 PROJECT TITLE:

Auto-Healer Linux

P. RAKESH SHASHIDHAR

CYBER SECURITY CYBER SECURITY

TAH2706251671 TRAINER

TEKS ACADEMY 96033 43287

9515924411

**📎 Abstract:**

In modern Linux-based systems, it is important to ensure high availability and service uptime, especially for servers and essential background processes. The auto-heelers address this requirement to monitor the Linux project system services by applying a mild, automated solution and to ensure their continuous operation. The project includes a bash script that examines the status of time -time services, automatically restarts to fail, and logs all functions. Additionally, it sends an email alert or system log to inform the administrators of any service obstruction, making this system an active tool for health management. Designed to be simple, efficient and easily configuble, auto-heeler Linux helps reduce manual intervention, reduces downtime, and overall system improves reliability.

📝 Problem Statement:

In many server-based Linux environments, the bedrock of services consists of a web server to respond to requests (e.g., Apache), secure shell access (SSH) to allow remote management, or a scheduling service (Cron) to execute regular tasks. If these one or more of these services fail or become unavailable for some reason (e.g., software bug, system overloaded, configuration issue, etc.), it can severely impact the availability and reliability of your system for users. The traditional approach of a human having to monitor, and possibly restart, the services and/or take corrective actions, will often lead to acceptable amounts of downtime before the response. Systems administrators would have to check the service statuses, and response time delays can vary, oftentimes being a very long time.

There is a clear need for an automated solution that can monitor the critical services, restart them if they stop, and notify the administrator quickly.

This project investigates the space and presents a contribution to solving the problem of service disruption in a multi-server environment with a Bash script-based auto-healing system, working almost completely without human action login.

🎯 Project Objective(s):

* This tool keeps a close eye on essential Linux services like apache2, ssh, and cron, checking in at regular intervals.
* It monitors service activity through the systemctl command and jumps into action right away if something goes wrong with a service.
* If a service fails, it automatically restarts it without needing any human intervention.
* The system administrator gets an email notification whenever a service is found to be inactive or when it has been successfully restarted.
* All activities and events—like service failures, recovery attempts, and both success and failure messages—are logged in a custom log file as well as the system log using the logger command.
* It runs at set intervals using cron, ensuring continuous background monitoring.
* This setup significantly reduces the manual workload for system administrators by automating service recovery and notification processes.

🔍 Project Scope:

* Only use basic Linux tools (bash, systemctl, cron, mail, logger) – no 3rd party libraries or APIs.
* The final script will verify service status, start failed services, and ${LOG} everything with time stamps.
* Optional email notification for when the service fails and you issue a kill/reset.
* No GUI/dashboard interfaces – it’s a console-only tool.
* One does not have to worry about live monitoring of resources (CPU, RAM, disks) and complex alerting systems.
* Only single-node systems are considered; multi-server (distributed monitoring) is left out of scope. · Meant to be simple for light usage - ideal for small academic projects or small Linux usage.

🛠️ Tools & Technologies:

* Bash – primary language for automation reasons, likely this is what you are using.
* systemctl – to monitor and manage the system D service
* cron – to run the monitoring script at regular intervals
* mail / mailx - Send email mail for the administrator for notification.
* logger - to log all system actions and events to the system logs
* Linux OS (Kali) – execution platform on which you are running the script

🕒 Timeline (Tentative):

Week Task

Day 1 Research and determine what is essential to monitor on Linux services

Day 2 Write bash script to monitor and auto-restart service

Day 3 Add logging and send mail features.

Day 4 Cron the script and test it #44;

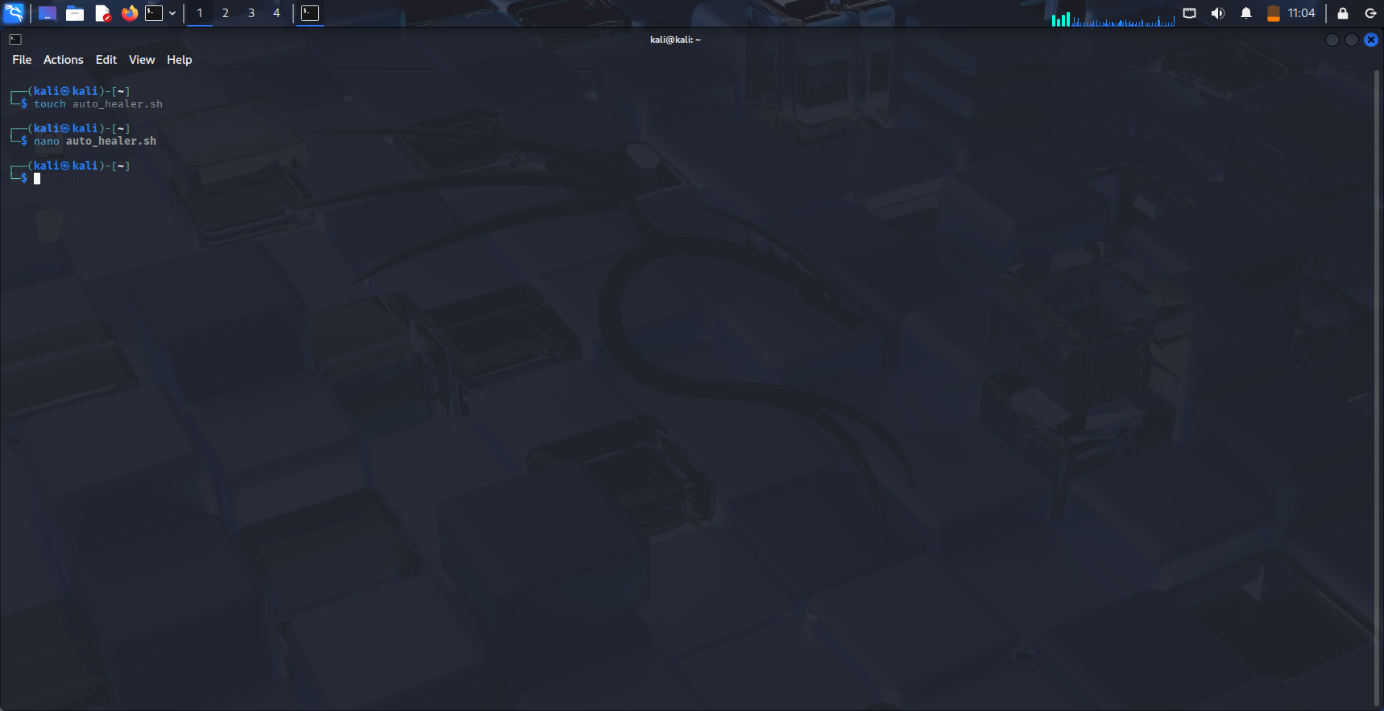
Day 5 Document and Demo the final demo

* **Deliverables:**
* simple bash script for auto-restarting failing services, google around simple
* bash script for auto re-starting failing services; google around, etc
* Log file to record all monitoring checks and restarts of all serviced.
* An e-mail alert system in case anything goes wrong (if you want it)
* cron Job to automatically execute the script at specific intervals
* A brief and concise report of the project including all the details.
* Fast presentation/demo vid to show how it works

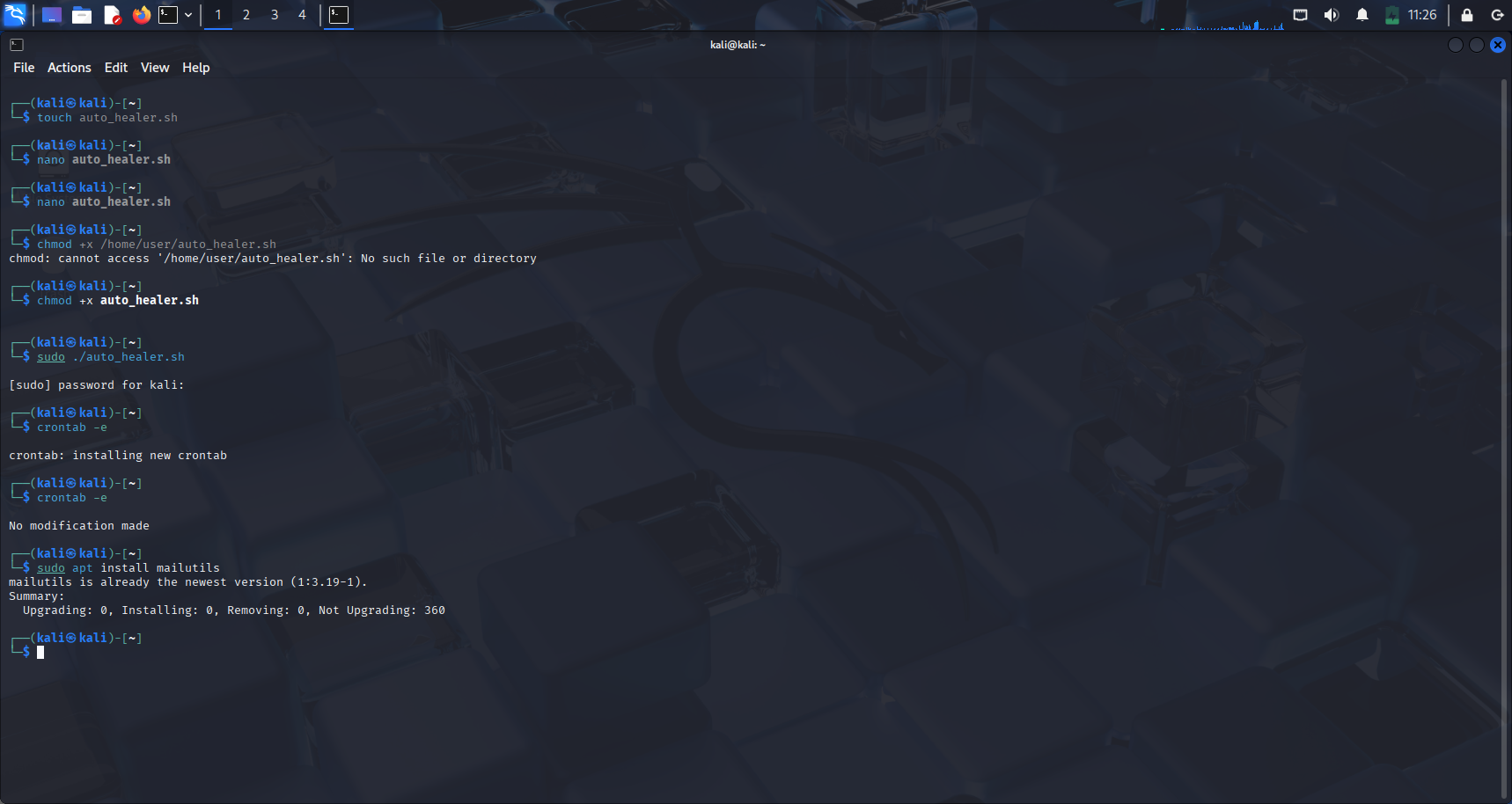
Project implementation:

Step 1: First, we have to install Linux

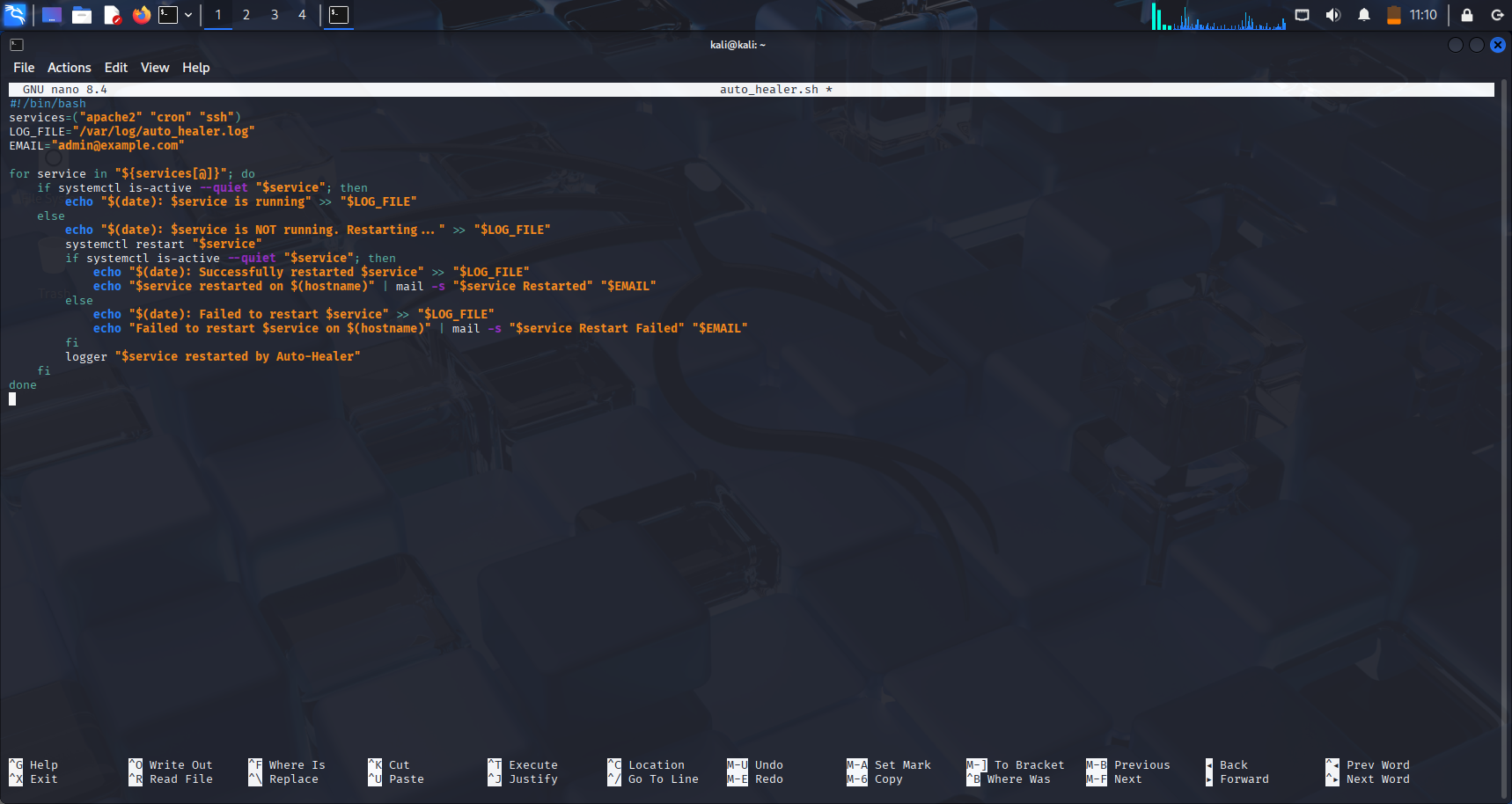


Step 2: Open Terminal & Create Script File

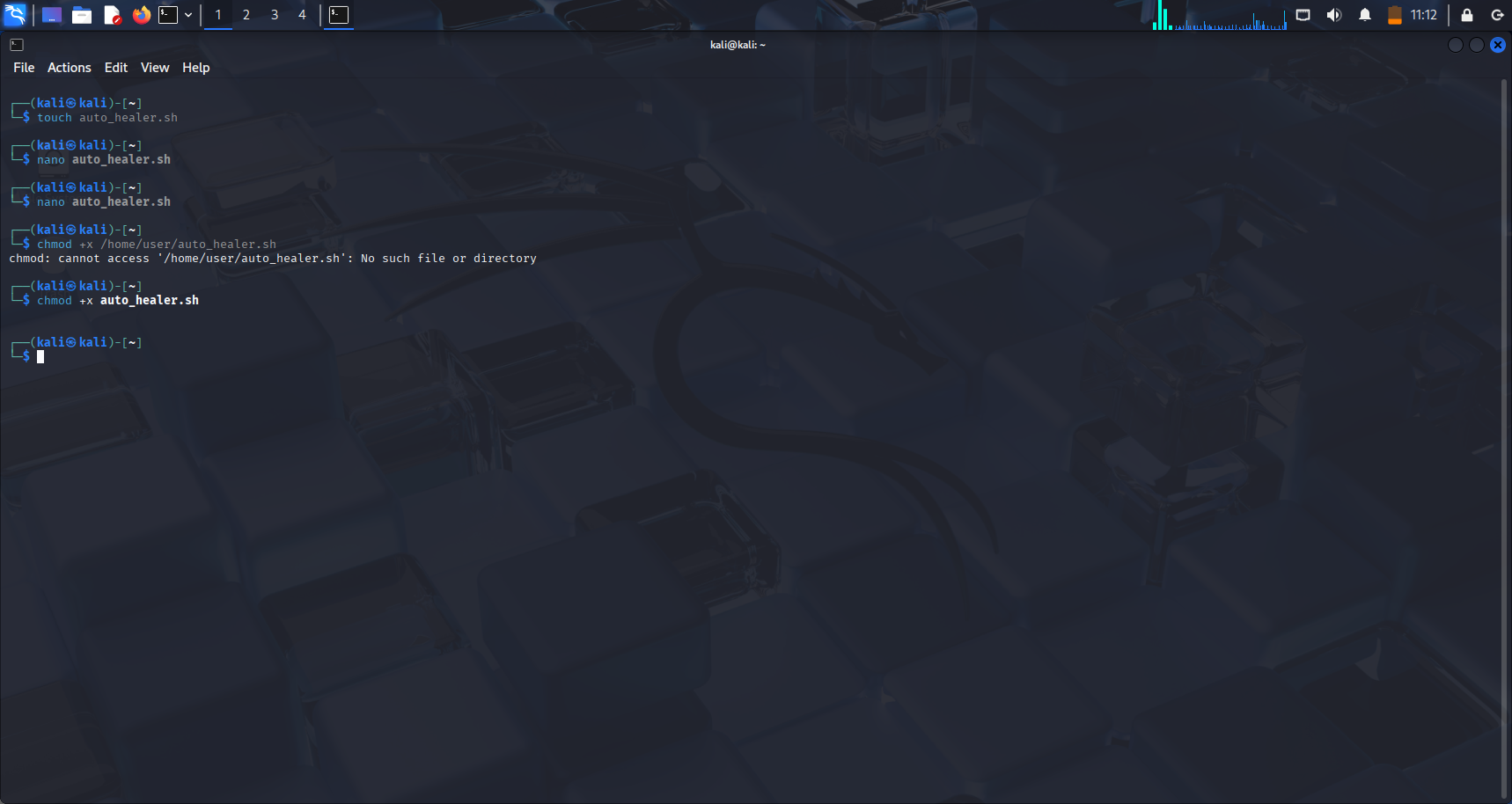
Step 3: Install mailutils



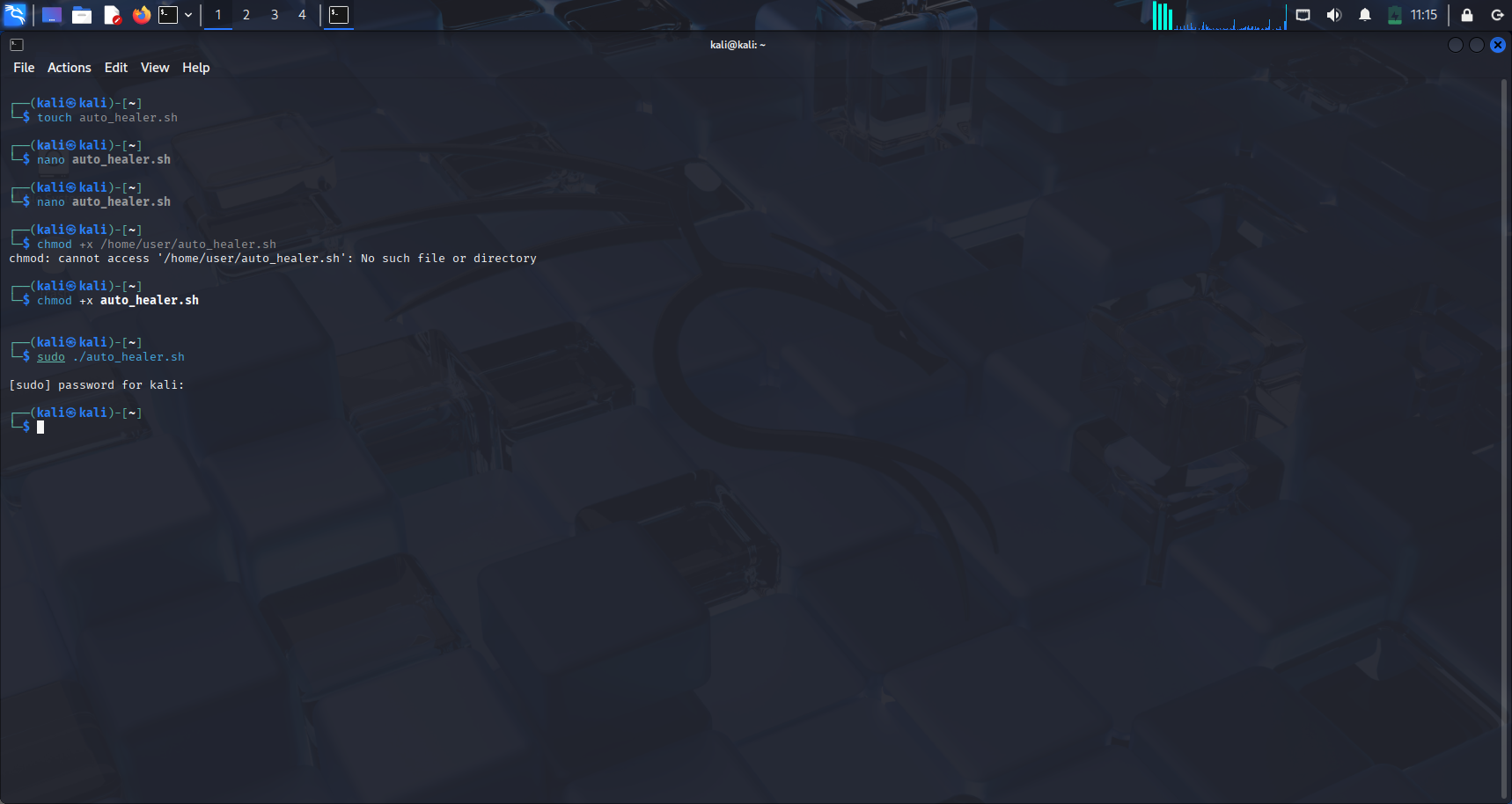
STEP 3: Write the Script



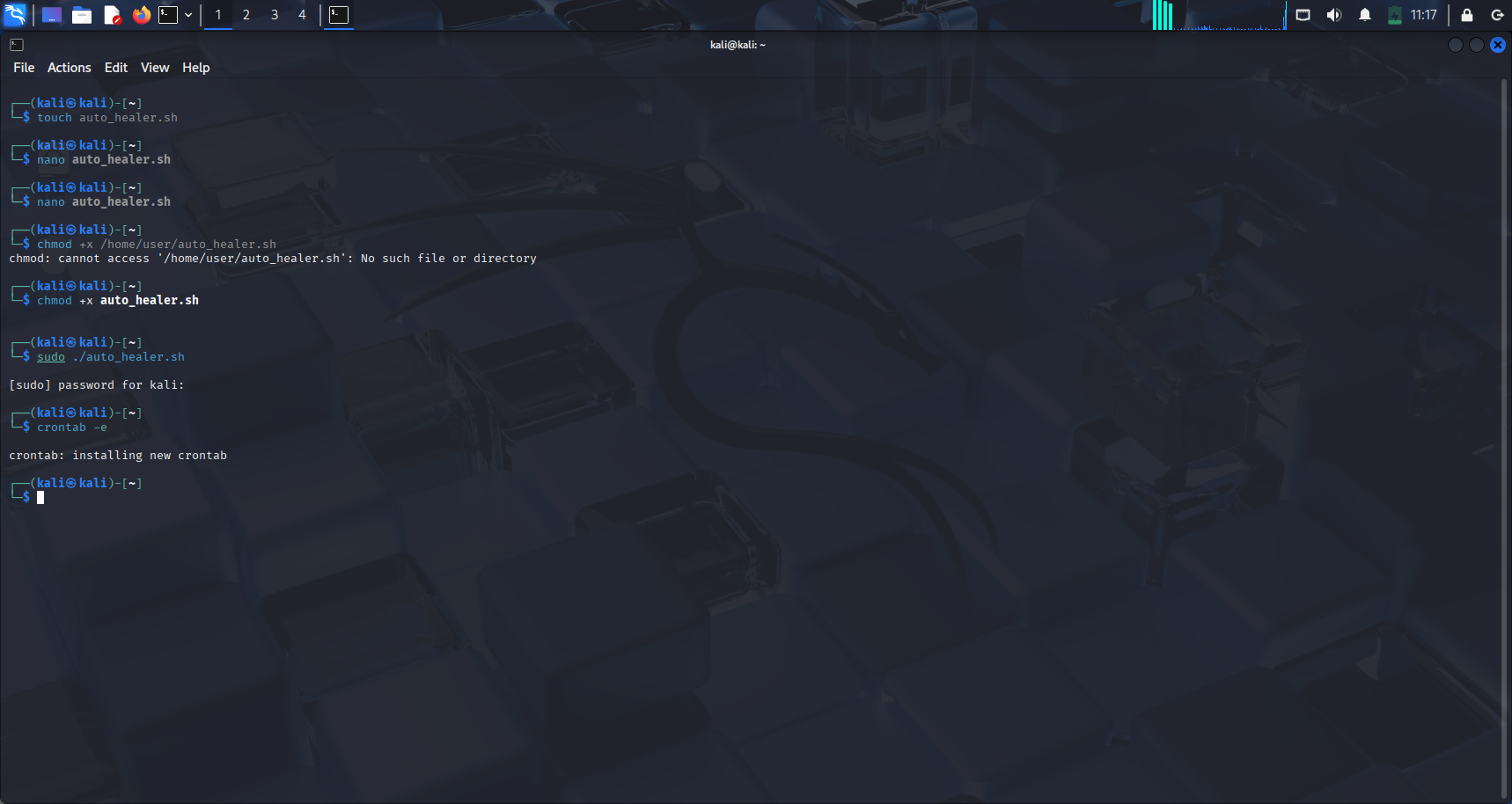
Step 4: Make the Script Executable



Step 4: Run Script Manually



⏱️ Step 7: Schedule Script Using Cron



And the line \*/5 \* \* \* \* /usr/local/bin/auto-healer.sh

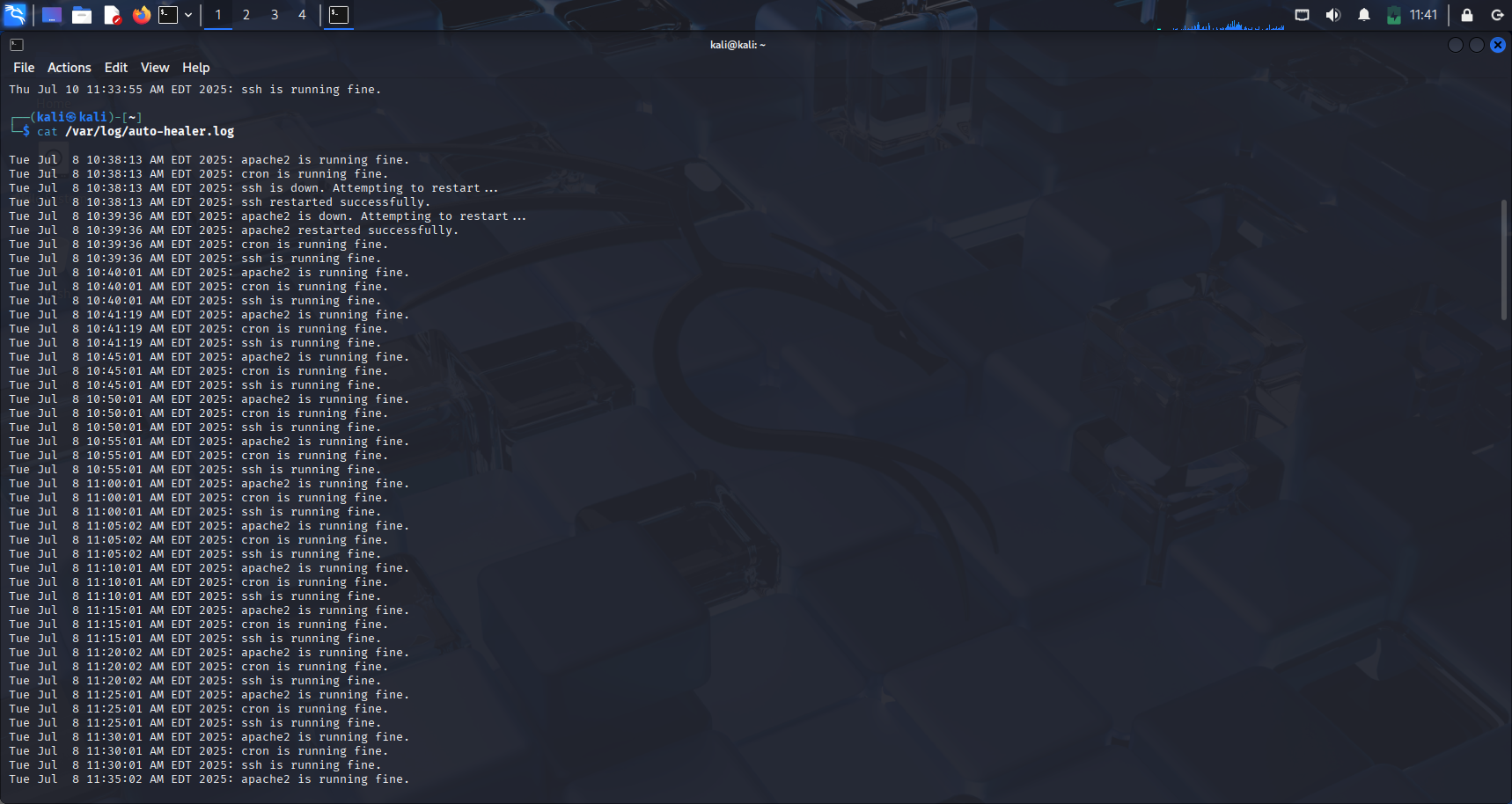


Step 8: All services are running

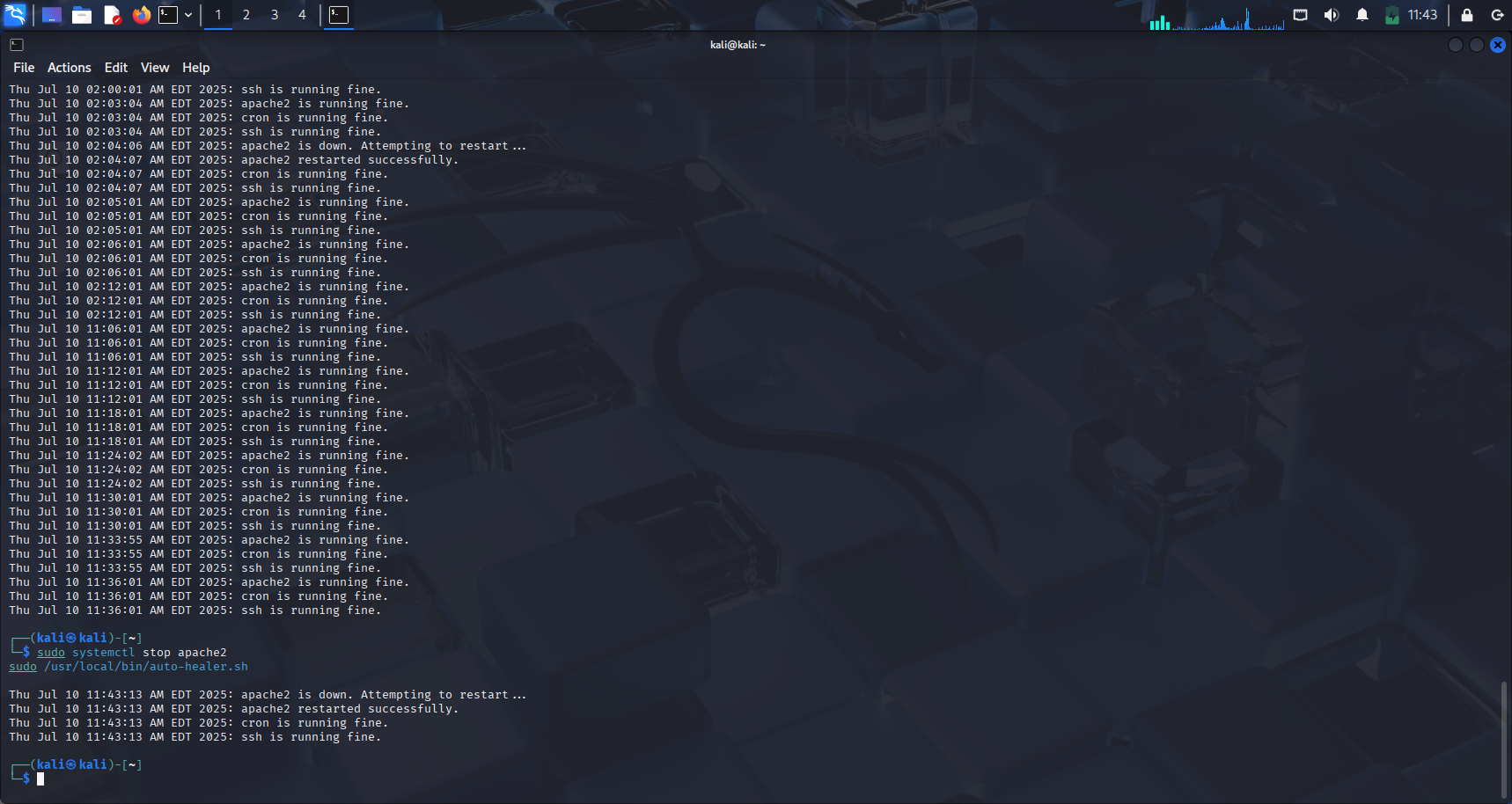
Terminal Output



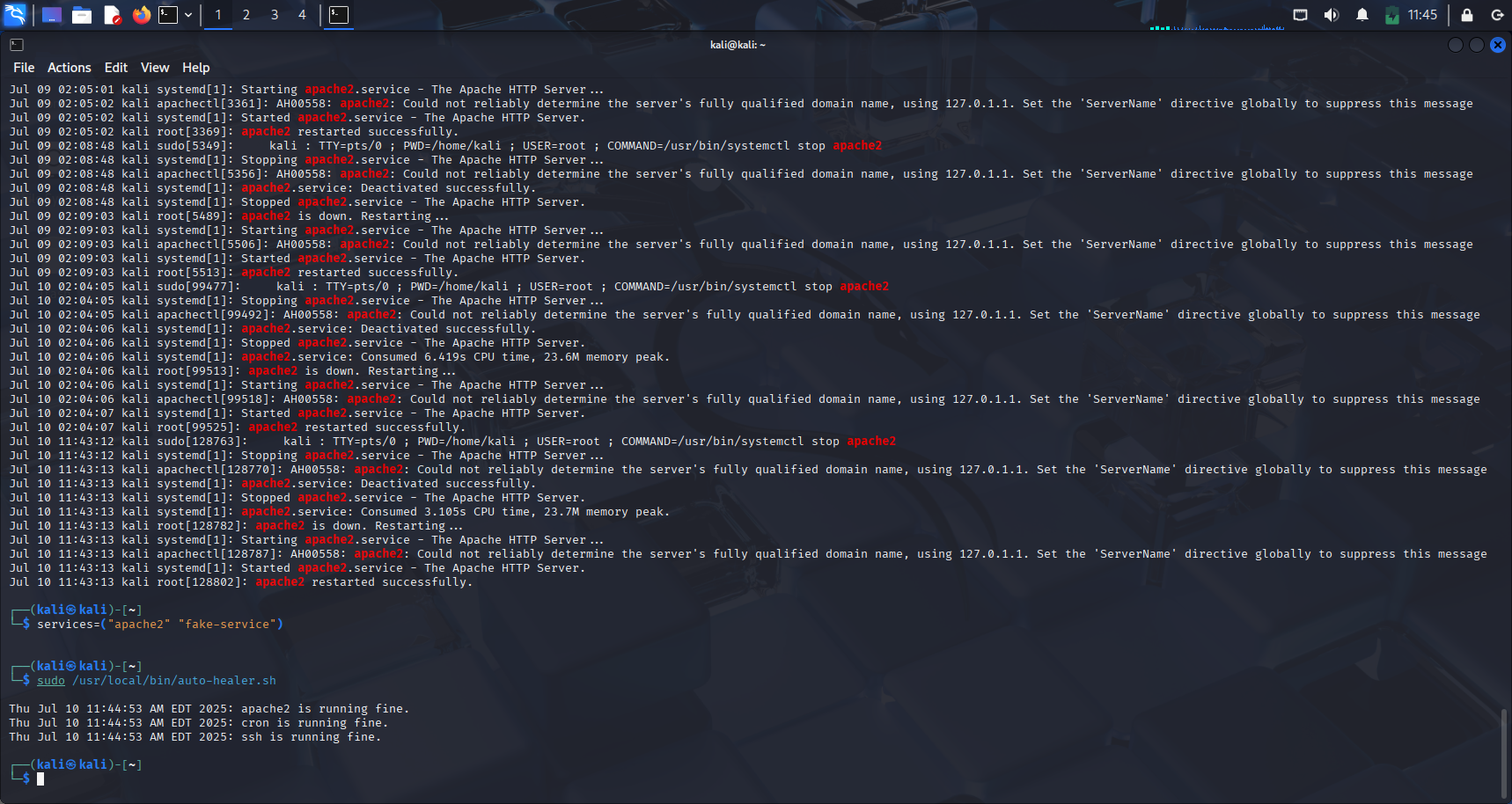
Step 9: Log Outputs



STEP 9: Apache 2 is stopped



STEP 10: Service Fails to Restart (Simulate with fake service)



FRONT END IMPLEMENTATION:

Front-end on the Auto-Heeler Linux website is designed using HTML, CSS, and JavaScript to create a clean, responsible, and user-friendly interface. The main page contains an input field where users can provide the name of a Linux System Service (eg, Apache 2 or MySQL). Under the entrance field, the interactive buttons are: "Check status", "Restart Service", and "Show Logs". When a user clicks on any of these buttons, JavaScript (using the Fetch API) sends a request to the backend, which drives the actual Linux. The results are shown dynamically on the website without loading again, which makes the interface more comfortable. Front-end also includes a rolling log area, where the latest rows of the Linux system log are displayed, which helps the user to look at previous features and errors. Alternative promotion as bootstrap can be added to styling, which gives the interface more polished, modern look

