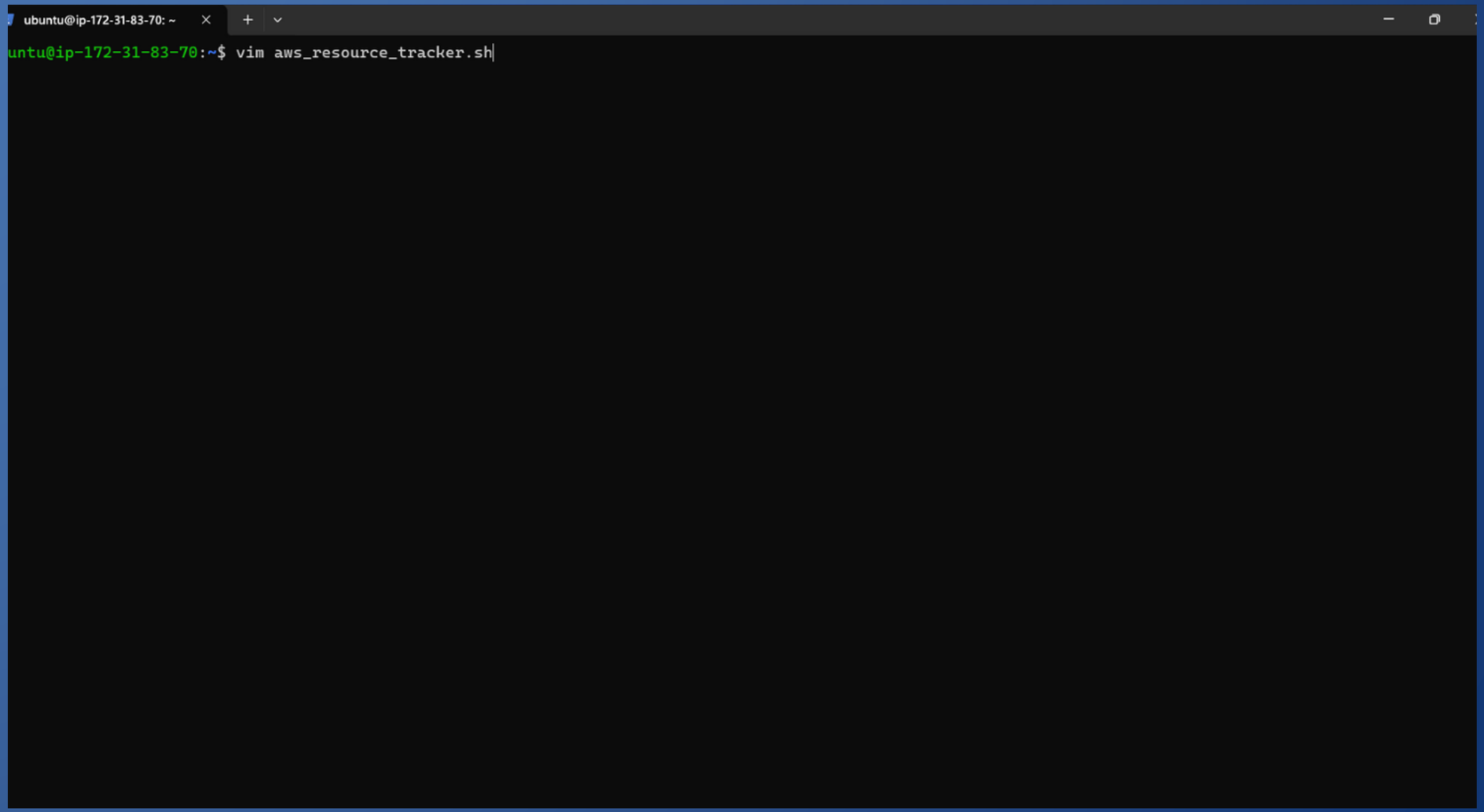


AWS RESOURCE TRACKER

This project centers on an "AWS Resource Tracker". I created and refined a bash script, aws_resource_tracker.sh, to report on AWS S3, EC2, Lambda, and IAM user usage. Key steps included configuring AWS access, making the script executable, running it to view resource output, and using jq to filter specific EC2 instance IDs. The script was further modified to include debugging with set -x and to redirect or append output to a file named resourceTracker.

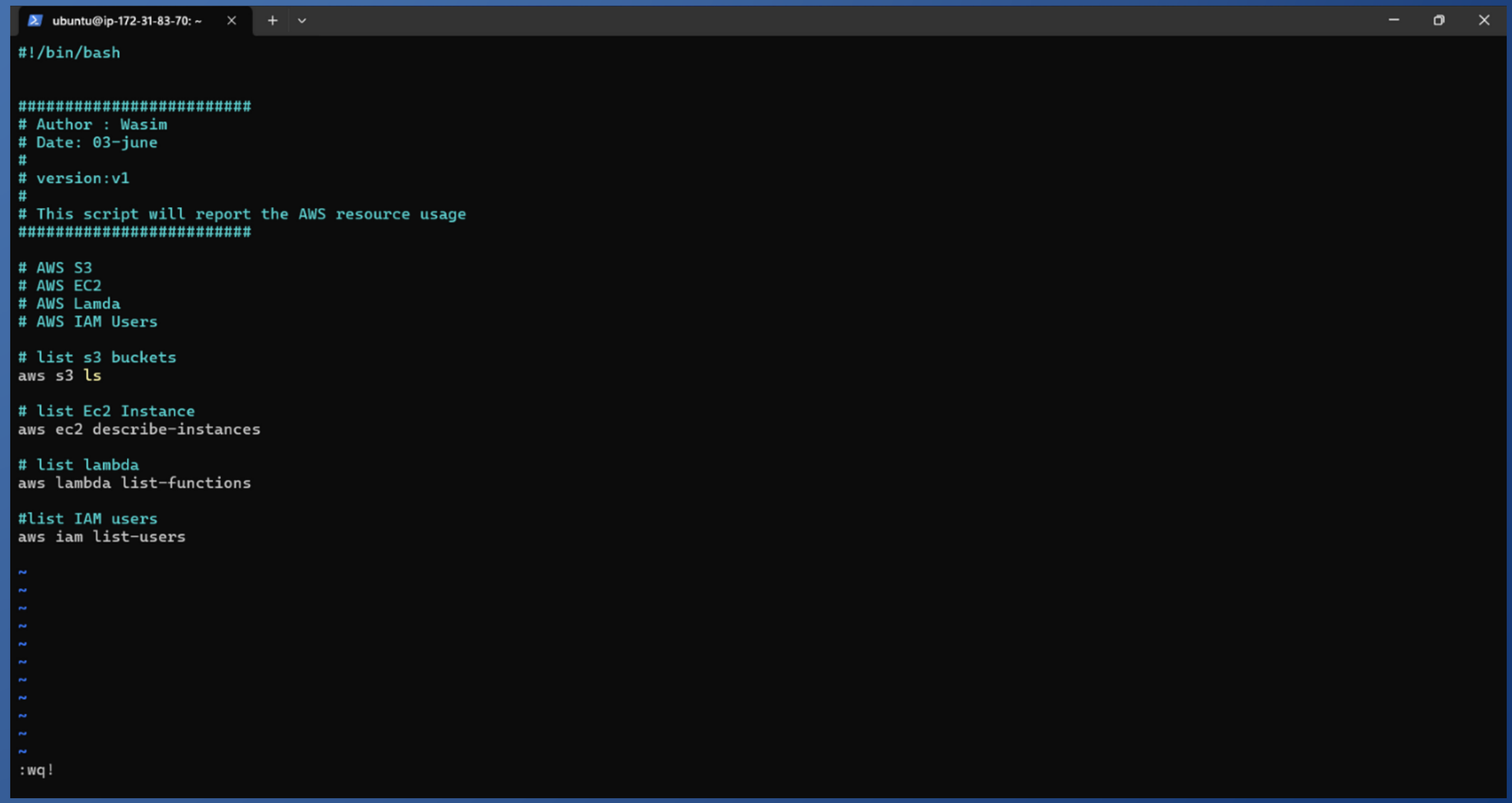
INITIAL SETUP & SCRIPT CREATION

This page shows a that i logged into an Ubuntu terminal, initiating the creation of a shell script named aws_resource_tracker.sh using the vim editor.



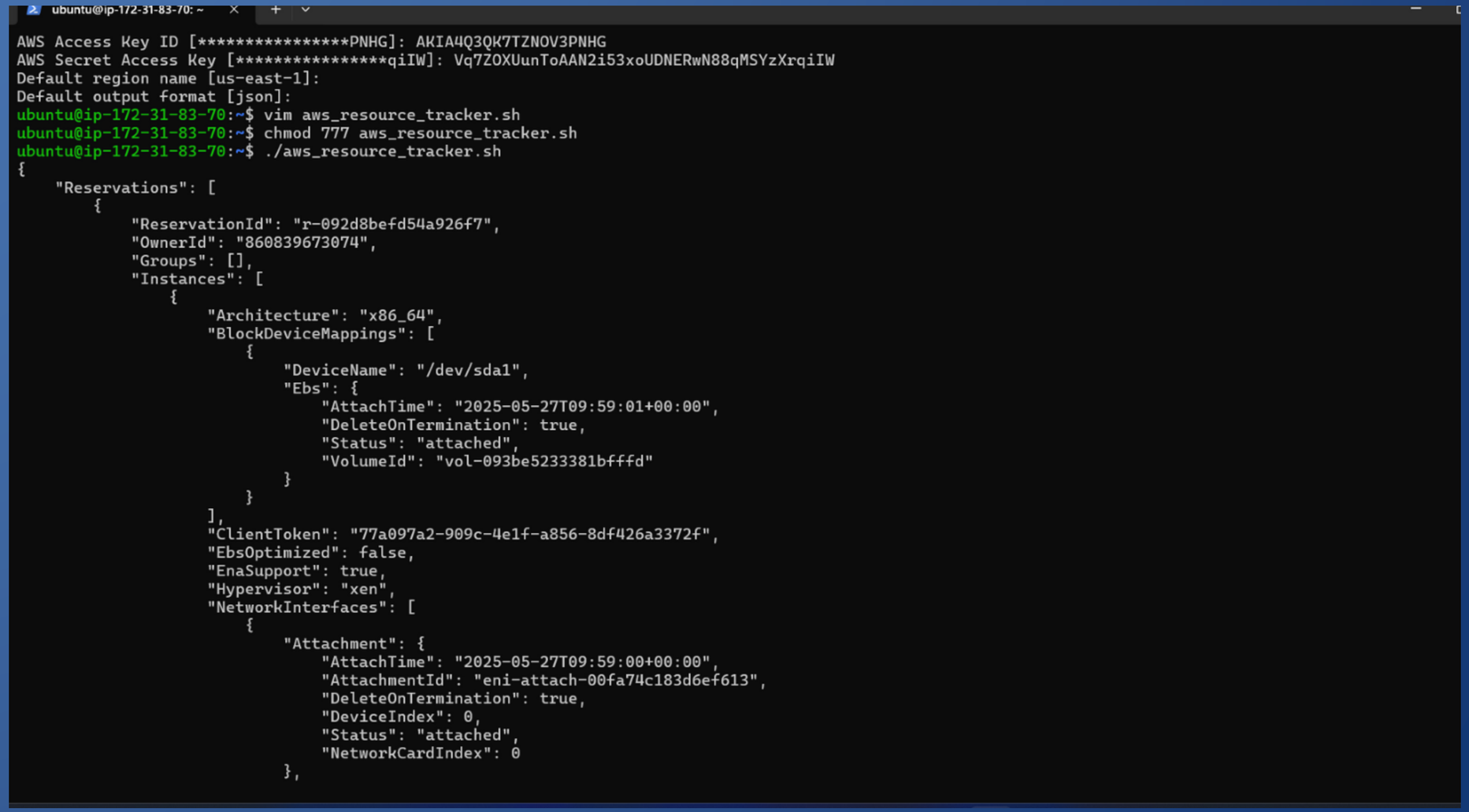
SCRIPT CONTENT - RESOURCE TRACKING

This page displays the initial content of the aws_resource_tracker.sh script. It's a bash script authored by Wasim on June 3rd, version v1, designed to report AWS resource usage. The script includes commands to list S3 buckets (aws s3 ls), describe EC2 instances (aws ec2 describe-instances), list Lambda functions (aws lambda list-functions), and list IAM users (aws iam list-users).



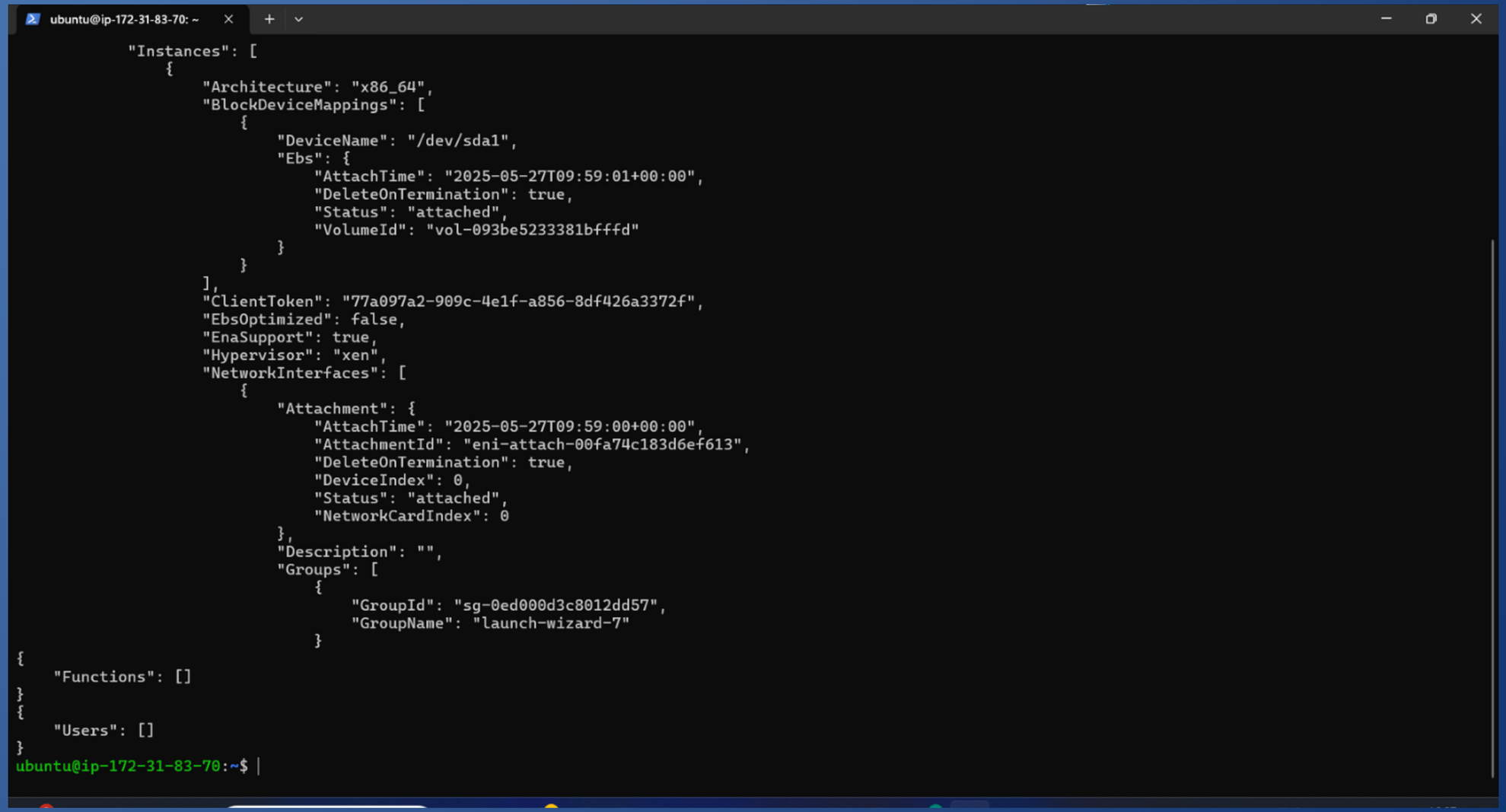
AWS CONFIGURATION & SCRIPT EXECUTION OUTPUT

This page captures the configuration of AWS access keys, secret access keys, default region (us-east-1), and output format (json). Following configuration, the aws_resource_tracker.sh script is made executable with chmod 777 and then executed. The output shows detailed JSON for EC2 instances, including a ReservationId, OwnerId, VolumeId ("vol-093be5233381bfffd"), and GroupId ("sg-0ed000d3c8012dd57").



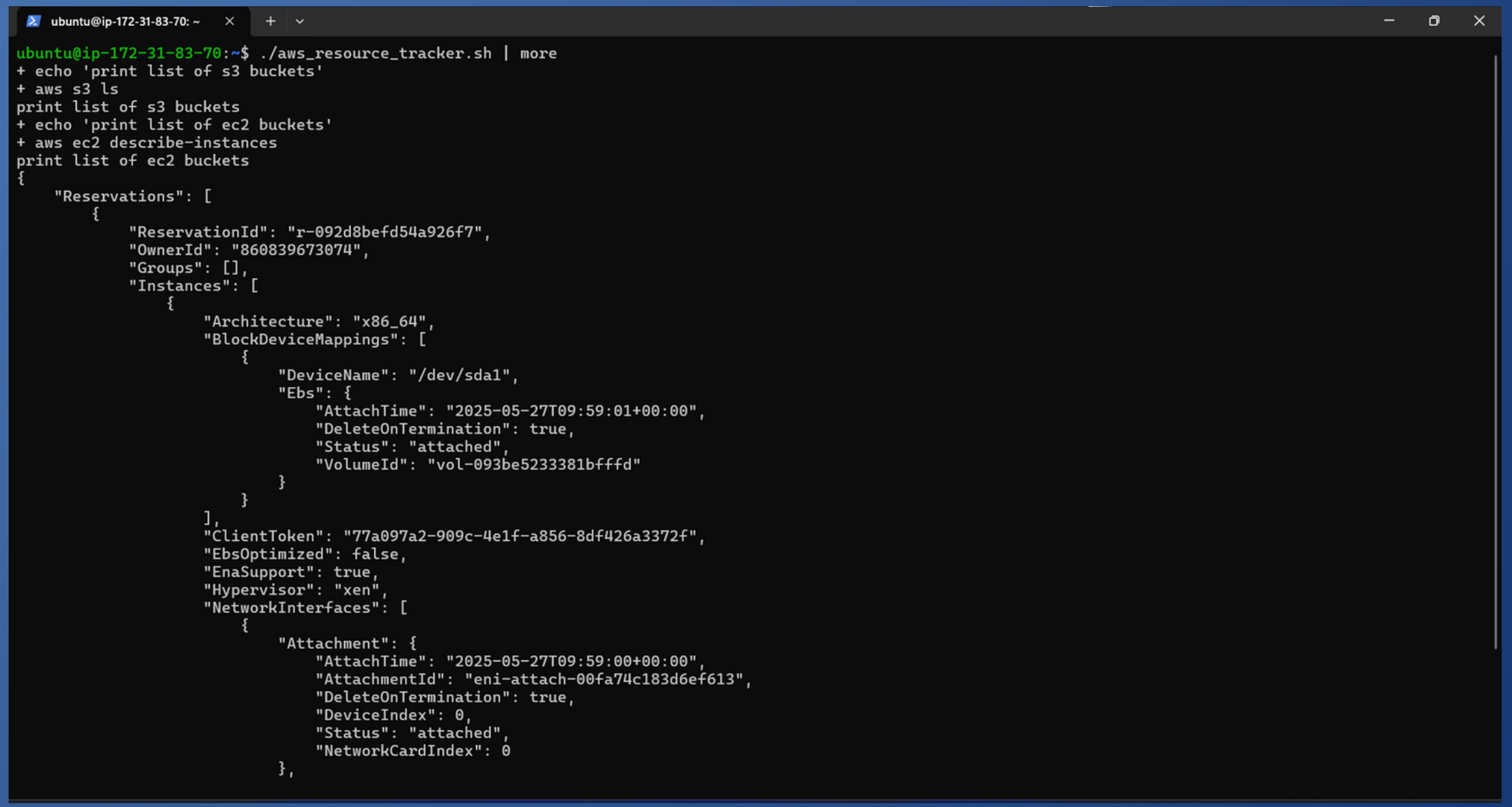
CONTINUED SCRIPT OUTPUT (EC2, LAMBDA, IAM)

This page continues the output from the aws_resource_tracker.sh script. It reiterates details for an EC2 instance, including its Architecture, BlockDeviceMappings, and NetworkInterfaces. It also shows empty arrays for "Functions" (Lambda) and "Users" (IAM), indicating no Lambda functions or IAM users were found at the time of execution.



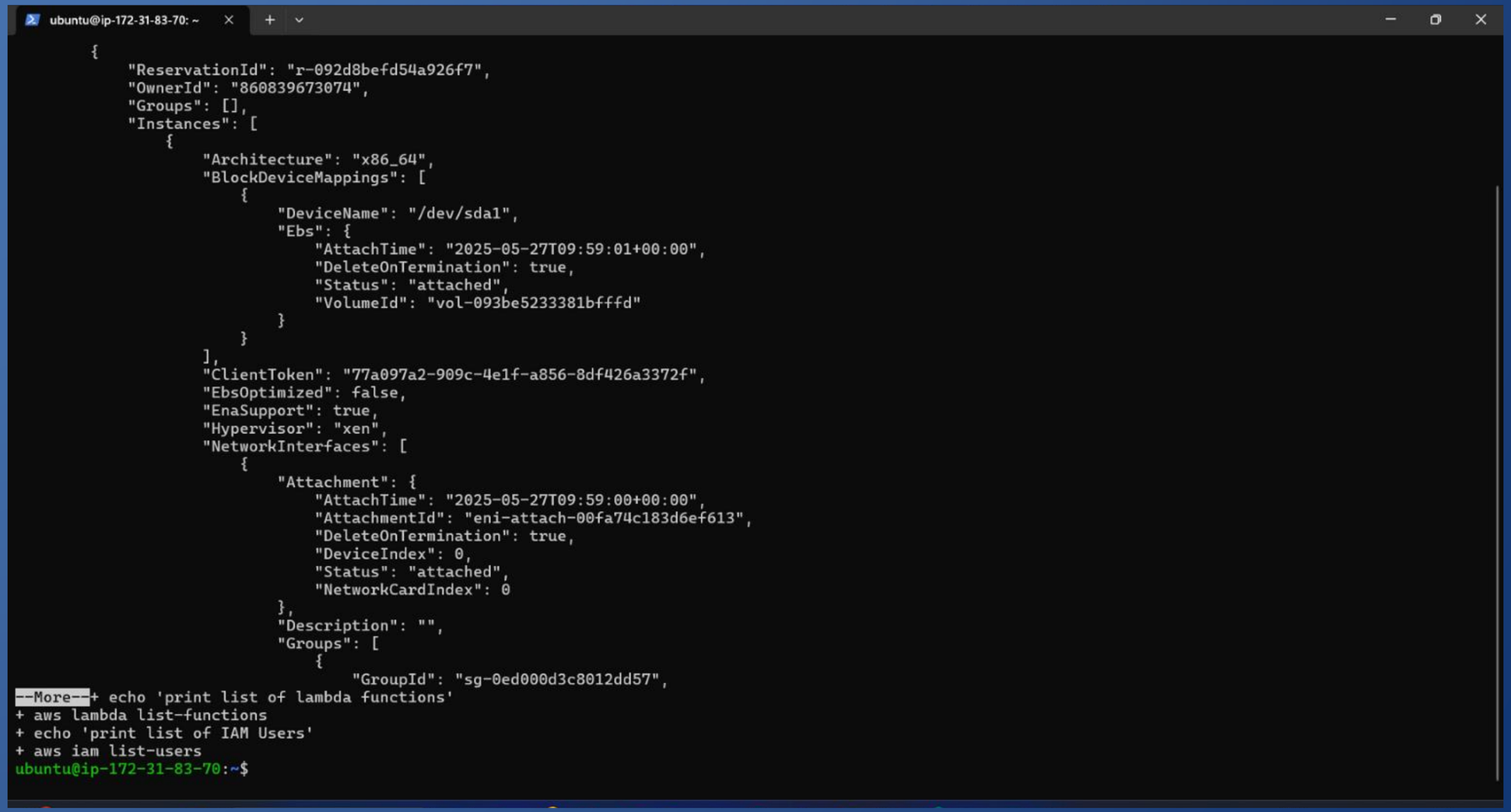
SCRIPT EXECUTION WITH MORE & INITIAL OUTPUT

This page shows the execution of the aws_resource_tracker.sh script piped to more. The set -x command reveals the execution flow, printing "print list of s3 buckets" before running aws s3 ls, and "print list of ec2 buckets" before aws ec2 describe-instances. The page then displays the beginning of the detailed JSON output for EC2 instances.



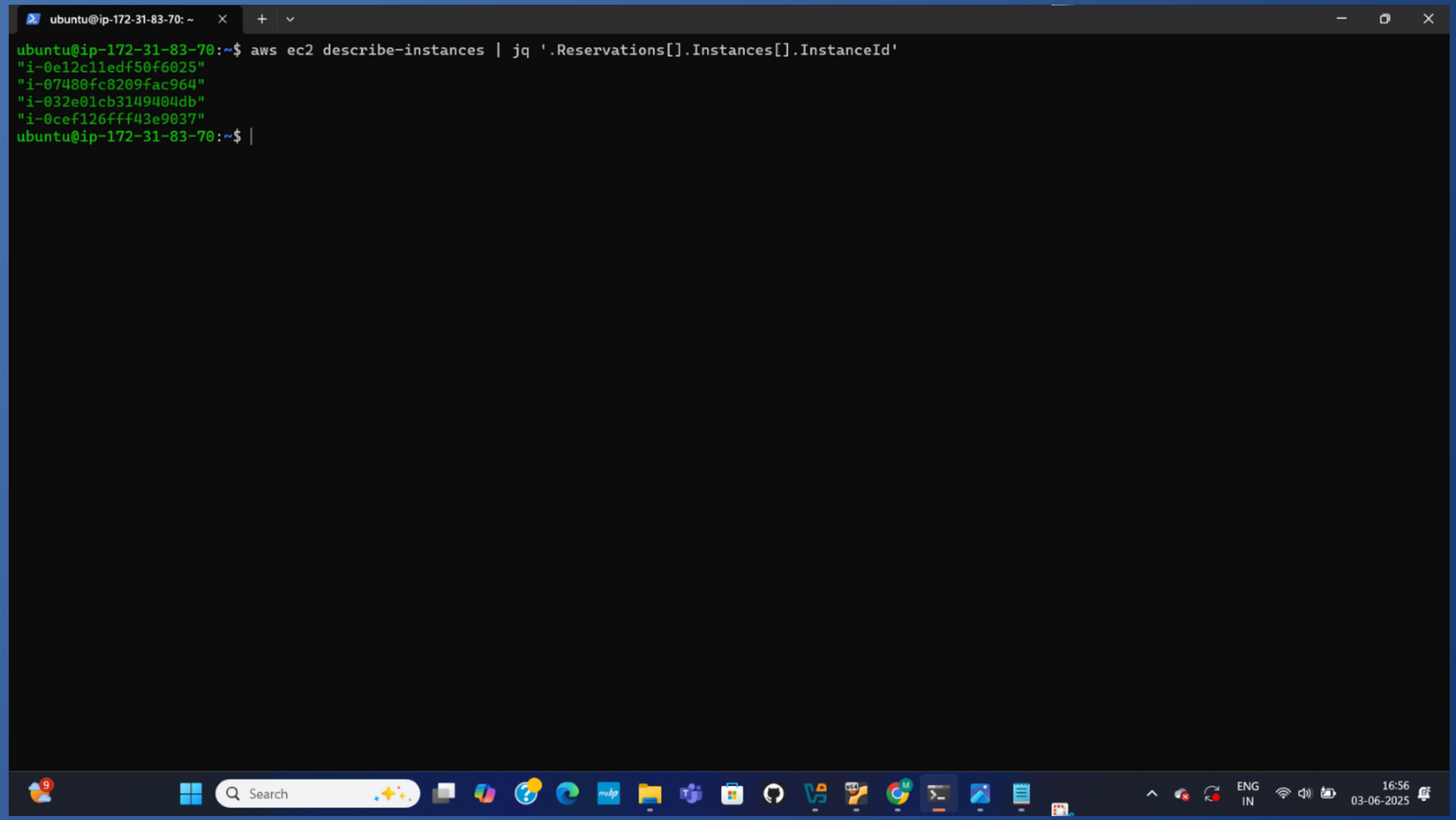
SCRIPT EXECUTION WITH MORE (CONTINUED OUTPUT)

This page is a continuation of the output from the aws_resource_tracker.sh script piped to more. It displays more of the EC2 instance details, including ReservationId, OwnerId, and various instance attributes. It also shows the echo commands for "print list of lambda functions" and "print list of IAM Users" before their respective aws cli commands.



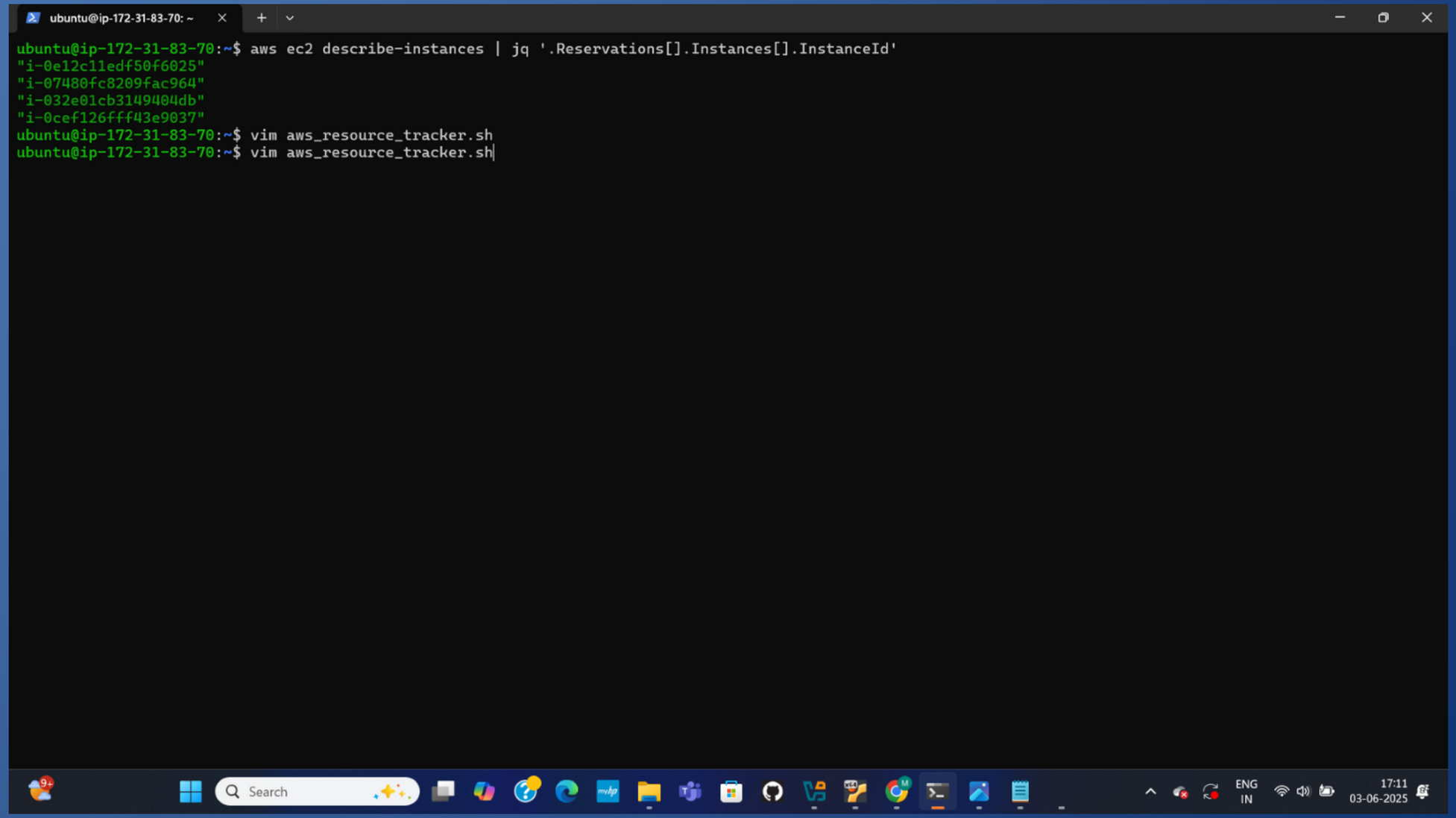
FILTERING EC2 INSTANCE IDS WITH JQ

This page demonstrates filtering EC2 instance IDs from the aws ec2 describe-instances command using jq. The command aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId' successfully extracts and displays four instance IDs: "i-0e12clledf50f6025", "i-07480fc8209fac964", "i-032e01cb3149404db", and "i-0cef126fff43e9037".



RE-EDITING THE SCRIPT

This page shows the re-accessing and presumably re-editing of the aws_resource_tracker.sh script using vim after successfully filtering EC2 instance IDs with jq.



MODIFIED SCRIPT - OUTPUT REDIRECTION

This page displays the modified aws_resource_tracker.sh script. The script now includes set -x for debugging. The output of aws s3 ls is redirected to a file named resourceTracker (> resourceTracker). The jq filter for EC2 instance IDs is now part of the script (aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'). Additionally, the output of aws lambda list-functions is appended to resourceTracker (>> resourceTracker).