Lindiwe Mkuzangwe
Konke Maphisa
Zikhona Bam
Siyabonga Mdaweni
Adivhaho Ndou

Table of Contents

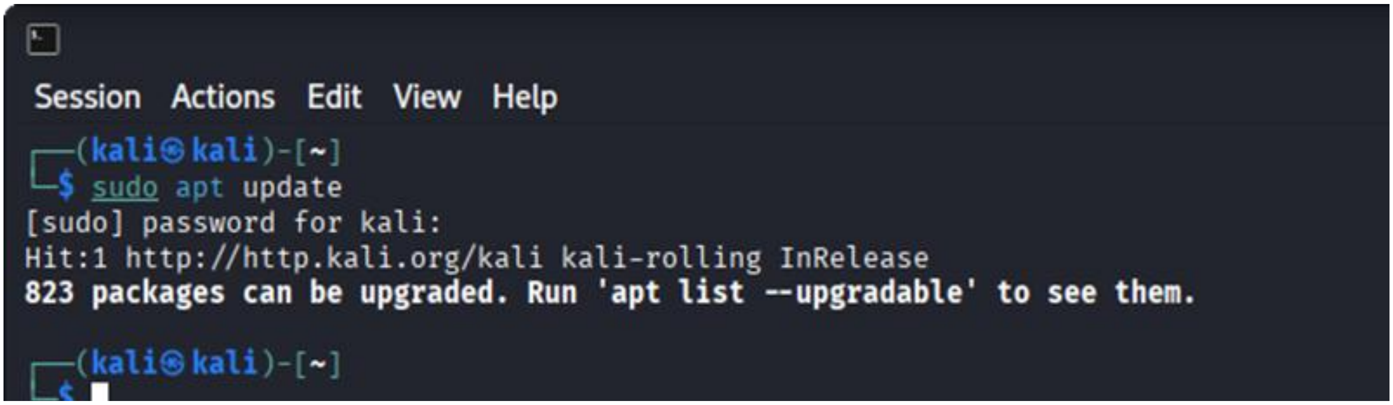
Introduction

Setting up a local server is like building a mini version of the internet on your own computer. It allows you to test websites, run web applications, and learn how servers work all without needing an internet connection or a hosting provider. In this guide, we used **Apache**, one of the most popular and reliable web servers, and **Kali Linux**, a powerful Linux distribution often used for cybersecurity and development tasks.

Installing Apache

We started by installing Apache using the following commands:

sudo apt update

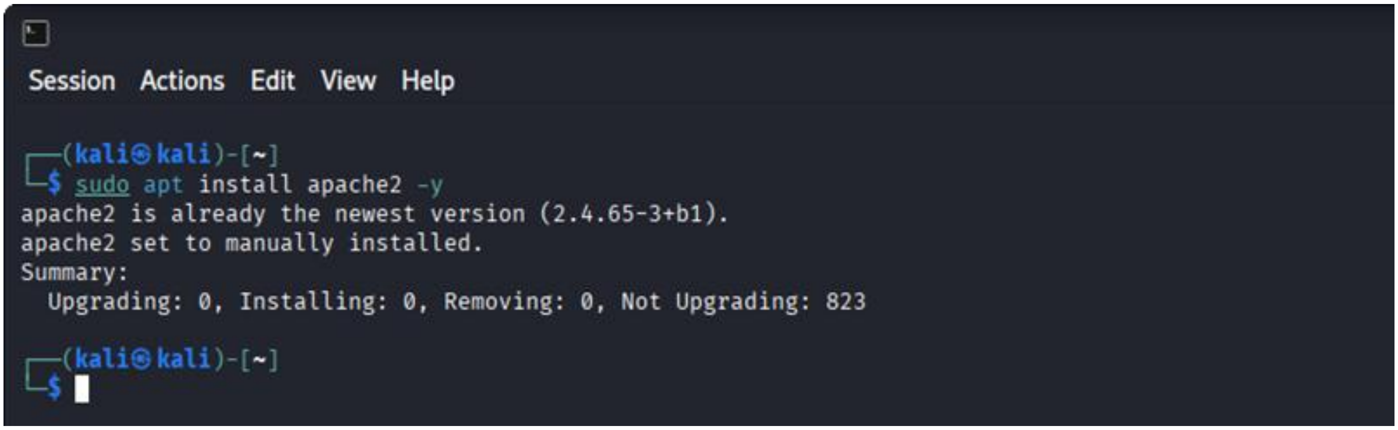


```
┌──(kali㉿kali)-[~]
└─$ sudo apt update
[sudo] password for kali:
Hit:1 http://http.kali.org/kali kali-rolling InRelease
823 packages can be upgraded. Run 'apt list --upgradable' to see them.

┌──(kali㉿kali)-[~]
└─$
```

Figure 1 Updating packages

followed by the **sudo apt install apache2 -y**

This downloaded and set up the Apache web server software. Apache is the tool that listens for requests (like when you open a webpage) and sends back the right files.

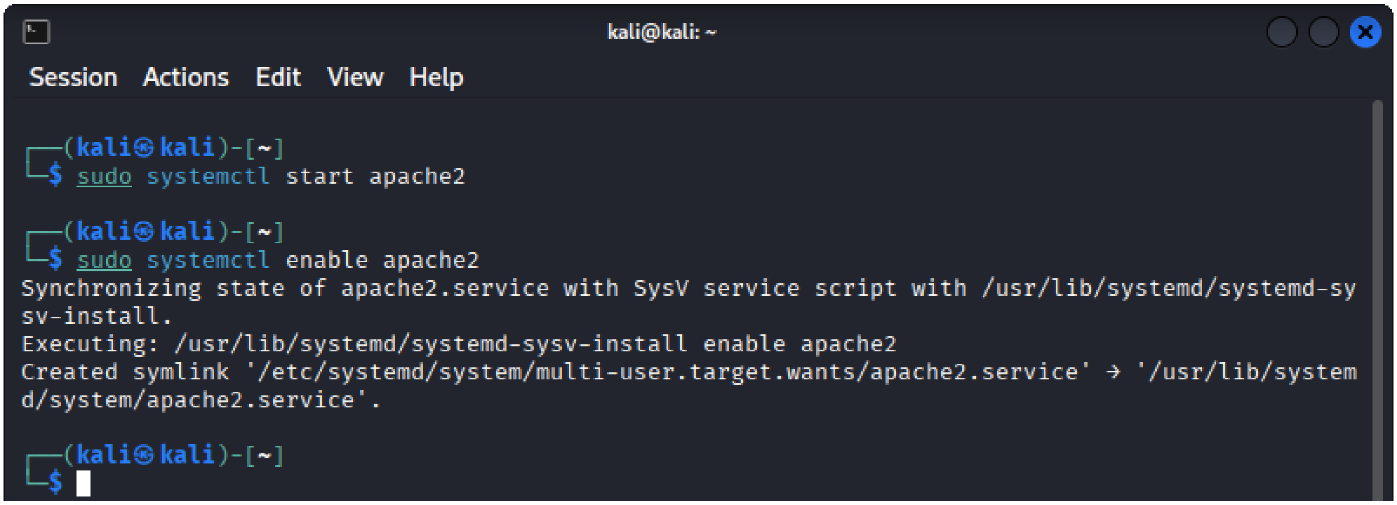


```
┌──(kali㉿kali)-[~]
└─$ sudo apt install apache2 -y
apache2 is already the newest version (2.4.65-3+b1).
apache2 set to manually installed.
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 823

┌──(kali㉿kali)-[~]
└─$
```

Figure 2 Installing Apache web server

Starting and enabling Apache

Using the **sudo systemctl start apache2** we enable Apache followed by **the sudo systemctl enable apache2.** This started the Apache service and made sure it would automatically start every time the computer boots up.

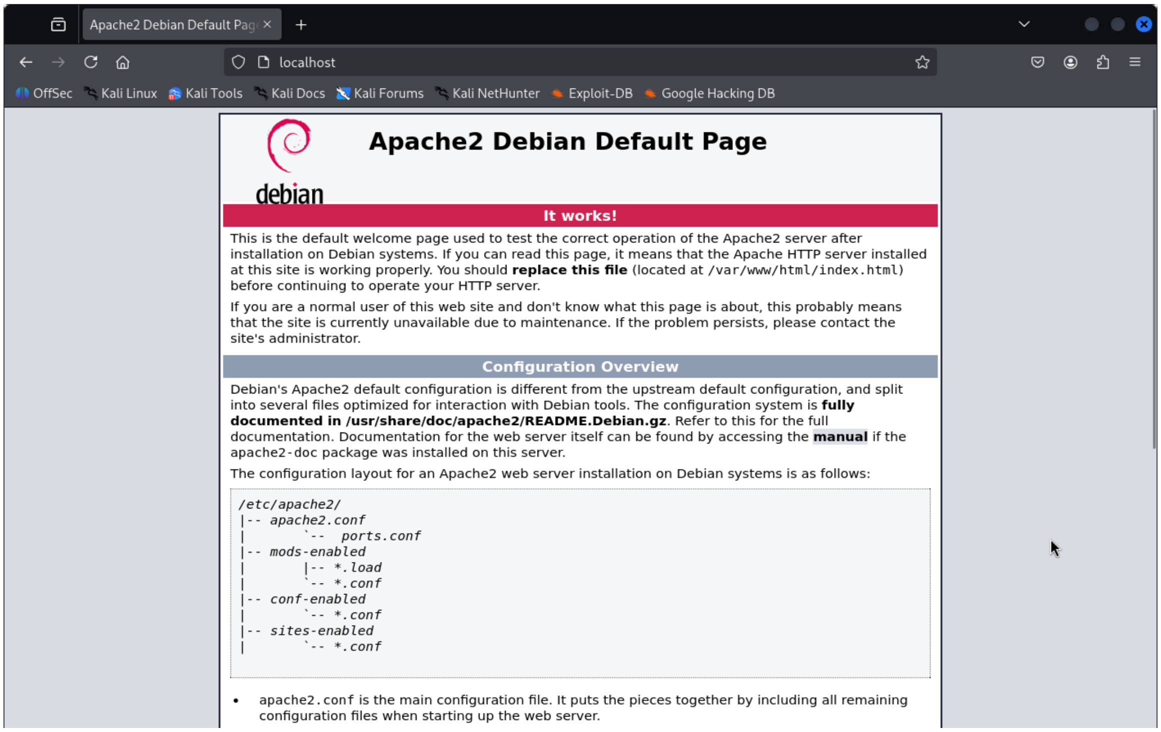


```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl start apache2

┌──(kali㉿kali)-[~]
└─$ sudo systemctl enable apache2
Synchronizing state of apache2.service with SysV service script with /usr/lib/systemd/systemd-sy
sv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable apache2
Created symlink '/etc/systemd/system/multi-user.target.wants/apache2.service' → '/usr/lib/system
d/system/apache2.service'.

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 3 Starting and enabling Apache

Testing the webserver

On the web browser we typed http://localhost and the Apache2 default webpage was displayed.



Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

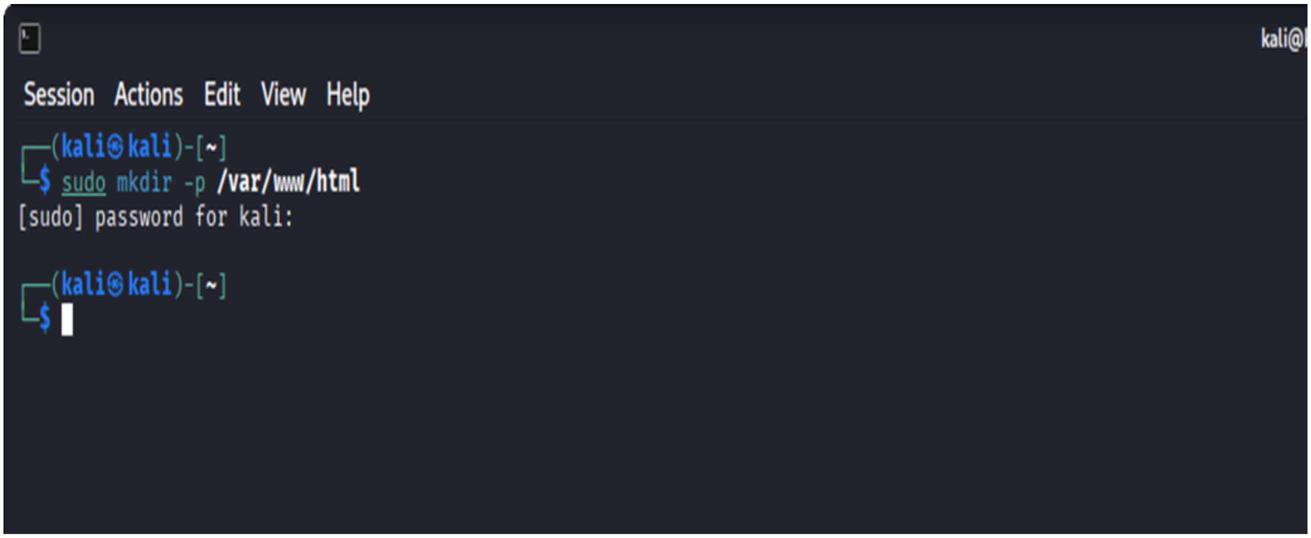
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.

Figure 4 Testing default Apache webpage

Creating the default web root directory

We checked if the default folder where Apache serves files from—/var/www/html—existed. This directory will allow us to open our html file in the local server and this is where we will save our custom web page.

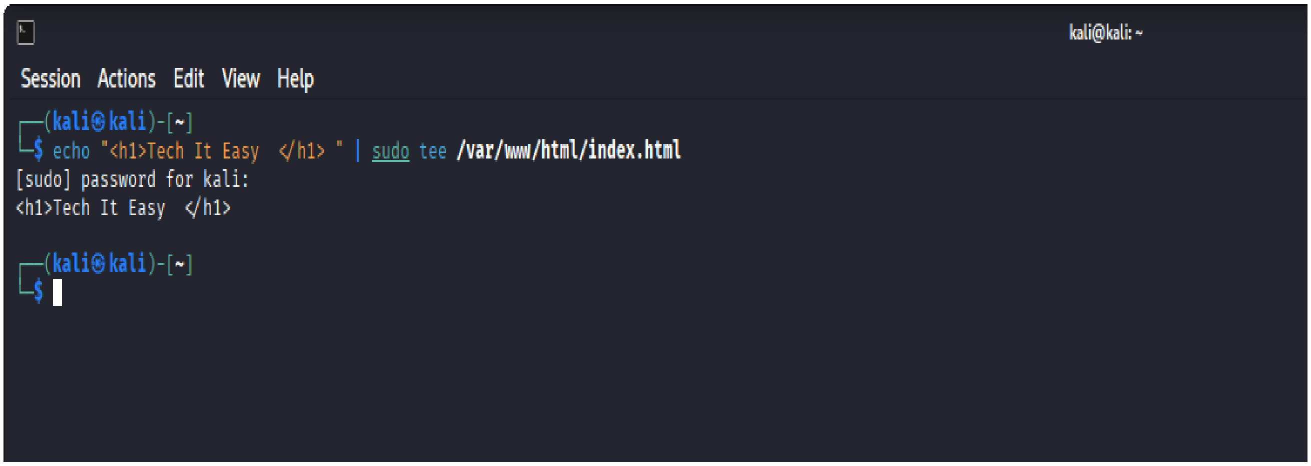


```
┌──(kali㉿kali)-[~]
└─$ sudo mkdir -p /var/www/html
[sudo] password for kali:

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 5 default web directory command

Creating and adding a test web page

We created a simple HTML file which was a basic webpage to confirm everything was working. When you opened http://localhost in your browser, Apache served this file just like a real website. Using the following command **echo "<h1>Tech It Easy</h1>" | sudo tee /var/www/html/index.html**

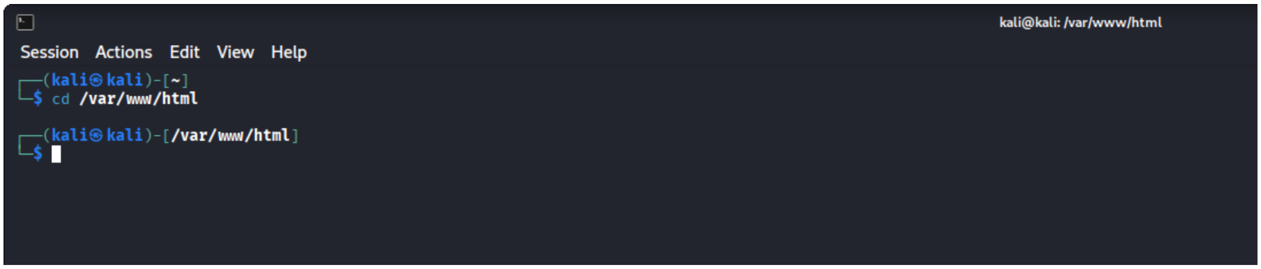


```
┌──(kali㉿kali)-[~]
└─$ echo "<h1>Tech It Easy  </h1> " | sudo tee /var/www/html/index.html
[sudo] password for kali:
<h1>Tech It Easy  </h1>

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 6 simple Html code

Customizing the webpage

We used the following command **cd /var/www/html** to make sure that the folder exists.

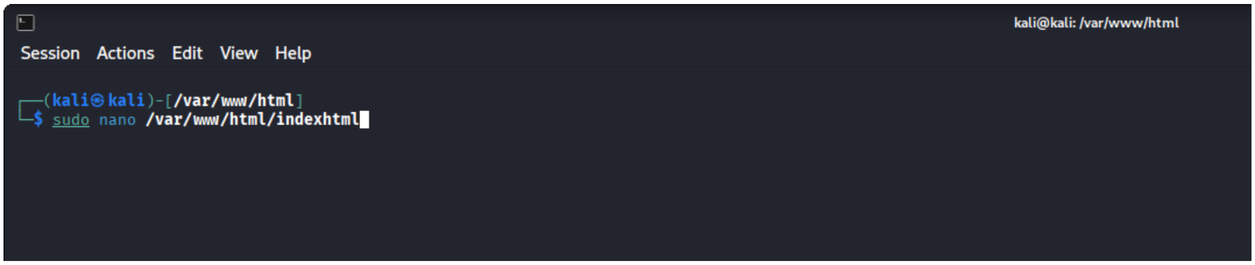


```
┌──(kali㉿kali)-[~]
└─$ cd /var/www/html

┌──(kali㉿kali)-[/var/www/html]
└─$ █
```

Figure 7 verifying folder exists

We then went on to type the following command **sudo nano /var/www/html/index.html**

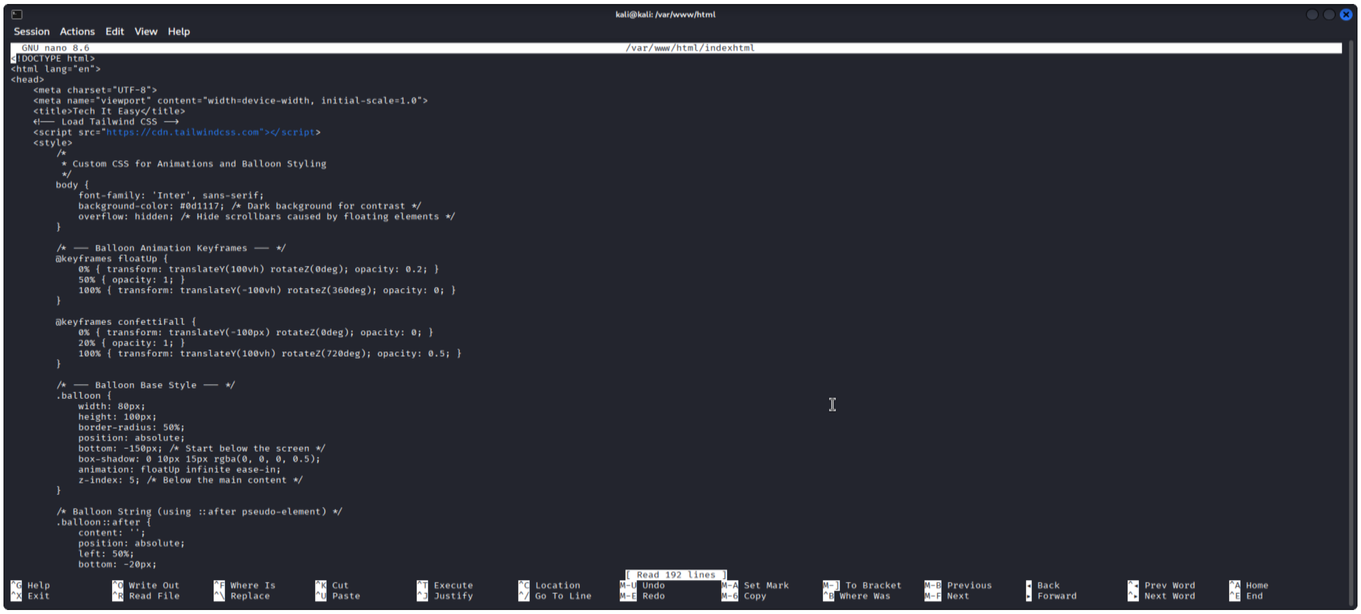
to open the kali built in Nano text editor.



```
┌──(kali㉿kali)-[/var/www/html]
└─$ sudo nano /var/www/html/indexhtml█
```

Figure 8 Opening nano Text editor

Nano file editor now opens an then customize the webpage. Then pressed CTRL + O to save.

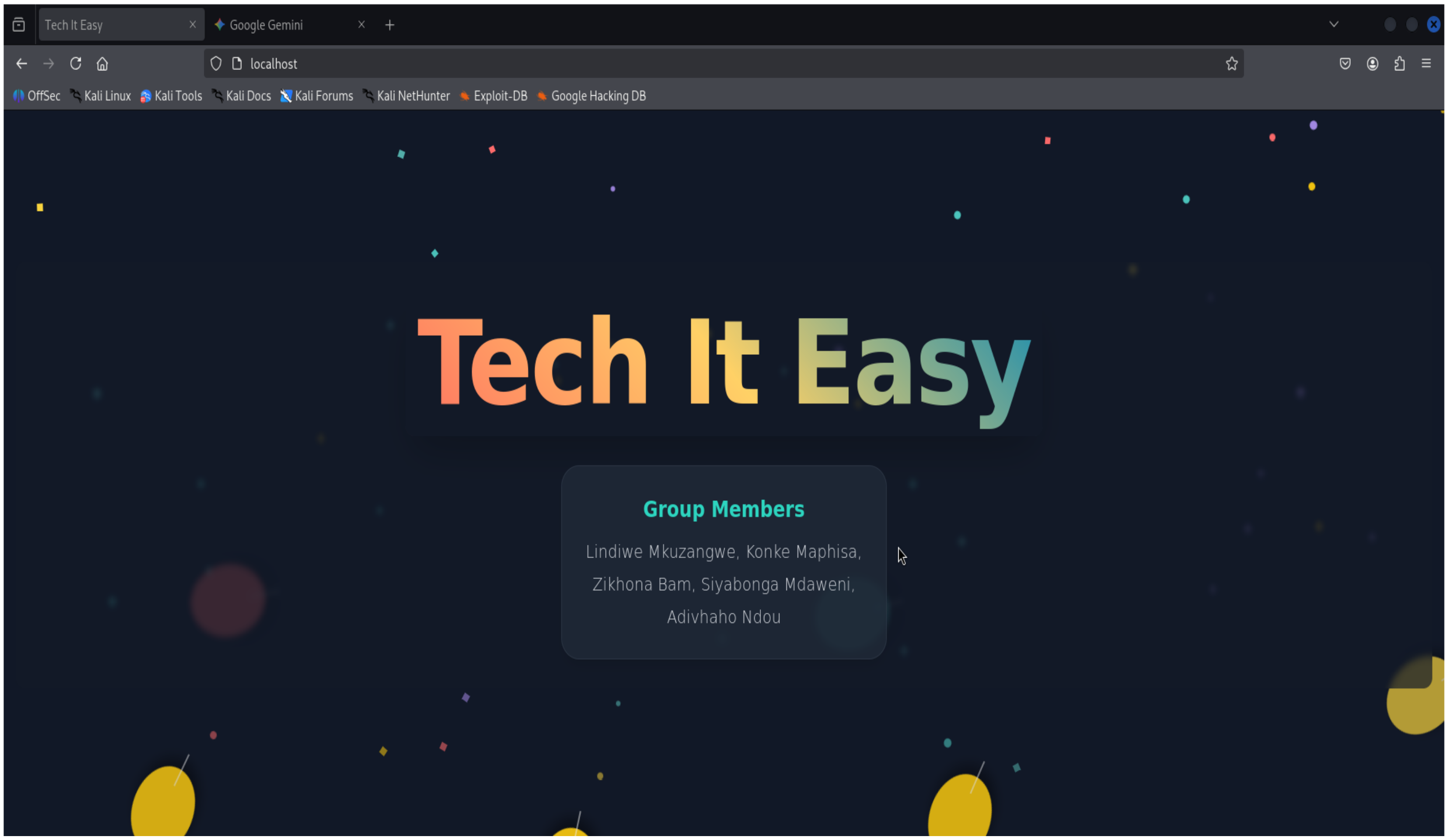


```
  GNU nano 8.6                                         /var/www/html/indexhtml
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tech It Easy</title>
    <!── Load Tailwind CSS ──>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        /*
         * Custom CSS for Animations and Balloon Styling
         */
        body {
            font-family: 'Inter', sans-serif;
            background-color: #0d1117; /* Dark background for contrast */
            overflow: hidden; /* Hide scrollbars caused by floating elements */
        }

        /* ── Balloon Animation Keyframes ── */
        @keyframes floatUp {
            0% { transform: translateY(100vh) rotateZ(0deg); opacity: 0.2; }
            50% { opacity: 1; }
            100% { transform: translateY(-100vh) rotateZ(360deg); opacity: 0; }
        }

        @keyframes confettiFall {
            0% { transform: translateY(-100px) rotateZ(0deg); opacity: 0; }
            20% { opacity: 1; }
            100% { transform: translateY(100vh) rotateZ(720deg); opacity: 0.5; }
        }

        /* ── Balloon Base Style ── */
        .balloon {
            width: 80px;
            height: 100px;
            border-radius: 50%;
            position: absolute;
            bottom: -150px; /* Start below the screen */
            box-shadow: 0 10px 15px rgba(0, 0, 0, 0.5);
            animation: floatUp infinite ease-in;
            z-index: 5; /* Below the main content */
        }

        /* Balloon String (using ::after pseudo-element) */
        .balloon::after {
            content: '';
            position: absolute;
            left: 50%;
            bottom: -20px;
                                              [ Read 192 lines ]
^G Help       ^O Write Out  ^F Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket M-B Previous  ◂ Back        ^◂ Prev Word  ^A Home
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^B Where Was   M-F Next      ▸ Forward     ^▸ Next Word  ^E End
```

Figure 9 Nano Text editor

Viewing the customized webpage

After typing http://localhost on the web browser we saw our custom message proof that our local server was working perfectly.



Tech It Easy

Group Members

Lindiwe Mkuzangwe, Konke Maphisa, Zikhona Bam, Siyabonga Mdaweni, Adivhaho Ndou

Figure 10 customized html file

Task summary

Setting up a local Apache server on Kali Linux wasn't just about typing commands, it was a hands on experience that gave us a real feel for how websites actually work behind the scenes. We started by installing Apache, which is basically the software that turns your computer into a mini web server. Once it was up and running, we used a few system commands to make sure everything was working smoothly. Then came the part where we created the main folder where my website files would live /var/www/html. Think of it like the front door of your site: whatever you put in there is what people (or in this case, your browser) will see when they visit. We added a customized HTML file which has the group name and group members names we also added some animations , and when we opened http://localhost in the browser, there it was our very own webpage, served directly from the machine. Along the way, we learned how Apache handles requests from browsers, how Linux uses systemctl to manage services like Apache, and how important file structure and permissions are when it comes to web development. We also got a better understanding of how to check if a server is running, how to fix missing folders, and how everything connects from the server software to the browser.