

Beginners To Advance Termux

A Complete Guide Using Mobile

Written By: Mahedi Hasan Rafsun (Founder of LWMHR)

"Unlock the power of Linux on Android!"

Organization: LWMHR

Published: 2025

From: Bangladesh

Email: hackwithhexcracker@gmail.com

Website: <https://weblearnerprosite.blogspot.com>

Tools Used: AI (ChatGPT), Grammarly, Canva

Language: English + Bengali (Dual-Language Option)

"Learn Termux, Code in Bash, Run Linux, and build CLI tools — all from your phone!"

Author Bio:

Mahedi Hasan Rafsun is an 18-year-old self-taught developer, ethical hacker, and mobile-first tech educator from Bangladesh. He is the founder of LWMHR and WebLearner Pro, two platforms dedicated to teaching programming, cybersecurity, Termux, and AI using simple language and mobile-friendly tools. With expertise in web development, Linux, Node.js, Python, and Termux scripting, Rafsun empowers learners without a computer to become real developers.

Organization Info:

Name: LWMHR (Learn With Mahedi Hasan Rafsun)

Website: <https://weblearnerprosite.blogspot.com>

Telegram Bot: https://t.me/WebLearner_Pro_bot

Email: hackwithhexcracker@gmail.com

Goal: Make programming, Termux, and ethical hacking accessible on Android devices using Bengali + English.

Table of Contents

Chapter	Title	Category
1	Introduction to Termux	Basics
2	How to Install Termux on Android (F-Droid method)	Basics
3	Understanding the Termux File System	Basics
4	Basic Linux Commands (ls, cd, pwd, mkdir, etc.)	Basics
5	Installing Packages & Using apt, pkg, dpkg	Basics
6	Storage Access & Linking Folders	Basics
7	Customizing Termux with Themes, Fonts, and Prompt	Intermediate
8	Shell Scripting Basics in Termux	Intermediate
9	Git & GitHub Usage (clone, commit, push)	Intermediate
10	Create Your Own CLI Tools	Intermediate
11	Python Programming in Termux	Programming
12	Node.js & JavaScript Projects	Programming
13	Hosting Servers in Termux (PHP, HTTP, Python)	Advanced
14	Ethical Hacking Tools in Termux (legal use only)	Advanced
15	SSH & Remote Server Access with Termux	Advanced
16	VNC Viewer: GUI Linux in Termux	Advanced
17	Using Termux API (Camera, SMS, Battery)	Advanced
18	Backup & Restore Full Termux Setup	Advanced
19	Full Projects: Tools, Bots, Servers	Expert
20	Automation, AI Tools & Final Mastery	Expert

Chapter 1: Introduction to Termux

Introduction

Welcome to the world of Termux! If you've ever wanted to experience the power of a Linux command line environment directly on your Android device, Termux is your gateway. It's a free and open-source terminal emulator and Linux environment app that works directly with no rooting or special setup required. Termux provides a minimal base system, and you can install additional packages using its package manager. This means you can run a wide range of Linux tools and utilities, develop software, and even host simple web servers, all from your phone or tablet.

Termux is incredibly versatile. It bridges the gap between mobile convenience and the robust capabilities of a desktop Linux system. Whether you're a student, a developer, an ethical hacker, or just curious about Linux, Termux offers a unique and accessible platform to learn and experiment.

Step-by-step guide

This chapter will introduce you to the core concepts of Termux and prepare you for its installation and usage. Think of Termux as a mini-Linux distribution running within your Android device. It doesn't modify your Android operating system; instead, it creates a separate, self-contained environment where you can execute Linux commands and run various programs.

Key Concepts:

1. **Terminal Emulator:** Termux provides a command-line interface (CLI) where you type commands and receive text-based output. This is similar to the Command Prompt on Windows or Terminal on macOS/Linux.
2. **Linux Environment:** Within Termux, you're essentially running a Debian-based Linux environment. This gives you access to many standard Linux tools and utilities.
3. **Package Manager:** Termux uses `pkg` (a wrapper around `apt` and `dpkg`) to install software packages. This is how you'll add new tools and programming languages to your Termux environment.
4. **No Root Required:** One of the biggest advantages of Termux is that it works without needing to root your Android device. This makes it safe and easy to use for everyone.

Important commands in code blocks

While we haven't installed Termux yet, it's good to familiarize yourself with some fundamental commands you'll use frequently. These are common Linux commands that will be your bread and butter in Termux.

```
# This command lists the contents of the current directory
ls

# This command changes the current directory to 'my_folder'
cd my_folder

# This command shows your current working directory
pwd

# This command creates a new directory named 'new_directory'
mkdir new_directory
```

Output examples if possible

Since we haven't installed Termux, we can't show live output yet. However, imagine running `ls` in an empty directory. The output would simply be nothing, or if there are files, it would list them:

```
# Example output of 'ls' in a directory with files 'file1.txt' and 'folder_a'
file1.txt  folder_a
```

Practice task

Even without Termux installed, you can start thinking about how you'd organize your files. Imagine you're setting up your Termux environment. What kind of folders would you create? Perhaps a `projects` folder, a `scripts` folder, or a `downloads` folder? Plan out a simple directory structure you'd like to have.

Chapter summary

In this introductory chapter, we've explored what Termux is and why it's a powerful tool for bringing Linux to your Android device. We've touched upon its core features like being a terminal emulator and a Linux environment, its package manager, and the fact that it doesn't require rooting. We also previewed some basic Linux commands that will be essential for navigating your Termux environment. Get ready to dive deeper into the installation process in the next chapter!

Chapter 2: How to Install Termux on Android (F-Droid method)

Introduction

Now that you have a basic understanding of what Termux is, it's time to install it on your Android device. It's important to note that the version of Termux available on the Google Play Store is outdated and no longer maintained. To get the latest version with all the new features and security updates, we will be using F-Droid, an alternative app store for free and open-source software.

This chapter will guide you through the process of installing F-Droid and then using it to install Termux. This is the recommended and official way to get a working and up-to-date Termux environment on your Android device.

Step-by-step guide

Follow these steps carefully to install Termux on your Android device:

1. Enable Installation from Unknown Sources:

- Before you can install F-Droid, you need to allow your device to install apps from sources other than the Google Play Store.
- Go to your Android **Settings** > **Security** (or **Privacy** on some devices).
- Find the option "**Install unknown apps**" or "**Allow installation from unknown sources**" and enable it for your web browser (e.g., Chrome, Firefox).

2. Download and Install F-Droid:

- Open your web browser and go to the official F-Droid website: <https://f-droid.org>
- Tap the "**Download F-Droid**" button to download the F-Droid APK file.
- Once the download is complete, open the downloaded file from your notification bar or file manager and tap "**Install**".

3. Update F-Droid Repositories:

- Open the F-Droid app.
- It will take a few moments to update its repositories (the list of available apps). You can check the progress in the notification bar.

4. Install Termux from F-Droid:

- Once the repositories are updated, tap the search icon in F-Droid.
- Type "**Termux**" in the search bar.
- Select the "Termux" app from the search results.
- Tap the "**Install**" button to download and install Termux.

5. Install Termux:API (Optional but Recommended):

- While you are in F-Droid, it is highly recommended to also install the "**Termux:API**" add-on. This will allow you to access your device's hardware features like the camera, battery, and SMS from within Termux. We will cover this in a later chapter.

Important commands in code blocks

After installing Termux, the first thing you should do is update its package lists and upgrade any installed packages. This ensures you have the latest versions of all the core utilities.

```
# Update the package lists
pkg update

# Upgrade installed packages
pkg upgrade
```

Output examples if possible

When you run `pkg update`, you will see output similar to this:

```
Hit:1 https://termux.librehat.com/apt/termux-main stable InRelease
Ign:2 https://dl.bintray.com/termux/termux-packages-24 stable InRelease
Get:3 https://dl.bintray.com/termux/termux-packages-24 stable Release [14.0 kB]
Get:4 https://dl.bintray.com/termux/termux-packages-24 stable Release.gpg [821 B]
Reading package lists... Done
Building dependency tree...
Reading state information... Done
All packages are up to date.
```


And when you run `pkg upgrade`, if there are packages to upgrade, you will be prompted to confirm the upgrade. You should type `y` and press Enter.

Practice task

Your task for this chapter is to successfully install Termux and the Termux:API add-on from F-Droid. Once you have installed Termux, open the app and run the `pkg update` and `pkg upgrade` commands to get your environment ready for the next chapters.

Chapter summary

In this chapter, you learned how to install the latest version of Termux from F-Droid, the recommended source for the app. You also learned the importance of keeping your packages updated with `pkg update` and `pkg upgrade`. With Termux now installed on your Android device, you are ready to start exploring the Linux command line in the next chapter.

 **Screenshot Recommendation:** A screenshot of the Termux app page in the F-Droid store would be helpful for users to identify the correct app.

Chapter 3: Understanding the Termux File System

Introduction

One of the most crucial aspects of working with any Linux-like environment, including Termux, is understanding its file system. The file system is essentially how your operating system organizes and stores files. In Termux, this is particularly important because it operates within your Android device, but it has its own dedicated space and structure, separate from the main Android file system.

This chapter will guide you through the Termux file system hierarchy, explain where important directories are located, and how they relate to your Android device's storage. A solid grasp of this concept will prevent confusion and help you navigate your Termux environment efficiently.

Step-by-step guide

Termux creates its own isolated environment, which means it has its own root directory (`/`) that is distinct from the Android system's root. The primary location for Termux's files is typically within your Android device's internal storage, but it's sandboxed.

You won't directly see a `/bin` or `/etc` folder of Termux when browsing your phone's internal storage with a file manager, as these are part of its internal, isolated structure.

Key Directories in Termux:

1. `/data/data/com.termux/files/home` (or `~`): This is your **home directory**. When you open Termux, you are automatically placed here. It's your personal workspace where you'll create files, store scripts, and manage your projects. The `~` (tilde) symbol is a common shortcut for the home directory in Linux.
2. `/data/data/com.termux/files/usr`: This directory contains the core Termux system files, including:
 - `bin`: Executable programs and commands (e.g., `ls`, `cd`, `pkg`).
 - `etc`: System-wide configuration files.
 - `tmp`: Temporary files.
 - `var`: Variable data, including package manager data.
3. `/sdcard` or `/storage/emulated/0`: This refers to your Android device's **internal storage**. By default, Termux does not have direct access to this location for security reasons. You will learn how to grant access and link these folders in a later chapter (Chapter 6: Storage Access & Linking Folders).

Understanding this separation is key. You'll primarily work within your home directory (`~`) and use commands to interact with files and programs located in `/data/data/com.termux/files/usr`.

Important commands in code blocks

Let's use some basic commands to explore the Termux file system. Remember, you'll be starting in your home directory (`~`).

```
# Print the current working directory (should be /data/data/com.termux/files/home)
pwd

# List the contents of your home directory
ls

# Change directory to the root of the Termux file system
cd /

# List the contents of the Termux root directory
ls

# Change directory back to your home directory (using the tilde shortcut)
cd ~

# Change directory to the 'usr' directory
cd ../usr

# List the contents of the 'usr' directory
ls

# Go back to the previous directory (your home directory in this case)
cd -
```

Output examples if possible

Here's what you might see when executing some of the commands:

```
# Output of 'pwd' when you first open Termux
/data/data/com.termux/files/home

# Output of 'ls' after 'cd /'
cache data dev home proc root sys tmp usr var

# Output of 'ls' after 'cd ../usr'
bin etc include lib libexec local man share tmp var
```

Practice task

Open your Termux app and try navigating through the file system using the `cd` and `ls` commands. Try to:

1. Go to your home directory (`~`).
2. Create a new directory called `my_projects` inside your home directory.
3. Navigate into `my_projects`.
4. Create another directory called `first_script` inside `my_projects`.
5. Navigate back to your home directory.
6. Verify that `my_projects` exists by listing the contents of your home directory.

Chapter summary

In this chapter, you've gained a fundamental understanding of the Termux file system. You now know that Termux operates in an isolated environment within your Android device, with its own root (`/`), and that your primary workspace is the home directory (`~`). You've also practiced navigating through these directories using essential commands like `pwd`, `ls`, and `cd`. This knowledge is crucial for effectively managing your files and projects within Termux. In the next chapter, we'll delve into more basic Linux commands that will further enhance your command-line proficiency.

Chapter 4: Basic Linux Commands (ls, cd, pwd, mkdir, etc.)

Introduction

In the previous chapter, you got a glimpse of some fundamental Linux commands like `ls`, `cd`, and `pwd` while exploring the Termux file system. These commands are the building blocks of interacting with any Linux environment. Mastering them is essential for navigating, managing files, and generally working efficiently in Termux.

This chapter will dive deeper into these basic commands and introduce a few more crucial ones. We'll cover their various options and provide practical examples to help you become comfortable with the command line. Think of these as your everyday tools for working with your Termux environment.

Step-by-step guide

Let's break down each command, understand its purpose, and explore common ways to use it.

1. `pwd` (Print Working Directory):

- **Purpose:** Shows you the absolute path of your current location in the file system.
- **Usage:** Simply type `pwd` and press Enter.

2. `ls` (List Directory Contents):

- **Purpose:** Lists the files and directories in a specified location. If no location is given, it lists the contents of the current directory.
- **Common Options:**
 - `-l`: Long listing format (shows permissions, owner, size, date, etc.).
 - `-a`: Lists all files, including hidden ones (those starting with a dot `.`).
 - `-h`: Human-readable sizes (e.g., 1K, 234M, 2G).
 - `-F`: Appends a character to entries to indicate their type (e.g., `/` for directories, `*` for executables).

- **Usage:** `ls`, `ls -l`, `ls -a`, `ls -lh`

3. `cd` (Change Directory):

- **Purpose:** Navigates between directories.
- **Common Usage:**
 - `cd <directory_name>`: Go into a specific directory.
 - `cd ..`: Go up one level to the parent directory.
 - `cd /`: Go to the root directory of Termux.
 - `cd ~`: Go to your home directory (most common).
 - `cd -`: Go back to the previous directory you were in.

4. `mkdir` (Make Directory):

- **Purpose:** Creates new directories.
- **Common Options:**
 - `-p`: Creates parent directories as needed. Useful for creating nested directories in one go.
- **Usage:** `mkdir <new_directory_name>`, `mkdir -p <path/to/new/nested/directory>`

5. `rmdir` (Remove Directory):

- **Purpose:** Deletes empty directories. It will not delete directories that contain files.
- **Usage:** `rmdir <directory_name>`

6. `touch` (Create Empty File / Update Timestamp):

- **Purpose:** Creates a new, empty file. If the file already exists, it updates its last modified timestamp.
- **Usage:** `touch <file_name>`

7. `cp` (Copy Files and Directories):

- **Purpose:** Copies files or directories from one location to another.
- **Common Options:**
 - `-r` or `-R`: Required for copying directories (recursive).
- **Usage:** `cp <source_file> <destination_file>`, `cp -r <source_directory> <destination_directory>`

8. `mv` (Move/Rename Files and Directories):

- **Purpose:** Moves files or directories from one location to another, or renames them.
- **Usage:** `mv <source> <destination>` (if destination is a new name in the same directory, it renames; if it's a path, it moves).

9. `rm` (Remove Files and Directories):

- **Purpose:** Deletes files or directories. **Use with extreme caution, as deleted files are often unrecoverable.**
- **Common Options:**
 - `-r`: Required for deleting directories (recursive).
 - `-f`: Force deletion (no prompt for confirmation).
- **Usage:** `rm <file_name>`, `rm -r <directory_name>`, `rm -rf <directory_name>` (very dangerous!)

Important commands in code blocks

Let's put these commands into practice. Assume you are in your home directory (`~`).


```
# Create a new directory for practice
mkdir my_practice

# Change into the new directory
cd my_practice

# Create an empty file
touch my_first_file.txt

# Create a nested directory structure
mkdir -p projects/web/my_site

# List all contents, including hidden files, in long format, with human-readable sizes
ls -lah

# Copy my_first_file.txt to a new file named copied_file.txt
cp my_first_file.txt copied_file.txt

# Move copied_file.txt into the projects directory
mv copied_file.txt projects/

# Rename my_first_file.txt to renamed_file.txt
mv my_first_file.txt renamed_file.txt

# Go back to home directory
cd ~

# Remove the my_practice directory and all its contents (use with caution!)
rm -rf my_practice
```

Output examples if possible

```
# Output after 'ls -lah' in 'my_practice' directory (example)
total 12K
drwxr-xr-x  3 u0_a123 u0_a123 4.0K Jul 12 10:00 .
drwxr-xr-x  3 u0_a123 u0_a123 4.0K Jul 12 09:55 ..
drwxr-xr-x  3 u0_a123 u0_a123 4.0K Jul 12 10:00 projects
-rw-r--r--  1 u0_a123 u0_a123    0 Jul 12 10:00 my_first_file.txt
```

Practice task

Your task is to create a small file and directory structure, manipulate it, and then clean it up. In your home directory:

1. Create a directory named `termux_exercise`.
2. Inside `termux_exercise`, create two files: `notes.txt` and `todo.txt`.
3. Create a subdirectory named `scripts` inside `termux_exercise`.
4. Move `todo.txt` into the `scripts` directory.
5. Rename `notes.txt` to `important_notes.txt`.
6. Copy `important_notes.txt` to `scripts/backup_notes.txt`.
7. List the contents of `termux_exercise` and `termux_exercise/scripts` to verify your changes.
8. Finally, delete the entire `termux_exercise` directory and its contents.

Chapter summary

This chapter has equipped you with the essential basic Linux commands for navigating and managing files and directories within Termux. You've learned how to check your current location (`pwd`), list contents (`ls`), change directories (`cd`), create (`mkdir`), remove (`rmdir`, `rm`), create empty files (`touch`), copy (`cp`), and move/rename (`mv`) files and folders. Remember to always be careful when using `rm -rf`, as it permanently deletes data. With these commands under your belt, you're now much more proficient in controlling your Termux environment. Next, we'll explore how to install new software packages to expand Termux's capabilities.

Chapter 5: Installing Packages & Using apt, pkg, dpkg

Introduction

Termux, by itself, is a powerful terminal emulator, but its true strength lies in its ability to install a vast array of Linux packages. These packages are pre-compiled software applications, tools, and libraries that extend Termux's functionality, allowing you to do everything from programming in various languages to running web servers and ethical hacking tools. Understanding how to install, update, and manage these packages is fundamental to becoming proficient in Termux.

This chapter will introduce you to Termux's package management system, primarily focusing on `pkg`, which is a convenient wrapper around the more traditional `apt` and `dpkg` commands. You'll learn how to search for packages, install them, and keep your Termux environment up-to-date.

Step-by-step guide

Termux uses a package manager similar to those found in Debian-based Linux distributions (like Ubuntu). The primary command you'll use is `pkg`. It simplifies the process of interacting with `apt` (Advanced Package Tool) and `dpkg` (Debian Package management system), which are the underlying tools.

Understanding the Package Managers:

- **pkg** : This is the user-friendly command provided by Termux. It's a wrapper that handles common tasks like installing, updating, and removing packages, often calling `apt` or `dpkg` in the background.
- **apt** : The Advanced Package Tool is a powerful command-line tool for handling packages. It's used for searching, installing, upgrading, and removing packages, as well as managing APT repositories.
- **dpkg** : This is a lower-level tool that directly interacts with `.deb` package files. You typically won't use `dpkg` directly for installing packages from repositories, but it's good to know it exists as the foundation.

Common pkg Commands:

1. **pkg update** : This command updates the list of available packages from the Termux repositories. It's crucial to run this before installing new packages or upgrading existing ones to ensure you're getting the latest information.
2. **pkg upgrade** : After updating the package list, `pkg upgrade` will upgrade all installed packages to their latest versions. It's good practice to run `pkg update` & `pkg upgrade` regularly.
3. **pkg install <package_name>** : This is used to install a new package. Replace `<package_name>` with the actual name of the software you want to install (e.g., `pkg install python`).
4. **pkg search <keyword>** : If you're looking for a specific tool but don't know its exact package name, `pkg search` can help. It will search the repositories for packages whose names or descriptions match your keyword.
5. **pkg show <package_name>** : Displays detailed information about an installed or available package, such as its version, description, and dependencies.
6. **pkg uninstall <package_name>** / **pkg remove <package_name>** : Removes an installed package. `uninstall` and `remove` are generally interchangeable for `pkg`.
7. **pkg autoclean** : Removes downloaded `.deb` package files that are no longer needed. This can free up some space.

Important commands in code blocks

Let's try installing a common package, `neofetch`, which displays system information in a cool ASCII art format.

```
# Always start by updating your package lists
pkg update

# Then upgrade any outdated packages
pkg upgrade

# Search for the 'neofetch' package
pkg search neofetch

# Install neofetch
pkg install neofetch

# Run neofetch to see your system information
neofetch

# Show details about the neofetch package
pkg show neofetch

# Uninstall neofetch (if you want to remove it later)
pkg uninstall neofetch
```

Output examples if possible

When installing a package, you'll see progress indicators and a prompt to confirm the installation. Type `y` and press Enter.

```
# Example output during pkg install neofetch
The following additional packages will be installed:
  libandroid-support ncurses-utils
The following NEW packages will be installed:
  libandroid-support neofetch ncurses-utils
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 102 kB of archives.
After this operation, 423 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://termux.librehat.com/apt/termux-main stable/main aarch64 libandroid-support all 28-1 [21.5 kB]
Get:2 https://termux.librehat.com/apt/termux-main stable/main aarch64 ncurses-utils aarch64 6.2.20200212-1 [45.7 kB]
Get:3 https://termux.librehat.com/apt/termux-main stable/main aarch64 neofetch aarch64 7.1.0-1 [34.8 kB]
Fetched 102 kB in 1s (102 kB/s)
Selecting previously unselected package libandroid-support.
(Reading database ... 12345 files and directories currently installed.)
Preparing to unpack .../libandroid-support_28-1_all.deb ...
Unpacking libandroid-support (28-1) ...
Selecting previously unselected package ncurses-utils.
Preparing to unpack .../ncurses-utils_6.2.20200212-1_aarch64.deb ...
Unpacking ncurses-utils (6.2.20200212-1) ...
Selecting previously unselected package neofetch.
Preparing to unpack .../neofetch_7.1.0-1_aarch64.deb ...
Unpacking neofetch (7.1.0-1) ...
Setting up libandroid-support (28-1) ...
Setting up ncurses-utils (6.2.20200212-1) ...
Setting up neofetch (7.1.0-1) ...
```

Practice task

Your task is to install a text editor called `nano` and then use it to create a simple file.

1. Run `pkg update` and `pkg upgrade` to ensure your system is ready.
2. Search for the `nano` package.
3. Install `nano`.
4. Once installed, type `nano my_first_file.txt` and press Enter. This will open the nano text editor.
5. Type some text into the editor (e.g., "Hello from Termux!").
6. To save and exit, press `Ctrl + o` (write out), then `Enter` to confirm the filename, and then `Ctrl + x` (exit).
7. Verify the file was created and contains your text using `cat my_first_file.txt`.
8. Finally, uninstall `nano`.

Chapter summary

In this chapter, you've learned the essential skill of managing packages in Termux using the `pkg` command. You now know how to update your package lists, upgrade installed software, search for new tools, and install or remove them. This knowledge empowers you to customize your Termux environment with a vast array of Linux applications, opening up endless possibilities for development, scripting, and exploration. In the next chapter, we'll tackle how to access and link your Android device's external storage with Termux, allowing you to work with files outside of Termux's isolated environment.

Chapter 6: Storage Access & Linking Folders

Introduction

By default, Termux operates in its own isolated file system, separate from your Android device's internal storage. This isolation is a security feature, but it also means that files you download or create outside of Termux (e.g., photos, documents, music) are not directly accessible from within Termux, and vice-versa. To truly unlock Termux's potential, you'll need to bridge this gap and allow Termux to read from and write to your device's shared storage.

This chapter will guide you through the process of granting Termux storage permissions and linking your Android's internal storage directories to your Termux home directory. This will enable seamless interaction between your Termux projects and your device's files, making it much easier to manage your work.

Step-by-step guide

To access your Android device's storage from Termux, you need to perform two main steps: grant storage permissions to Termux and then set up symbolic links to your storage folders.

1. Grant Storage Permissions to Termux:

- Open your Android **Settings**.
- Go to **Apps & notifications** (or similar, depending on your Android version).
- Find and tap on **Termux** in the list of installed apps.
- Tap on **Permissions**.
- Locate **Storage** and ensure it is **enabled** or **allowed**. If it's not, tap on it and select "Allow" or "Allow all the time."

Note: Without this permission, Termux cannot access your device's shared storage, and the next step will not work.

2. Run `termux-setup-storage` :

- Open the Termux app.
- At the Termux prompt, type the following command and press Enter:

```
bash termux-setup-storage
```
- When you run this command for the first time, Android will pop up a permission request dialog asking you to grant Termux access to your device's storage. **You MUST accept this permission request.** If you deny it, the command will fail, and you'll need to grant the permission manually via Android settings as described in step 1, then run `termux-setup-storage` again.
- Upon successful execution, `termux-setup-storage` creates a directory named `storage` in your Termux home directory (`~`). Inside this `storage` directory, you will find symbolic links to various common Android storage locations:
 - `~/storage/dcim` (for camera photos and videos)
 - `~/storage/downloads` (for downloaded files)

- `~/storage/external-1` (for external SD card, if present)
- `~/storage/movies`
- `~/storage/music`
- `~/storage/pictures`
- `~/storage/shared` (this is usually your main internal storage, often `/sdcard` or `/storage/emulated/0`)

These are symbolic links, meaning they are shortcuts that point to the actual folders on your Android device. When you access `~/storage/shared` in Termux, you are directly interacting with your phone's internal storage.

Important commands in code blocks

```
# Run this command in Termux to set up storage access
termux-setup-storage

# After running the above, navigate to the storage directory
cd ~/storage

# List the contents of the storage directory to see the linked folders
ls -F

# Navigate to your shared internal storage
cd shared

# List the contents of your phone's internal storage
ls

# Go back to your Termux home directory
cd ~
```

Output examples if possible

After running `termux-setup-storage` and then `ls -F ~/storage`, you might see output similar to this:

```
# Output of ls -F ~/storage
dcim/  downloads/  external-1/  movies/  music/  pictures/  shared/
```

And if you `cd ~/storage/shared` and then `ls`, you will see the contents of your phone's internal storage:

```
# Output of ls ~/storage/shared (example)
Android/  DCIM/  Download/  Movies/  Music/  Notifications/  Pictures/  Podcasts/  Ringtones/  WhatsApp/
```

Practice task

Your task is to successfully link your Termux environment to your Android device's storage and then verify access.

1. Ensure Termux has storage permissions enabled in your Android settings.
2. Open Termux and run `termux-setup-storage`.
3. Navigate to `~/storage/shared`.
4. Create a new text file named `termux_test.txt` in this `shared` directory using `touch termux_test.txt`.
5. Minimize Termux and use your Android device's file manager (e.g., Files by Google, My Files) to navigate to your internal storage. Verify that `termux_test.txt` exists there.
6. Go back to Termux, navigate to `~/storage/shared`, and delete `termux_test.txt` using `rm termux_test.txt`.
7. Verify its deletion using your Android file manager.

Chapter summary

In this chapter, you've learned how to grant Termux the necessary permissions to access your Android device's internal storage and how to use the `termux-setup-storage` command to create convenient symbolic links. This crucial step breaks down the barrier between Termux's isolated environment and your phone's shared files, allowing you to seamlessly work with documents, media, and other data stored on your device. With storage access configured, you're now ready to customize your Termux environment and make it truly your own, which we'll explore in the next chapter.

Chapter 7: Customizing Termux with Themes, Fonts, and Prompt

Introduction

While Termux is incredibly powerful out-of-the-box, its default appearance can be a bit plain. Customizing your Termux environment with themes, fonts, and a personalized prompt not only makes it more visually appealing but also enhances your productivity and makes your command-line experience more enjoyable. A well-configured terminal can reduce eye strain and provide useful information at a glance.

This chapter will guide you through the process of transforming your Termux terminal from its default look to something that suits your style and needs. We'll cover how to change colors, fonts, and modify the command prompt to display relevant information.

Step-by-step guide

Customizing Termux involves installing a few packages and editing some configuration files. We'll primarily use `termux-api` (which you should have installed in Chapter 2) and a package called `termux-tools`.

1. **Install `termux-tools` and `termux-api` (if not already installed):** These packages provide utilities for customization.

```
bash pkg install termux-tools termux-api
```

2. **Changing the Color Scheme and Font:** Termux has a built-in feature to change themes and fonts. This is done by long-pressing anywhere on the Termux screen to bring up the context menu.

- **Long-press** on the Termux terminal screen.
- A menu will appear. Tap on **"More..."**.
- Tap on **"Style"**.
- You will see options for **"Color preset"** and **"Font"**.
- Tap on **"Color preset"** to choose from various pre-defined color schemes (e.g., Solarized Light, Dracula, Monokai).
- Tap on **"Font"** to select a different font. Termux comes with a few pre-installed fonts, but you can also add your own (advanced topic, usually involves placing `.ttf` files in `~/.termux/fonts/`).

Experiment with different combinations to find what you like best!

3. **Customizing the Prompt (PS1):** The command prompt is the text you see before you type a command (e.g., `$`` or `$`). You can customize this to display useful information like your current directory, username, hostname, or even the current time. This is done by modifying the `PS1` environment variable, typically in your shell's configuration file (e.g., `.bashrc` for Bash).

- First, open your `.bashrc` file for editing. We'll use `nano` (which you installed in Chapter 5).

```
bash nano ~/.bashrc
```

- If the file is empty or doesn't exist, you can create it. Add or modify the `PS1` line. Here are some examples:

- **Simple prompt (username@hostname:current_directory\$)** `bash PS1='\u@\h:\w\$ '`
- **Prompt with date and time:** `bash PS1='\D{%H:%M:%S} \u@\h:\w\$ '`
- **Colorful prompt (example - green text, blue directory):** `bash PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '` *Explanation of color codes:* `\[\033[01;32m\]` starts bold green, `\[\033[00m\]` resets color.

- After editing, save the file (`Ctrl + o` , Enter) and exit (`Ctrl + x`).
- To apply the changes without restarting Termux, `source` the file:

```
bash source ~/.bashrc
```

Important commands in code blocks

```
# Install necessary tools for customization
pkg install termux-tools termux-api

# Open .bashrc to edit your prompt
nano ~/.bashrc

# Example PS1 variable for a custom prompt
# PS1='\u@\h:\w\$ '

# Apply changes to .bashrc without restarting Termux
source ~/.bashrc
```

Output examples if possible

After changing your `PS1` variable and sourcing `.bashrc`, your prompt will immediately change. For example, if you set `PS1='\u@\h:\w\$ '`, your prompt might look like this:

```
u0_a123@localhost:~$
```

If you set a colorful prompt, the text will appear in the colors you specified.

Practice task

Your task is to customize your Termux environment to your liking.

1. Install `termux-tools` and `termux-api` if you haven't already.
2. Experiment with different color presets and fonts using the long-press menu in Termux.
3. Edit your `~/.bashrc` file and set a custom `PS1` variable. Try to include your username, hostname, and current working directory. You can also try adding colors if you feel adventurous.
4. Source your `.bashrc` file to see the changes immediately.
5. (Optional) Search online for more `PS1` examples and try implementing one that you find interesting.

Chapter summary

In this chapter, you've learned how to personalize your Termux environment by changing its color scheme, font, and customizing your command prompt. These aesthetic and functional improvements can significantly enhance your command-line experience, making it more pleasant and efficient. By mastering these customization techniques, you're taking another step towards making Termux truly your own. In the next chapter, we'll dive into the exciting world of shell scripting, where you'll learn to automate tasks and create your own powerful command-line programs.

Chapter 8: Shell Scripting Basics in Termux

Introduction

One of the most powerful features of any Linux-like environment, including Termux, is the ability to write shell scripts. A shell script is simply a text file containing a series of commands that the shell (your command-line interpreter, usually Bash in Termux) can execute. This allows you to automate repetitive tasks, combine multiple commands into a single executable file, and create your own custom tools.

This chapter will introduce you to the fundamentals of shell scripting in Termux. You'll learn how to create your first script, make it executable, understand basic scripting concepts like variables and conditional statements, and run it. Mastering shell scripting is a significant step towards becoming a more efficient and powerful Termux user.

Step-by-step guide

Let's break down the process of creating and running a simple shell script.

1. **Choose a Text Editor:** You'll need a text editor to write your scripts. `nano` (which you installed in Chapter 5) is a good choice for beginners. You can also use `vim` or `emacs` if you're familiar with them.
2. **Create a New File:** All shell scripts are plain text files. It's a common convention to give them a `.sh` extension, though it's not strictly required.

```
bash nano my_first_script.sh
```

3. **Add the Shebang Line:** The very first line of any shell script should be the 'shebang' (short for 'hash-bang'). This line tells the system which interpreter to use to execute the script. For Bash scripts, it's `#!/bin/bash`.

```
`` `bash
```

#!/bin/bash

This is a comment. Comments start with # and are ignored by the interpreter.

```
echo "Hello from my first Termux script!" `` `
```

4. **Make the Script Executable:** By default, newly created files are not executable. You need to give your script execute permissions using the `chmod` command.

```
bash chmod +x my_first_script.sh
```

5. **Run the Script:** Once the script is executable, you can run it by preceding its name with `./` (which means 'in the current directory').

```
bash ./my_first_script.sh
```

Basic Scripting Concepts:

- **Variables:** You can store data in variables. Variable names are case-sensitive. `bash my_name="Mahedi" echo "My name is $my_name"`

- **User Input:** Use the `read` command to get input from the user. `bash echo "What is your favorite color?" read fav_color echo "Your favorite color is $fav_color"`
- **Conditional Statements (if/else):** Execute different commands based on conditions. ```bash #!/bin/bash echo "Enter a number:" read num`
`if [$num -gt 10]; then echo "The number is greater than 10." else echo "The number is 10 or less." fi ``` *Note: Spaces around `[` and `]` are crucial. `* -gt`` means 'greater than'.
- **Loops (for, while):** Repeat a block of commands. `bash #!/bin/bash for i in 1 2 3 4 5; do echo "Counting: $i" done`

Important commands in code blocks

```
# Create a new script file
nano my_script.sh

# Make the script executable
chmod +x my_script.sh

# Run the script
./my_script.sh

# Example of a script with variables and input
#!/bin/bash
name="Termux User"
echo "Hello, $name!"
echo "What is your age?"
read age
echo "You are $age years old."
```

Output examples if possible

Running `my_first_script.sh`:

```
Hello from my first Termux script!
```

Running the variable and input script:

```
Hello, Termux User!
What is your age?
25
You are 25 years old.
```

Practice task

Create a script named `welcome.sh` that does the following:

1. Asks the user for their name.
2. Greets the user by name.
3. Asks the user for their favorite Termux command.
4. Prints a message acknowledging their favorite command.
5. Make the script executable and run it.

Chapter summary

In this chapter, you've taken your first steps into the world of shell scripting in Termux. You've learned how to create, make executable, and run basic scripts. You've also been introduced to fundamental scripting concepts such as variables, user input, conditional statements, and loops. With these skills, you can start automating tasks and building custom tools to streamline your

workflow in Termux. In the next chapter, we'll explore Git and GitHub, essential tools for version control and collaboration, which are invaluable for any developer or scripter.

Chapter 9: Git & GitHub Usage (clone, commit, push)

Introduction

As you start creating more scripts and projects in Termux, managing different versions of your code and collaborating with others becomes essential. This is where Git and GitHub come in. Git is a powerful version control system that tracks changes in your files, allowing you to revert to previous versions, work on different features simultaneously, and merge changes seamlessly. GitHub is a web-based platform that uses Git for version control, providing a central place to host your repositories, share your code, and collaborate with a global community of developers.

This chapter will guide you through installing Git in Termux and using it to manage your local projects. You'll also learn the basics of interacting with GitHub, including cloning repositories, making changes, committing them, and pushing them to your remote repository.

Step-by-step guide

To use Git and GitHub in Termux, you'll first need to install Git and then configure it. You'll also need a GitHub account (if you don't have one already).

1. **Install Git:** Git is available as a package in Termux. Use `pkg` to install it.

```
bash pkg install git
```

2. **Configure Git:** After installation, you need to tell Git who you are. This information will be attached to your commits.

```
bash git config --global user.name "Your Name" git config --global user.email "your_email@example.com"
```

Replace "Your Name" and "your_email@example.com" with your actual name and email. --global means these settings apply to all your Git repositories.

3. **Create a GitHub Account (if you don't have one):** Go to <https://github.com/> and sign up for a free account. You'll need this to host your projects online.

4. **Clone a Repository:** To start working on an existing project from GitHub, you can clone its repository. This downloads a copy of the project to your Termux environment.

```
bash git clone https://github.com/username/repository_name.git
```

Replace username and repository_name with the actual GitHub username and repository name. You can find the clone URL on the GitHub repository page.

5. **Make Changes and Stage Them:** Navigate into the cloned repository directory. Make some changes to a file or create a new one using `nano` or another editor.

After making changes, you need to tell Git to track these changes. This is called 'staging'.

```
bash git add <file_name> # To stage a specific file git add . # To stage all changes in the current directory
```

6. **Commit Changes:** Once changes are staged, you 'commit' them. A commit is a snapshot of your repository at a specific point in time, along with a message describing the changes.

```
bash git commit -m "Your descriptive commit message here"
```

The -m flag allows you to provide a commit message directly. Good commit messages are concise and explain what changes were made. If you omit -m, Git will open a text editor (like nano) for you to write the message.

7. Push Changes to GitHub: After committing, your changes are only on your local machine. To upload them to your GitHub repository, you 'push' them.

`bash git push origin main` *origin refers to the remote repository you cloned from, and main (or master) is the branch you are pushing to. You might be prompted for your GitHub username and password/Personal Access Token (PAT) the first time you push.* For security, GitHub recommends using PATs instead of passwords for command-line operations.

Important commands in code blocks

```
# Install Git
pkg install git

# Configure your Git username and email
git config --global user.name "Your Name"
git config --global user.email "your_email@example.com"

# Clone a repository from GitHub
git clone https://github.com/username/repository_name.git

# Check the status of your repository (shows modified, staged, untracked files)
git status

# Stage changes (add files to the staging area)
git add .

# Commit staged changes with a message
git commit -m "Added new feature"

# Push committed changes to GitHub
git push origin main
```

Output examples if possible

git status output after modifying a file:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

git commit output:

```
[main 7c9a0b1] Added new feature
1 file changed, 2 insertions(+), 1 deletion(-)
```

git push output:

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 312 bytes | 312.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/username/repository_name.git
 a1b2c3d..7c9a0b1  main -> main
```

Practice task

Your task is to create a new repository on GitHub, clone it to Termux, make some changes, and push them back.

1. Go to GitHub.com and create a **new public repository** (e.g., `my-termux-scripts`). Initialize it with a `README.md` file.
2. In Termux, clone your newly created repository: `bash git clone https://github.com/YOUR_USERNAME/my-termux-scripts.git` Remember to replace `YOUR_USERNAME`.

3. Navigate into the `my-termux-scripts` directory.
4. Create a new file named `hello.sh` with the following content: `bash #!/bin/bash echo "Hello from my Termux Git repo!"`
5. Make `hello.sh` executable: `chmod +x hello.sh`.
6. Stage the new file: `git add hello.sh`.
7. Commit your changes: `git commit -m "Add hello.sh script"`.
8. Push your changes to GitHub: `git push origin main`.
9. Verify on GitHub.com that `hello.sh` is now in your repository.

Chapter summary

In this chapter, you've learned the essentials of using Git for version control and interacting with GitHub. You've installed Git, configured your user information, cloned a remote repository, made changes, staged them with `git add`, committed them with `git commit`, and pushed them to GitHub with `git push`. These are fundamental skills for any developer, enabling you to track your code, collaborate with others, and manage your projects effectively. In the next chapter, you'll apply some of this knowledge to create your own command-line interface (CLI) tools.

Chapter 10: Create Your Own CLI Tools

Introduction

In the previous chapters, you've learned how to navigate the Termux file system, use basic Linux commands, install packages, and even write simple shell scripts. Now, it's time to combine these skills to create your own Command Line Interface (CLI) tools. CLI tools are programs that you interact with directly from the terminal by typing commands. They are incredibly powerful for automating tasks, processing data, and extending the functionality of your Termux environment.

This chapter will guide you through the process of designing, writing, and making your own CLI tools accessible from anywhere in your Termux terminal. We'll focus on using shell scripting (Bash) for simplicity, but the principles apply to tools written in other languages like Python or Node.js.

Step-by-step guide

Creating a CLI tool involves writing a script and then placing it in a location that's part of your system's `PATH`.

1. **Design Your Tool:** Before writing code, think about what your tool should do. What problem does it solve? What input does it need? What output should it produce? For this chapter, let's create a simple tool that greets the user and tells them the current date and time.
2. **Write the Script:** Use your preferred text editor (like `nano`) to create a new script file. It's good practice to give your tool a descriptive name, without a `.sh` extension if you want it to behave like a standard command.

```
bash nano mygreet
```

Add the following content to the `mygreet` file:

```
```bash
```

# #!/bin/bash

---

## This is my first custom CLI tool

---

```
echo "Hello, Termux user!" echo "Today is $(date +%Y-%m-%d) and the time is $(date +%H:%M:%S)." ``
```

\*Explanation: \* \$(date +%Y-%m-%d) executes the date command with a specific format and inserts its output into the echo` command.

**3. Make the Script Executable:** Just like with any shell script, you need to give your tool execute permissions.

```
bash chmod +x mygreet
```

**4. Place the Tool in Your PATH :** For your tool to be accessible from any directory, it needs to be in a directory that is included in your system's PATH environment variable. In Termux, a common place for user-created executables is ~/bin .

- First, create the bin directory in your home folder if it doesn't exist: `bash mkdir -p ~/bin`
- Then, move your mygreet script into this ~/bin directory: `bash mv mygreet ~/bin/`

*Termux automatically adds ~/bin to your PATH when you start a new session, so any executable scripts placed there will be recognized as commands.*

**5. Test Your Tool:** Now, you should be able to run your tool from any directory.

```
bash mygreet
```

### Adding Arguments to Your Tool:

Most CLI tools accept arguments (additional information passed to the command). You can access these arguments in your script using \$1, ``\$2, etc., where \$1 is the first argument, ``\$2 is the second, and so on. \$# gives the number of arguments, and ``\$@ refers to all arguments.

Let's modify mygreet to accept a name as an argument:

```
Edit the tool
nano ~/bin/mygreet
```

Change the content to:

```
#!/bin/bash
This is my first custom CLI tool with an argument

if [-z "$1"]; then
 echo "Usage: mygreet <your_name>"
 echo "Please provide your name as an argument."
else
 echo "Hello, $1!"
 echo "Today is $(date +%Y-%m-%d) and the time is $(date +%H:%M:%S)."
fi
```

Now, try running it with an argument:

```
mygreet Mahedi
```

## Important commands in code blocks

---

```
Create a new script file for your tool
nano my_custom_tool

Make the script executable
chmod +x my_custom_tool

Create the ~/bin directory if it doesn't exist
mkdir -p ~/bin

Move your tool into the ~/bin directory
mv my_custom_tool ~/bin/

Run your custom tool
my_custom_tool

Example script content with arguments
#!/bin/bash
if [-z "$1"]; then
 echo "Error: No argument provided."
else
 echo "You provided: $1"
fi
```

## Output examples if possible

---

Output of `mygreet` without arguments (after modification):

```
Usage: mygreet <your_name>
Please provide your name as an argument.
```

Output of `mygreet Mahedi` :

```
Hello, Mahedi!
Today is 2025-07-12 and the time is 10:30:00.
```

## Practice task

---

Create a CLI tool named `mkproject` that automates the creation of a new project directory with a basic structure.

1. The tool should accept one argument: the name of the new project.
2. If no project name is provided, it should print a usage message.
3. If a project name is provided, it should create a new directory with that name.
4. Inside the new project directory, it should create two subdirectories: `src` and `docs`.
5. Inside the `src` directory, it should create an empty file named `main.sh`.
6. Make the `mkproject` script executable and place it in `~/bin`.
7. Test your `mkproject` tool by creating a project (e.g., `mkproject my_new_app`) and verify its structure.

## Chapter summary

---

In this chapter, you've learned how to create your own Command Line Interface (CLI) tools in Termux using shell scripting. You now understand the process of writing a script, making it executable, and placing it in your `PATH` so it can be run from anywhere. You also explored how to make your tools more interactive by accepting arguments. This skill is incredibly valuable for automating your workflow and building personalized utilities that enhance your Termux experience. In the next chapter, we'll shift our focus to a popular programming language: Python, and how to use it effectively within Termux.

---

# Chapter 11: Python Programming in Termux

---

## Introduction

---

Python is one of the most popular and versatile programming languages in the world, known for its readability and extensive libraries. It's widely used for web development, data analysis, artificial intelligence, automation, and much more. The good news is that you can run Python directly on your Android device using Termux, turning your phone into a powerful development environment.

This chapter will guide you through installing Python in Termux, running your first Python script, and understanding how to manage Python packages using `pip`. By the end of this chapter, you'll be ready to start coding in Python on your mobile device.

## Step-by-step guide

---

Installing Python in Termux is straightforward, thanks to the `pkg` package manager.

1. **Install Python:** Termux provides Python 3.x as a package. Use the `pkg install` command to get it.

```
bash pkg install python
```

This command will install Python along with `pip`, the Python package installer, which you'll use to install additional Python libraries.

2. **Verify Installation:** After installation, you can verify that Python and `pip` are correctly installed by checking their versions.

```
bash python --version pip --version
```

3. **Run Your First Python Script:** Let's create a simple

Python script. Use `nano` to create a file named `hello.py`:

```
```bash
nano hello.py
```

Add the following Python code to the file:

```python
# hello.py
print("Hello from Python in Termux!")

name = input("What's your name? ")
print(f"Nice to meet you, {name}!")
```

Save and exit `nano` (`Ctrl + O`, Enter, `Ctrl + X`).

Now, run your Python script:

```bash
python hello.py
```
```

1. **Manage Python Packages with `pip`:** `pip` is the standard package manager for Python. It allows you to install and manage libraries that extend Python's capabilities. For example, let's install the `requests` library, which is commonly used for making HTTP requests.

```
bash pip install requests
```

You can also list installed packages:

```
bash pip list
```

To uninstall a package:

```
bash pip uninstall requests
```

**2. Virtual Environments (Optional but Recommended):** For larger projects, it's good practice to use virtual environments. This creates an isolated environment for your Python project, preventing conflicts between different project dependencies. Termux supports `venv` (built-in with Python 3) or `virtualenv`.

```
```bash
```

Create a virtual environment named 'myenv'

```
python -m venv myenv
```

Activate the virtual environment

```
source myenv/bin/activate
```

You'll see (myenv) before your prompt, indicating it's active

Now, any packages you install with pip will only be in this environment

```
pip install some-package
```

Deactivate the virtual environment

```
deactivate ```
```

Important commands in code blocks

```
# Install Python
pkg install python

# Run a Python script
python your_script.py

# Install a Python package using pip
pip install package_name

# List installed Python packages
pip list

# Uninstall a Python package
pip uninstall package_name

# Create a virtual environment
python -m venv my_project_env

# Activate a virtual environment
source my_project_env/bin/activate

# Deactivate a virtual environment
deactivate
```


Output examples if possible

Output of `python hello.py` :

```
Hello from Python in Termux!  
What's your name? Mahedi  
Nice to meet you, Mahedi!
```

Output of `pip install requests` :

```
Collecting requests  
  Downloading requests-2.31.0-py3-none-any.whl (62 kB)  
      _____ 62.6/62.6 kB 1.3 MB/s eta 0:00:00  
Collecting charset-normalizer<4,>=2  
  Downloading charset_normalizer-3.3.2-cp311-cp311-linux_aarch64.whl (120 kB)  
      _____ 120.6/120.6 kB 2.3 MB/s eta 0:00:00  
... (more output)  
Successfully installed certifi-2023.11.17 charset-normalizer-3.3.2 idna-3.6 requests-2.31.0 urllib3-2.1.0
```

Practice task

Your task is to write a Python script that calculates the area of a rectangle.

1. Create a new Python file named `rectangle_area.py` .
2. Inside the script, ask the user to input the length and width of the rectangle.
3. Calculate the area (length * width).
4. Print the calculated area to the console.
5. Run your script and test it with different values.
6. (Optional) Install the `math` module (though not strictly needed for this task, it's good practice to try installing a module) and then uninstall it.

Chapter summary

This chapter has shown you how to set up and start programming with Python in Termux. You've learned to install Python, run basic scripts, and manage external libraries using `pip` . You also got an introduction to virtual environments, which are crucial for managing project dependencies. With Python installed and configured, your Termux environment is now even more powerful, ready for you to develop a wide range of applications and scripts. In the next chapter, we'll explore another popular programming language: Node.js, and how to use it for JavaScript projects.

Chapter 12: Node.js & JavaScript Projects

Introduction

Node.js is an open-source, cross-platform JavaScript runtime environment that allows you to execute JavaScript code outside of a web browser. It's widely used for building scalable network applications, web servers, APIs, and command-line tools. If you're familiar with JavaScript or want to learn it for backend development, Node.js in Termux provides an excellent mobile-first platform.

This chapter will guide you through installing Node.js and npm (Node Package Manager) in Termux, running your first Node.js script, and managing Node.js packages. You'll be able to develop and run JavaScript applications directly from your Android device.

Step-by-step guide

Installing Node.js in Termux is straightforward using the `pkg` package manager.

1. **Install Node.js:** The `nodejs` package in Termux includes both Node.js and npm.

```
bash pkg install nodejs
```

2. **Verify Installation:** After installation, check the versions of Node.js and npm to ensure they are correctly installed.

```
bash node -v npm -v
```

3. **Run Your First Node.js Script:** Let's create a simple JavaScript file. Use `nano` to create `app.js`:

```
bash nano app.js
```

Add the following JavaScript code to the file:

```
```javascript // app.js console.log("Hello from Node.js in Termux!");
```

```
const readline = require("readline").createInterface({ input: process.stdin, output: process.stdout });
```

```
readline.question(What's your favorite Node.js module?, (moduleName) => { console.log(You like the
${moduleName} module!); readline.close(); }); ```
```

Save and exit nano (Ctrl + O, Enter, Ctrl + X).

Now, run your Node.js script:

```
bash node app.js
```

4. **Manage Node.js Packages with npm:** npm (Node Package Manager) is the default package manager for Node.js. It allows you to install, share, and manage packages (libraries and tools) for your Node.js projects.

Let's initialize a new Node.js project and install a popular package like `lodash`.

```
```bash
```

Create a new project directory

```
mkdir my_node_project cd my_node_project
```

Initialize a new npm project (answer the prompts or use -y for defaults)

```
npm init -y
```

Install a package (e.g., lodash)

```
npm install lodash ```
```

This will create a `node_modules` directory (where packages are stored) and update `package.json` and `package-lock.json`.

To use the installed package in your `app.js` (assuming `app.js` is in `my_node_project`):

```
```javascript // app.js const _ = require("lodash");
```

```
const numbers = [1, 2, 3, 4, 5]; console.log(_.reverse(numbers)); // Output: [5, 4, 3, 2, 1] ``
```

To uninstall a package:

```
bash npm uninstall lodash
```

## Important commands in code blocks

---

```
Install Node.js and npm
pkg install nodejs

Run a Node.js script
node your_script.js

Initialize a new npm project
npm init -y

Install a Node.js package
npm install package_name

Uninstall a Node.js package
npm uninstall package_name

List globally installed npm packages
npm list -g --depth=0
```

## Output examples if possible

---

**Output of node app.js :**

```
Hello from Node.js in Termux!
What's your favorite Node.js module? Express
You like the Express module!
```

**Output of npm install lodash :**

```
npm WARN deprecated lodash@4.17.21: lodash has been updated to v5.0.0. Please upgrade to the latest version.

added 1 package, and audited 2 packages in 1s

found 0 vulnerabilities
```

## Practice task

---

Your task is to create a simple Node.js script that acts as a basic calculator.

1. Create a new directory named `calculator_app`.
2. Navigate into `calculator_app` and initialize a new npm project.
3. Create a JavaScript file named `calculator.js`.
4. Inside `calculator.js`, write a script that takes two numbers and an operation (add, subtract, multiply, divide) as command-line arguments.
5. Perform the calculation and print the result. *Hint: You can access command-line arguments using `process.argv`.*
6. Run your script with different numbers and operations (e.g., `node calculator.js 10 5 add`).

## Chapter summary

---

In this chapter, you've successfully set up your Termux environment for Node.js and JavaScript development. You've learned how to install Node.js and npm, execute JavaScript files, and manage project dependencies using npm. This opens up a vast ecosystem of JavaScript libraries and frameworks, allowing you to build powerful backend applications, web tools, and more, all from your Android device. In the next chapter, we'll explore how to host simple web servers directly from Termux using various technologies.

---

# Chapter 13: Hosting Servers in Termux (PHP, HTTP, Python)

---

## Introduction

---

One of the most exciting capabilities of Termux is its ability to host simple web servers directly on your Android device. This means you can serve web pages, test web applications, or even share files over your local network without needing a dedicated computer. This is incredibly useful for developers, for quick testing, or for demonstrating projects on the go.

This chapter will show you how to set up and run basic HTTP servers using Python, PHP, and Node.js. You'll learn how to serve static files, and understand the fundamentals of making your device act as a web server.

## Step-by-step guide

---

We'll cover three popular ways to host a simple web server in Termux:

### 1. Python's Built-in HTTP Server

Python comes with a simple HTTP server built-in, perfect for serving static files from a directory. This is often the quickest way to get a web server up and running.

1. **Ensure Python is installed:** (Refer to Chapter 11 if you haven't installed Python yet).

```
bash pkg install python
```

2. **Navigate to your web content directory:** Create a directory for your web files (e.g., `my_website`) and put an `index.html` file inside it.

```
bash mkdir my_website cd my_website echo "<h1>Hello from Termux Web Server!</h1>" > index.html
```

3. **Start the Python HTTP server:**

```
bash python -m http.server 8080
```

 This command starts a server on port 8080. You can choose any available port (e.g., 8000, 5000).

4. **Access the server:** Open your web browser (on the same device or another device on the same Wi-Fi network) and go to `http://localhost:8080` (if on the same device) or `http://YOUR_TERMUX_IP:8080` (if from another device). To find your Termux device's IP address, use `ifconfig` (you might need to `pkg install net-tools`).

*To stop the server, press `Ctrl + C` in the Termux terminal.*

### 2. PHP's Built-in Development Server

If you're working with PHP, its built-in development server is a convenient way to test PHP applications without configuring a full-fledged web server like Apache or Nginx.

1. **Install PHP:**

```
bash pkg install php
```

2. **Navigate to your PHP project directory:** Create a directory and an `index.php` file.

```
bash mkdir my_php_app cd my_php_app echo "<?php echo \"Hello from PHP in Termux!\"; ?>" > index.php
```

3. **Start the PHP development server:**

```
bash php -S 0.0.0.0:8080 0.0.0.0
```

 makes the server accessible from other devices on your network.

4. **Access the server:** Similar to the Python server, open your web browser and go to `http://localhost:8080` or `http://YOUR_TERMUX_IP:8080`.

*To stop the server, press `Ctrl + C`.*

### 3. Node.js HTTP Server (using Express.js for simplicity)

For more robust web applications, Node.js with frameworks like Express.js is a popular choice. While you can create a server with Node.js's built-in `http` module, Express.js simplifies the process.

1. **Ensure Node.js is installed:** (Refer to Chapter 12 if you haven't installed Node.js yet).

```
bash pkg install nodejs
```

2. **Initialize a Node.js project and install Express:**

```
bash mkdir my_node_server cd my_node_server npm init -y npm install express
```

3. **Create your server file (e.g., `server.js`):**

```
bash nano server.js
```

Add the following content:

```
`` `javascript // server.js const express = require("express"); const app = express(); const port = 3000;

app.get("/", (req, res) => { res.send("Hello from Node.js Express server in Termux!"); });

app.listen(port, () => { console.log(Server listening at http://localhost:${port}); }); `` `
```

4. **Start the Node.js server:**

```
bash node server.js
```

5. **Access the server:** Open your web browser and go to `http://localhost:3000` or `http://YOUR_TERMUX_IP:3000`.

*To stop the server, press `Ctrl + C`.*

## Important commands in code blocks

```
Install Python (if not already installed)
pkg install python

Start Python's simple HTTP server
python -m http.server 8080

Install PHP (if not already installed)
pkg install php

Start PHP's built-in development server
php -S 0.0.0.0:8080

Install Node.js (if not already installed)
pkg install nodejs

Install Express.js for Node.js web apps
npm install express

Start a Node.js server (assuming server.js is your app file)
node server.js

Find your Termux device's IP address (install net-tools if needed)
pkg install net-tools
ifconfig
```

## Output examples if possible

Python HTTP server output:

```
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/)
```

#### PHP development server output:

```
PHP 8.2.10 Development Server (http://0.0.0.0:8080) started
```

#### Node.js Express server output:

```
Server listening at http://localhost:3000
```

## Practice task

---

Your task is to set up a simple web page and serve it using the Python HTTP server.

1. Create a new directory named `my_static_site`.
2. Inside `my_static_site`, create an `index.html` file with some basic HTML content (e.g., a heading and a paragraph).
3. Start the Python HTTP server from within the `my_static_site` directory on port 8000.
4. Open your mobile browser and navigate to `http://localhost:8000` to verify that your page is being served.
5. (Optional) If you have another device on the same Wi-Fi network, find your Termux device's IP address and try accessing the page from that device.
6. Stop the server when you are done.

## Chapter summary

---

In this chapter, you've learned how to transform your Termux environment into a basic web server. You've explored how to use Python's built-in HTTP server, PHP's development server, and a simple Node.js Express server to serve static files and dynamic content. This capability is incredibly useful for local development, testing, and sharing. Understanding how to host servers locally is a crucial step for anyone looking to develop web applications or experiment with network services on their Android device. In the next chapter, we'll delve into the world of ethical hacking tools available in Termux, emphasizing their legal and responsible use.

---

# Chapter 14: Ethical Hacking Tools in Termux (legal use only)

---

## Introduction

---

Termux, with its Linux environment, provides access to a wide array of tools that are commonly used in cybersecurity, including those for ethical hacking and penetration testing. It's crucial to understand that while these tools are powerful, they must **only be used for legal and ethical purposes**, such as securing your own systems, learning about vulnerabilities in controlled environments, or with explicit permission from the owner of the system you are testing. Unauthorized access or malicious use of these tools is illegal and can lead to severe consequences.

This chapter will introduce you to some common ethical hacking tools available in Termux and demonstrate their basic, legal usage. We will focus on tools for network scanning, information gathering, and vulnerability analysis, always emphasizing responsible and authorized use.

## Step-by-step guide

---

Before installing any tools, always ensure your Termux environment is updated:

```
pkg update && pkg upgrade
```

## 1. Nmap (Network Mapper)

Nmap is a free and open-source utility for network discovery and security auditing. It's used to discover hosts and services on a computer network, thus creating a "map" of the network.

1. **Install Nmap:** `bash pkg install nmap`

### 2. Basic Usage (Legal and Ethical):

- **Scan your own local network:** You can scan your home Wi-Fi network to see what devices are connected and what ports are open on your own devices. This helps you understand your network's security posture.

```
```bash
```

Find your local IP address (e.g., 192.168.1.x)

```
ifconfig
```

Scan your local network range (replace 192.168.1.0/24 with your network)

`nmap -sn 192.168.1.0/24 ```` * -sn` performs a ping scan, which only checks if hosts are up, without port scanning.*

- **Scan your own device for open ports:**

```
bash nmap localhost nmap 127.0.0.1
```

This scans your Termux environment for open ports, useful if you're running local servers.

- **Never scan networks or devices you do not own or have explicit permission to test.**

2. Wireshark (TShark - Command Line Version)

Wireshark is a popular network protocol analyzer. While the full GUI version isn't available in Termux, its command-line counterpart, TShark, is. TShark allows you to capture and analyze network traffic.

1. **Install TShark:** `bash pkg install tshark`

2. Basic Usage (Legal and Ethical):

- **Capture traffic on your own device:** You can capture traffic originating from or destined for your own Android device. This is useful for debugging network applications you develop or understanding how apps on your phone communicate.

```
```bash
```

## List available network interfaces

---

```
tshark -D
```

## Capture traffic on a specific interface (e.g., wlan0 for Wi-Fi)

---

## Press Ctrl+C to stop capturing

---

```
tshark -i wlan0
```

## Capture and save to a file (useful for later analysis)

---

```
tshark -i wlan0 -w my_capture.pcap `` *Analyzing my_capture.pcap` would typically be done on a desktop computer with the full Wireshark GUI.*
```

- **Only capture traffic on networks you own or have explicit permission to monitor.** Capturing traffic on public Wi-Fi without consent is illegal.

### 3. Metasploit Framework (Advanced - Use with Extreme Caution)

Metasploit is a powerful penetration testing framework that provides a vast collection of exploits and payloads. **Installing and using Metasploit requires significant disk space and can be complex.** Its use is strictly for advanced users and **must only be performed in controlled, isolated lab environments with explicit permission.**

1. **Installation (Not Recommended for Beginners):** The installation process for Metasploit in Termux is lengthy and resource-intensive. It typically involves downloading a script and running it.

```
`` `bash
```

## This is an example, actual installation steps may vary and require more resources

---

```
curl -LO
```

```
https://raw.githubusercontent.com/Hax4us/Metasploit_termux/master/metasploit.sh
```

---

```
chmod +x metasploit.sh
```

---

```
./metasploit.sh
```

---

```
`` `
```

2. **Legal and Ethical Use:**



- **Lab Environment:** Metasploit should only be used against systems you own, or systems for which you have explicit, written permission to test (e.g., a virtual machine on your own computer, a dedicated test server).
- **Learning:** Use it to understand how vulnerabilities are exploited and how to defend against them. Never use it to gain unauthorized access to any system.

**Due to the high risk of misuse and the complexity, we will not provide detailed usage examples for Metasploit in this beginner-focused book. Misuse can lead to severe legal consequences.**

## Important commands in code blocks

```
Update Termux packages
pkg update && pkg upgrade

Install Nmap
pkg install nmap

Scan local network for active hosts (ethical use only)
nmap -sn 192.168.1.0/24

Install TShark
pkg install tshark

List network interfaces
tshark -D

Capture traffic on a specific interface (ethical use only)
tshark -i wlan0
```

## Output examples if possible

### Nmap scan output (example):

```
Starting Nmap 7.92 (https://nmap.org) at 2025-07-12 11:00 EDT
Nmap scan report for 192.168.1.1
Host is up (0.0030s latency).
Nmap scan report for 192.168.1.100
Host is up (0.0050s latency).
Nmap done: 256 IP addresses (2 hosts up) scanned in 2.56 seconds
```

### TShark interface list output (example):

```
1. wlan0 (IEEE 802.11 Wireless LAN)
2. eth0 (Ethernet)
3. any (Pseudo-device that captures on all interfaces)
4. lo (Loopback)
```

## Practice task

Your task is to legally and ethically explore your own network using Nmap.

1. Ensure Nmap is installed.
2. Find your Android device's local IP address using `ifconfig`.
3. Identify your local network range (e.g., if your IP is `192.168.1.105`, your network range is likely `192.168.1.0/24`).
4. Perform a ping scan ( `nmap -sn` ) of your *entire local network range* to discover active devices. Note down the IP addresses of devices you recognize (e.g., your router, other phones, smart devices).
5. Perform a port scan ( `nmap localhost` ) on your own Termux environment to see if any ports are open (e.g., if you ran a web server in the previous chapter).

## Chapter summary

---

This chapter introduced you to some powerful ethical hacking tools available in Termux, such as Nmap for network scanning and TShark for network traffic analysis. We emphasized the critical importance of using these tools legally and ethically, only on systems you own or have explicit permission to test. While Termux provides the environment for these tools, responsible usage is paramount. Understanding these tools helps you learn about cybersecurity and protect your own digital assets. In the next chapter, we'll explore how to use Termux for SSH and remote server access, connecting your mobile device to powerful remote machines.

---

# Chapter 15: SSH & Remote Server Access with Termux

---

## Introduction

---

Secure Shell (SSH) is a cryptographic network protocol that allows secure remote access to computers over an unsecured network. For Termux users, SSH is incredibly powerful, enabling you to connect to and control remote servers, cloud instances, or even other Linux machines from your Android device. This transforms your phone into a portable command center, allowing you to manage your web servers, deploy code, or perform administrative tasks from anywhere with an internet connection.

This chapter will guide you through setting up SSH in Termux, connecting to a remote server, and using SSH keys for more secure and convenient authentication. We will cover both connecting *from* Termux to a remote server and briefly touch upon connecting *to* Termux from another device.

## Step-by-step guide

---

### 1. Connecting from Termux to a Remote Server

This is the most common use case: using Termux as an SSH client to control a remote machine.

1. **Install OpenSSH:** Termux provides the OpenSSH client, which is essential for making SSH connections.

```
bash pkg install openssh
```

2. **Basic SSH Connection (Password Authentication):** To connect to a remote server, you need its username and IP address (or hostname).

```
bash ssh username@remote_ip_address
```

*Replace username with your remote server username and remote\_ip\_address with the server's IP or hostname.*

The first time you connect to a new server, you will be asked to confirm the server's authenticity. Type `yes` and press Enter. Then, you will be prompted for the password of the remote user.

3. **SSH Key-Based Authentication (Recommended):** Password authentication can be cumbersome and less secure. SSH keys provide a much more secure and convenient way to log in. You generate a pair of keys: a private key (kept secret on your Termux device) and a public key (uploaded to the remote server).

- **Generate SSH Key Pair in Termux:**

```
bash ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

*The -t rsa specifies the encryption type, -b 4096 sets the key length (stronger), and -C adds a comment for identification. You will be prompted to enter a passphrase. It's highly recommended to use a strong passphrase to protect your private key. Press Enter to accept the default file location ( ~/.ssh/id\_rsa ).*

This will create two files in your `~/.ssh` directory: `id_rsa` (your private key) and `id_rsa.pub` (your public key).

- **Copy Public Key to Remote Server:** The easiest way to copy your public key to the remote server is using `ssh-copy-id`.

```
bash ssh-copy-id username@remote_ip_address
```

*You will be prompted for the remote user's password.* This command appends your public key to the `~/.ssh/authorized_keys` file on the remote server.

If `ssh-copy-id` is not available or doesn't work, you can manually copy the content of `~/.ssh/id_rsa.pub` and paste it into the `~/.ssh/authorized_keys` file on the remote server.

```
bash cat ~/.ssh/id_rsa.pub
```

 Copy the output, then SSH into your server with password, and use `nano ~/.ssh/authorized_keys` to paste it on a new line. (Create the `.ssh` directory and `authorized_keys` file if they don't exist, ensuring correct permissions: `mkdir -p ~/.ssh && chmod 700 ~/.ssh && touch ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys`).

- **Connect using SSH Key:** Now, when you try to connect, you should be prompted for your SSH key passphrase (if you set one), not the remote user's password.

```
bash ssh username@remote_ip_address
```

## 2. Connecting to Termux from Another Device (SSH Server)

Termux can also act as an SSH server, allowing you to connect to your Android device from a computer or another phone. This is useful for transferring files or running commands on your Termux environment from a larger screen.

1. **Start SSH Server in Termux:**

```
bash sshd
```

*This starts the SSH daemon. It will run in the background. The default SSH port in Termux is 8022.* To stop it, you might need to find its process ID (`ps aux | grep sshd`) and `kill` it.
2. **Find Your Termux IP Address:**

```
bash ifconfig
```

 Look for your device's local IP address (e.g., `192.168.1.105`).
3. **Connect from Another Device:** From your computer's terminal or another Linux machine:

```
bash ssh -p 8022 localhost # If connecting from Termux to itself ssh -p 8022 username@your_termux_ip_address
```

*The username for connecting to Termux is your Termux username, which is usually `u0_aXXX` (where XXX is a number) or `root` if you've set it up. You can find your username by typing `whoami` in Termux. The password is the one you set using `passwd` in Termux.*

**Important:** For security, it's generally not recommended to expose your Termux SSH server to the public internet unless you know what you're doing. Use it primarily on your local network.

## Important commands in code blocks

```
Install OpenSSH
pkg install openssh

Connect to a remote server
ssh username@remote_ip_address

Generate SSH key pair
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

Copy public key to remote server
ssh-copy-id username@remote_ip_address

Start Termux SSH server
sshd

Find your Termux IP address
ifconfig
```

## Output examples if possible

**First time SSH connection prompt:**

```
The authenticity of host 'remote_ip_address (remote_ip_address)' can't be established.
ECDSA key fingerprint is SHA256:...
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'remote_ip_address' (ECDSA) to the list of known hosts.
```

### ssh-keygen output:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/data/data/com.termux/files/home/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /data/data/com.termux/files/home/.ssh/id_rsa
Your public key has been saved in /data/data/com.termux/files/home/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:...
your_email@example.com
The key's randomart image is:
+---[RSA 4096]-----+
|
| .
| .
| . + .
| . = B
| . * S .
| . + = +
| . o .
| . E
| .
+---[SHA256]-----+
```

## Practice task

---

Your task is to set up SSH key-based authentication for a remote server (you can use a free tier cloud instance like Oracle Cloud Free Tier, or a local virtual machine if you have one).

1. Install `openssh` in Termux.
2. Generate an SSH key pair in Termux, making sure to set a strong passphrase.
3. Copy your public key to your remote server using `ssh-copy-id` (or manually if needed).
4. Attempt to SSH into your remote server from Termux using your SSH key. You should be prompted for your passphrase, not the remote user's password.
5. (Optional) If you have another device (e.g., a computer), try to SSH into your Termux environment from that device. Remember to use port 8022.

## Chapter summary

---

This chapter has provided you with the knowledge and tools to securely access and manage remote servers using SSH from your Termux environment. You've learned how to install OpenSSH, connect using password authentication, and, more importantly, how to set up and use SSH keys for enhanced security and convenience. You also briefly explored how to turn your Termux device into an SSH server. SSH is an indispensable tool for anyone working with remote systems, and mastering it in Termux significantly expands your mobile computing capabilities. In the next chapter, we'll explore how to get a graphical Linux environment running within Termux using VNC Viewer.

---

# Chapter 16: VNC Viewer: GUI Linux in Termux

---

## Introduction

---

While the command line is powerful, sometimes a graphical user interface (GUI) is simply more convenient or necessary for certain tasks. Termux, being a command-line environment, doesn't natively provide a GUI. However, you can set up a full-fledged Linux desktop environment within Termux and access it remotely using a VNC (Virtual Network Computing) viewer. This allows you to run GUI applications like web browsers, office suites, or development environments, all from your Android device.

This chapter will guide you through installing a lightweight desktop environment (like XFCE or LXDE) and a VNC server in Termux, and then connecting to it using a VNC client on your Android phone or another device. This will transform your mobile device into a portable Linux desktop.

## Step-by-step guide

---

Setting up a GUI in Termux involves several steps:

1. **Update Termux and Install Core Packages:** Always start by ensuring your Termux is up-to-date and install necessary packages.

```
bash pkg update && pkg upgrade pkg install x11-repo # This repository contains GUI-related packages pkg install tigervnc # This installs the VNC server
```

2. **Install a Desktop Environment:** You need to choose a desktop environment. XFCE and LXDE are lightweight and suitable for mobile devices. We'll use XFCE as an example.

```
bash pkg install xfce4 # Or pkg install lxde This might take a while as it downloads many packages.
```

3. **Set up VNC Server:** After installing the VNC server, you need to configure it.

```
bash vncserver The first time you run vncserver, it will ask you to set a VNC password (this is separate from your Termux password). Make sure to remember it. It will also ask if you want to set a view-only password (optional). After setting the password, it will start a VNC server and tell you the display number (e.g., :1).
```

*If you run vncserver again, it might start on a different display number (e.g., :2). You can kill existing VNC servers with vncserver -kill :1 (replace :1 with your display number).*

4. **Create a VNC Startup Script:** You need to tell the VNC server what desktop environment to start. Create a file named `~/.vnc/xstartup`:

```
bash nano ~/.vnc/xstartup
```

Add the following content for XFCE:

```
`` `bash
```

## #!/bin/bash

---

```
startxfce4 & `` *If you installed LXDE, use startlxde &` instead.*
```

Make the script executable:

```
bash chmod +x ~/.vnc/xstartup
```

5. **Restart VNC Server:** Kill any running VNC server and restart it to apply the `xstartup` changes.

```
bash vncserver -kill :1 # Replace :1 with your VNC display number if different vncserver Note the new display number (e.g., :1).
```

6. **Install a VNC Viewer App:** Download a VNC viewer app from the Google Play Store or F-Droid on your Android device. Popular choices include "VNC Viewer" by RealVNC or "bVNC Secure VNC Viewer."

7. **Connect with VNC Viewer:**

- Open your VNC viewer app.
- Create a new connection.
- For the address, use `localhost:1` (if connecting from the same device, and your VNC server is on display `:1`). If connecting from another device on the same network, use `YOUR_TERMUX_IP:1` (replace `YOUR_TERMUX_IP` with your Android device's IP address, found using `ifconfig` in Termux).
- Enter the VNC password you set earlier.

You should now see your Linux desktop environment running within the VNC viewer!

## Important commands in code blocks

---

```
Update and install x11-repo and tigervnc
pkg update && pkg upgrade
pkg install x11-repo tigervnc

Install a desktop environment (e.g., XFCE)
pkg install xfce4

Start VNC server for the first time (set password)
vncserver

Kill a running VNC server (replace :1 with your display number)
vncserver -kill :1

Create/edit VNC startup script
nano ~/.vnc/xstartup

Make xstartup executable
chmod +x ~/.vnc/xstartup

Restart VNC server
vncserver

Find your Termux IP address
ifconfig
```

## Output examples if possible

---

### vncserver output (first run):

```
You will require a password to access your desktops.

Password:
Verify:
xauth: file /data/data/com.termux/files/home/.Xauthority does not exist

New 'X' desktop is localhost:1

Creating default startup script /data/data/com.termux/files/home/.vnc/xstartup
Starting applications specified in /data/data/com.termux/files/home/.vnc/xstartup
Log file is /data/data/com.termux/files/home/.vnc/localhost:1.log
```

## Practice task

---

Your task is to set up and connect to a GUI Linux environment in Termux.

1. Install `x11-repo` and `tigervnc`.
2. Install either `xfce4` or `lxde`.
3. Run `vncserver` for the first time and set a password.
4. Create and make executable the `~/.vnc/xstartup` file with the correct desktop environment command.
5. Restart your VNC server.
6. Download and install a VNC Viewer app on your Android device.
7. Connect to your Termux VNC server using `localhost:1` (or your device's IP if connecting from another device) and the password you set.
8. Once connected, try opening a web browser (like Firefox, if installed within XFCE/LXDE) or a terminal within the GUI.

## Chapter summary

---

In this chapter, you've unlocked a new dimension of Termux's capabilities by setting up a graphical Linux desktop environment. You've learned how to install a VNC server and a lightweight desktop environment like XFCE, configure the VNC startup script, and connect to it using a VNC viewer. This powerful setup allows you to run GUI applications and interact with your Termux environment in a more visual way, bridging the gap between the command line and a traditional desktop experience. In the next

chapter, we'll explore how to interact with your Android device's hardware and features directly from Termux using the Termux API.

---

## Chapter 17: Using Termux API (Camera, SMS, Battery)

---

### Introduction

---

Termux is not just a Linux environment; it can also interact directly with your Android device's hardware and features. This is made possible through the Termux:API add-on, which provides a set of command-line utilities to access functionalities like the camera, SMS, battery status, GPS, and more. This capability allows you to create powerful scripts that bridge the gap between your Linux environment and your mobile device's native features, opening up a world of automation and unique applications.

This chapter will guide you through installing and using the Termux:API tools to interact with your device's camera, send SMS messages, and retrieve battery information. We'll demonstrate how to use these commands and integrate them into simple shell scripts.

### Step-by-step guide

---

To use the Termux:API, you first need to install the Termux:API app from F-Droid (as mentioned in Chapter 2) and then install the `termux-api` package within Termux.

#### 1. Install Termux:API App and Package:

- Ensure you have the **Termux:API app** installed from F-Droid on your Android device.
- Install the `termux-api` package in Termux:

```
bash pkg install termux-api
```

2. **Grant Permissions:** When you first use a Termux:API command that requires a specific permission (e.g., camera, SMS), Android will prompt you to grant that permission to the Termux:API app. **You must grant these permissions for the commands to work.** You can also grant them manually via Android Settings -> Apps -> Termux:API -> Permissions.

#### 1. Battery Information

Retrieve detailed information about your device's battery status.

```
termux-battery-status
```

This command will output a JSON (JavaScript Object Notation) string containing battery health, percentage, status, temperature, and more.

#### 2. Camera Interaction

Use your device's camera to take photos or record videos.

1. **Take a Photo:** `bash termux-camera-photo -c 0 ~/camera_photo.jpg -c 0` specifies the front camera (0 for front, 1 for back). `~/camera_photo.jpg` is the path where the photo will be saved. The photo will be saved in your Termux home directory.
2. **Record a Video:** `bash termux-camera-record -c 1 -d 10 ~/camera_video.mp4 -c 1` specifies the back camera. `-d 10` sets the duration to 10 seconds. `~/camera_video.mp4` is the output file path.

### 3. Sending SMS

Send SMS messages directly from your Termux terminal.

```
termux-sms-send -n "+8801XXXXXXX" "Hello from Termux!"
```

Replace `+8801XXXXXXX` with the recipient's phone number (including country code) and `"Hello from Termux!"` with your message.

### 4. Other Useful Termux:API Commands

- `termux-clipboard-get` : Get text from the Android clipboard.
- `termux-clipboard-set <text>` : Set text to the Android clipboard.
- `termux-toast <message>` : Display a small pop-up (toast) message on your screen.
- `termux-vibrate -d <duration_ms>` : Vibrate the device for a specified duration.
- `termux-location` : Get device location (requires GPS).
- `termux-notification --title "Title" --content "Message"` : Display a notification.

## Important commands in code blocks

```
Install Termux:API package
pkg install termux-api

Get battery status
termux-battery-status

Take a photo with the front camera and save it
termux-camera-photo -c 0 ~/my_selfie.jpg

Send an SMS message
termux-sms-send -n "+8801XXXXXXX" "Testing Termux SMS API."

Display a toast message
termux-toast "Task Completed!"

Get clipboard content
termux-clipboard-get

Set clipboard content
termux-clipboard-set "This text is from Termux."
```

## Output examples if possible

Output of `termux-battery-status` :

```
{
 "health": "GOOD",
 "percentage": 85,
 "plugged": "AC",
 "status": "CHARGING",
 "temperature": 30.5,
 "current": 123456,
 "voltage": 4200
}
```

Output of `termux-clipboard-get` :

```
This is some text from my clipboard.
```

## Practice task

Your task is to create a simple shell script that uses a few Termux:API commands.



1. Create a script named `device_info.sh`.
2. Inside the script, use `termux-battery-status` to get the battery percentage and print it in a user-friendly format (e.g., "Battery: 85%"). You'll need to parse the JSON output using a tool like `jq` (install with `pkg install jq`).
3. Add a `termux-toast` message that says "Device info retrieved!".
4. Make the script executable and run it.
5. (Optional) Modify the script to take a quick photo and then display a toast message confirming the photo was taken.

---

## Chapter summary

In this chapter, you've explored the powerful capabilities of the Termux:API, allowing your Termux environment to interact directly with your Android device's hardware and features. You've learned how to retrieve battery information, take photos, send SMS messages, and use other useful commands like clipboard access and toast notifications. This integration opens up exciting possibilities for creating highly personalized and automated scripts that leverage the full potential of your mobile device. In the next chapter, we'll focus on the crucial task of backing up and restoring your entire Termux setup, ensuring your hard work is always safe.

---

# Chapter 18: Backup & Restore Full Termux Setup

---

## Introduction

---

As you invest more time customizing your Termux environment, installing packages, writing scripts, and developing projects, your setup becomes increasingly valuable. Losing this progress due to a device reset, a corrupted installation, or simply upgrading to a new phone can be frustrating. Therefore, knowing how to properly back up and restore your entire Termux setup is an essential skill for any serious user.

This chapter will guide you through the process of backing up your Termux environment, including all your installed packages, configuration files, and home directory contents. We will also cover how to restore this backup to a new or reinstalled Termux instance, ensuring that your hard work is always safe and easily recoverable.

## Step-by-step guide

---

Backing up and restoring Termux involves compressing your Termux data directory and then extracting it. This method captures almost everything, including your installed packages, `~/.termux` configurations, and your home directory (`~`).

### 1. Backing Up Your Termux Setup

1. **Ensure you have `tar` installed:** `tar` is a common archiving utility used for creating compressed archives.

```
bash pkg install tar
```

2. **Exit Termux:** It's crucial to exit Termux completely before performing a backup to ensure all files are closed and consistent. You can do this by typing `exit` multiple times until the Termux session closes, or by force-closing the app from Android's app switcher.
3. **Use Android's File Manager or a PC:** You need to access the Termux data directory from outside the Termux app. This directory is usually located at `/data/data/com.termux/files/`.
  - **Using a File Manager (e.g., Files by Google, Solid Explorer):**
    - Navigate to `Internal Storage/Android/data/com.termux/files/`.
    - You will see a `home` folder and a `usr` folder. These contain all your Termux data.
    - Select both `home` and `usr` folders.

- Compress them into a `.tar.gz` or `.zip` archive. Most modern file managers have a

compress/archive option. Save this archive to a safe location, preferably off your device (e.g., cloud storage, external drive).

```
* **Using `tar` from a Rooted Device or ADB (Advanced):**
 If your device is rooted or you have ADB (Android Debug Bridge) access from a PC, you can create the backup directly using `tar`.

  ```bash
  # From a rooted shell or ADB shell
  su # if not already root
  cd /data/data/com.termux/files/
  tar -czvf /sdcard/termux_backup_$(date +%Y%m%d).tar.gz home usr
  ```

 This command creates a compressed tar archive of `home` and `usr` directories and saves it to your internal storage (`/sdcard`).
```

## 2. Restoring Your Termux Setup

To restore your Termux setup, you will typically need a fresh installation of Termux.

1. **Install a Fresh Termux:** Install Termux from F-Droid (as in Chapter 2).
2. **Grant Storage Permissions:** Run `termux-setup-storage` to ensure Termux can access your internal storage where your backup file is located.

```
bash termux-setup-storage
```

3. **Transfer Backup File:** Copy your `termux_backup.tar.gz` (or `.zip`) file to your device's internal storage, preferably in the `~/storage/shared` directory or `~/downloads`.
4. **Remove Existing Termux Data (Caution!):** Before restoring, it's best to remove the default `home` and `usr` directories of the fresh Termux installation to avoid conflicts. **Be extremely careful with this step, as it deletes your current Termux data.**

```
bash rm -rf $HOME/* `PREFIX`/* $HOME refers to /data/data/com.termux/files/home and `PREFIX` refers to /data/data/com.termux/files/usr.
```

5. **Extract the Backup:** Navigate to the root of your Termux data directory and extract your backup file.

```
bash cd /data/data/com.termux/files/ tar -xzf ~/storage/shared/termux_backup_$(date +%Y%m%d).tar.gz
Replace ~/storage/shared/termux_backup_$(date +%Y%m%d).tar.gz with the actual path to your backup file.
```

6. **Fix Permissions (Important!):** After extraction, some files might have incorrect permissions. It's crucial to fix them.

```
bash termux-fix-shebangs This command fixes the shebang lines in scripts to point to the correct Termux interpreter paths.
```

You might also need to run `pkg upgrade` again to ensure all package links are correctly re-established.

```
bash pkg upgrade
```

7. **Restart Termux:** Close and reopen Termux. Your environment should now be restored to its backed-up state.

## Important commands in code blocks

---

```
Install tar for archiving
pkg install tar

Command to create a backup (from rooted shell or ADB)
tar -czvf /sdcard/termux_backup_$(date +%Y%m%d).tar.gz /data/data/com.termux/files/home
/data/data/com.termux/files/usr

Grant storage permissions in Termux
termux-setup-storage

Remove existing Termux data before restoring (CAUTION!)
rm -rf $HOME/* `PREFIX/*`

Extract the backup file (adjust path to your backup)
cd /data/data/com.termux/files/
tar -xzf ~/storage/shared/your_backup_file.tar.gz

Fix script shebangs after restore
termux-fix-shebangs

Re-establish package links
pkg upgrade
```

## Output examples if possible

---

### Output of `tar -czvf ...` (example):

```
home/
home/my_script.sh
home/.bashrc
usr/
usr/bin/
usr/bin/python
... (many more files)
```

### Output of `tar -xzf ...` (example):

```
home/
home/my_script.sh
home/.bashrc
usr/
usr/bin/
usr/bin/python
... (many more files)
```

## Practice task

---

Your task is to perform a full backup of your current Termux setup and then simulate a restore (if you have a secondary device or are comfortable reinstalling Termux).

1. Ensure `tar` is installed.
2. Exit Termux completely.
3. Using your Android file manager, navigate to `/data/data/com.termux/files/` and compress the `home` and `usr` folders into a `.tar.gz` file. Save it to `~/storage/shared/`.
4. (Optional, but recommended for testing) Uninstall Termux and reinstall it from F-Droid.
5. Run `termux-setup-storage` on the fresh installation.
6. Copy your backup file back to `~/storage/shared/` if you moved it.
7. Remove the default `home` and `usr` directories in the fresh Termux installation.
8. Extract your backup file into `/data/data/com.termux/files/`.
9. Run `termux-fix-shebangs` and `pkg upgrade`.
10. Restart Termux and verify that your custom prompt, installed packages (e.g., `neofetch`), and scripts are all back.

## Chapter summary

---

This chapter has provided you with the essential knowledge and steps to back up and restore your entire Termux environment. You learned how to create a comprehensive archive of your `home` and `usr` directories and how to extract it to restore your setup, including fixing permissions. Regularly backing up your Termux installation is a critical habit to adopt, protecting your valuable work and configurations. With this skill, you can confidently experiment, upgrade devices, or recover from issues, knowing your Termux setup is safe. In the next chapter, we will apply all the knowledge gained so far to build full-fledged projects, combining various tools and programming languages.

---

# Chapter 19: Full Projects: Tools, Bots, Servers

---

## Introduction

---

Throughout this book, you've acquired a diverse set of skills in Termux, from basic command-line navigation and package management to shell scripting, Python, Node.js, and even interacting with your Android device's API. Now, it's time to bring all these pieces together and build some complete, practical projects. This chapter will guide you through developing more complex tools, bots, and servers, demonstrating how different Termux capabilities can be combined to create powerful applications.

We will focus on three types of projects: a custom CLI tool, a simple Telegram bot, and a more advanced web server. These projects will serve as examples, encouraging you to adapt and expand upon them to suit your own needs and ideas.

## Step-by-step guide

---

### Project 1: Advanced CLI System Information Tool

Let's create a CLI tool that provides detailed system information, combining shell scripting with Python for more complex data processing.

1. **Create Project Directory:** `bash mkdir sysinfo_tool cd sysinfo_tool`
2. **Create the Main Shell Script ( `sysinfo` ):** This script will act as the entry point for our tool.

```
bash nano sysinfo
```

Add the following content:

```
```bash
```

#!/bin/bash

Main CLI tool for system information

```
function show_help { echo "Usage: sysinfo [option]" echo "Options:" echo " -b, --battery Show battery status" echo " -s, --storage Show storage usage" echo " -n, --network Show network information" echo " -h, --help Show this help message" echo "If no option is provided, shows a summary." }
```

```
function get_battery_status { # Requires termux-api and jq if ! command -v termux-battery-status &> /dev/null || ! command -v jq &> /dev/null; then echo "Error: termux-api or jq not installed. Please install them: pkg install termux-api jq" exit 1 fi echo "--- Battery Status ---" termux-battery-status | jq ".percentage, .status, .health, .temperature" }
```

```
function get_storage_usage { echo "--- Storage Usage ---" df -h /data/data/com.termux/files/home df -h ~/storage/shared }

function get_network_info { echo "--- Network Information ---" ifconfig wlan0 echo "Public IP: $(curl -s ifconfig.me)" }

case "$1" in -b|--battery) get_battery_status ;; -s|--storage) get_storage_usage ;; -n|--network) get_network_info ;; -h|--help)
show_help ;; *) echo "--- System Summary ---" get_battery_status echo "\n" get_storage_usage echo "\n" get_network_info
;; esac ````
```

3. **Make Executable and Place in PATH:** `bash chmod +x sysinfo mv sysinfo ~/bin/`

4. **Install Dependencies:** `bash pkg install termux-api jq net-tools curl`

5. **Test the Tool:** `bash sysinfo sysinfo -b sysinfo --help`

Project 2: Simple Telegram Echo Bot (Python)

Let's create a basic Telegram bot that echoes messages back to the user. This requires a Telegram bot token and the `python-telegram-bot` library.

1. Get a Telegram Bot Token:

- Open Telegram and search for `@BotFather`.
- Start a chat with BotFather and send `/newbot`.
- Follow the instructions to choose a name and username for your bot. BotFather will give you an **HTTP API Token** (e.g., `123456:ABC-DEF1234ghIkl-789_jkl-LMNOP`). Keep this token secret!

2. **Install Python and `python-telegram-bot`:** `bash pkg install python pip install python-telegram-bot --upgrade`

3. **Create the Bot Script (`echo_bot.py`):** `bash nano echo_bot.py`

Add the following Python code:

```
````python
```

## echo\_bot.py

---

```
from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters, ContextTypes
import logging
```

## Enable logging

---

```
logging.basicConfig(format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO)
logging.getLogger("httpx").setLevel(logging.WARNING)
```

## Replace with your actual bot token

---

```
TOKEN = "YOUR_TELEGRAM_BOT_TOKEN"
```

```
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
 await update.message.reply_text(f"Hello {update.effective_user.first_name}! I am an echo bot. Send me a message!")
```

```
async def echo(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
 await update.message.reply_text(update.message.text)
```

```
async def unknown(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
 await update.message.reply_text("Sorry, I didn't understand that command.")
```

```
def main(): application = ApplicationBuilder().token(TOKEN).build()
```

```
 application.add_handler(CommandHandler("start", start))
 application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, echo))
 application.add_handler(MessageHandler(filters.COMMAND, unknown))

 application.run_polling(allowed_updates=Update.ALL_TYPES)
```

if **name** == "**main**": main() `` \*Replace YOUR\_TELEGRAM\_BOT\_TOKEN` with the token you got from BotFather.\*

4. **Run the Bot:** bash python echo\_bot.py The bot will start running. Go to your Telegram app, find your bot by its username, and send it messages. It should echo them back.

*To stop the bot, press Ctrl + C in Termux.*

### Project 3: Simple Node.js REST API Server

Let's create a simple REST API using Node.js and Express.js that manages a list of tasks.

1. **Create Project Directory and Initialize:** bash mkdir task\_api cd task\_api npm init -y npm install express body-parser

2. **Create the Server Script ( server.js ):** bash nano server.js

Add the following content:

```
````javascript // server.js const express = require("express"); const bodyParser = require("body-parser"); const app =
express(); const port = 3000;
```

```
app.use(bodyParser.json());
```

```
let tasks = [ { id: 1, title: "Learn Termux", completed: false }, { id: 2, title: "Build a bot", completed: true } ];
```

```
// GET all tasks app.get("/tasks", (req, res) => { res.json(tasks); });
```

```
// GET a single task by ID app.get("/tasks/:id", (req, res) => { const taskId = parseInt(req.params.id); const task = tasks.find(t
=> t.id === taskId); if (task) { res.json(task); } else { res.status(404).send("Task not found"); } });
```

```
// POST a new task app.post("/tasks", (req, res) => { const newTask = { id: tasks.length + 1, title: req.body.title, completed:
false }; tasks.push(newTask); res.status(201).json(newTask); });
```

```
// PUT (update) a task app.put("/tasks/:id", (req, res) => { const taskId = parseInt(req.params.id); const taskIndex =
tasks.findIndex(t => t.id === taskId); if (taskIndex !== -1) { tasks[taskIndex].title = req.body.title || tasks[taskIndex].title;
tasks[taskIndex].completed = req.body.completed !== undefined ? req.body.completed : tasks[taskIndex].completed;
res.json(tasks[taskIndex]); } else { res.status(404).send("Task not found"); } });
```

```
// DELETE a task app.delete("/tasks/:id", (req, res) => { const taskId = parseInt(req.params.id); const initialLength =
tasks.length; tasks = tasks.filter(t => t.id !== taskId); if (tasks.length < initialLength) { res.status(204).send(); // No Content }
else { res.status(404).send("Task not found"); } });
```

```
app.listen(port, () => { console.log( Task API listening at http://localhost:${port} ); }); ````
```

3. **Run the Server:** bash node server.js

4. **Test the API (using curl):** Open another Termux session or use a different terminal.

```
````bash
```

## Get all tasks

---

curl http://localhost:3000/tasks

## Get a specific task

---

curl http://localhost:3000/tasks/1

## Add a new task

---

curl -X POST -H "Content-Type: application/json" -d '{"title":"Write book chapter"}' http://localhost:3000/tasks

## Update a task

---

curl -X PUT -H "Content-Type: application/json" -d '{"completed":true}' http://localhost:3000/tasks/1

## Delete a task

---

curl -X DELETE http://localhost:3000/tasks/2 ````

## Important commands in code blocks

---

```
Install dependencies for CLI tool
pkg install termux-api jq net-tools curl

Install Python Telegram Bot library
pip install python-telegram-bot --upgrade

Install Node.js and Express/body-parser for API server
pkg install nodejs
npm install express body-parser

Example of running Python bot
python echo_bot.py

Example of running Node.js server
node server.js

Example curl commands for API testing
curl http://localhost:3000/tasks
curl -X POST -H "Content-Type: application/json" -d '{"title":"New Task"}' http://localhost:3000/tasks
```

## Output examples if possible

---

`sysinfo` output:

```

--- System Summary ---
--- Battery Status ---
85
"CHARGING"
"GOOD"
30.5

--- Storage Usage ---
Filesystem Size Used Avail Use% Mounted on
/data/data/com.termux/files/home
 10G 2.5G 7.5G 25% /data/data/com.termux/files/home
Filesystem Size Used Avail Use% Mounted on
/storage/emulated/0
 64G 20G 44G 32% /data/data/com.termux/files/home/storage/shared

--- Network Information ---
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
 inet 192.168.1.105 netmask 255.255.255.0 broadcast 192.168.1.255
 inet6 fe80::... prefixlen 64 scopeid 0x20<link>
 ether 00:11:22:33:44:55 txqueuelen 1000 (Ethernet)
 RX packets 12345 bytes 67890 (67.8 KB)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 54321 bytes 98765 (98.7 KB)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
Public IP: 203.0.113.45

```

### Telegram Bot Interaction (in Telegram app):

User: /start Bot: Hello [Your Name]! I am an echo bot. Send me a message! User: Hello Termux! Bot: Hello Termux!

### API Server `curl` output:

```

curl http://localhost:3000/tasks
[{"id":1,"title":"Learn Termux","completed":false},{ "id":2,"title":"Build a bot","completed":true}]

curl -X POST -H "Content-Type: application/json" -d '{"title":"Write book chapter"}' http://localhost:3000/tasks
{"id":3,"title":"Write book chapter","completed":false}

```

## Practice task

Your task is to extend one of the projects or create a new simple project that combines at least two different Termux skills you've learned.

**Option A: Enhance the CLI Tool:** Modify the `sysinfo` tool to include a new option, e.g., `-p`, `--packages` that lists all installed Termux packages (`pkg list-installed`).

**Option B: Enhance the Telegram Bot:** Modify the `echo_bot.py` to respond to a `/sysinfo` command by sending back the battery status using `termux-battery-status` (you'll need to parse the JSON output in Python).

**Option C: Enhance the REST API:** Add a new endpoint to the Node.js API server, e.g., `/status`, that returns the current battery percentage of the Termux device using `termux-battery-status` (you'll need to execute a shell command from Node.js and parse its output).

## Chapter summary

In this chapter, you've moved beyond individual commands and scripts to build complete, multi-component projects within Termux. You've seen how to integrate shell scripting with Python for a CLI tool, develop a functional Telegram bot, and create a basic REST API server using Node.js. These projects demonstrate the power and flexibility of Termux as a mobile development environment. By combining the skills you've learned throughout this book, you can now tackle more ambitious projects and truly unlock the potential of Linux on your Android device. In our final chapter, we'll look at automation, AI tools, and achieving final mastery with Termux.



# Chapter 20: Automation, AI Tools & Final Mastery

---

## Introduction

---

Congratulations! You've journeyed from a beginner to an advanced Termux user, mastering everything from basic commands to building complex projects. In this final chapter, we'll explore the cutting edge of Termux capabilities: automation and the integration of AI tools. Automation allows you to schedule tasks, respond to events, and make your Termux environment work for you, even when you're not actively using it. Integrating AI tools opens up possibilities for intelligent scripting, natural language processing, and machine learning on your mobile device.

This chapter will tie together many of the concepts you've learned, pushing the boundaries of what's possible with Termux. We'll look at scheduling tasks, using simple AI-related tools, and discuss what it means to achieve 'final mastery' in Termux.

## Step-by-step guide

---

### 1. Automation with Cron

Cron is a time-based job scheduler in Unix-like operating systems. It allows you to schedule commands or scripts to run automatically at specified intervals.

1. **Install cronie** : Termux uses `cronie` for cron functionality.

```
bash pkg install cronie
```

2. **Start the Cron Service:**

```
bash crond
```

*This starts the cron daemon in the background. You might want to add `crond` to your `~/.bashrc` or `~/.profile` to start it automatically when Termux launches.*

3. **Edit your Crontab:** `crontab` is the command used to maintain cron entries.

```
bash crontab -e
```

 This will open a text editor (like `nano` ) with your crontab file. Each line in this file represents a scheduled task.

**Crontab Syntax:** `* * * * *` `command_to_execute` - Minute (0-59) - Hour (0-23) - Day of month (1-31) - Month (1-12) - Day of week (0-7, where 0 and 7 are Sunday)

**Example:** Schedule a script to run every day at 9:00 AM.

```
0 9 * * * /data/data/com.termux/files/home/my_daily_script.sh
```

*Make sure to use the full path to your script. You can also redirect output to a log file: `>> ~/cron_log.txt 2>&1`.*

4. **List Cron Jobs:**

```
bash crontab -l
```

### 2. Simple AI Tools and Libraries

While running heavy AI models directly on a mobile device can be challenging, Termux allows you to use Python and Node.js libraries that interact with AI services or perform lightweight AI tasks.

1. **Natural Language Processing (NLP) with NLTK (Python):** NLTK (Natural Language Toolkit) is a leading platform for building Python programs to work with human language data.

```
bash pkg install python pip install nltk python -c "import nltk; nltk.download('punkt')"
```

**Example Python script ( `nlp_example.py` ):**

```
```python
```

nlp_example.py

```
import nltk from nltk.tokenize import word_tokenize
```

```
text = "Termux is an amazing tool for mobile development and learning Linux." words = word_tokenize(text) print(f"Original text: {text}") print(f"Tokenized words: {words}") ````
```

Run it:

```
bash python nlp_example.py
```

2. Image Processing with Pillow (Python): Pillow is a powerful image processing library for Python.

```
bash pip install Pillow
```

Example Python script (`image_process.py`):

```
````python
```

# image\_process.py

---

```
from PIL import Image, ImageFilter
```

```
try: # Create a dummy image if one doesn't exist for demonstration img = Image.new("RGB", (100, 100), color = "red")
img.save("red_square.png") print("Created red_square.png")
```

```
Open an image (replace with your image path if you have one)
img = Image.open("red_square.png")

Apply a blur filter
blurred_img = img.filter(ImageFilter.BLUR)
blurred_img.save("blurred_red_square.png")
print("Created blurred_red_square.png")
```

```
except FileNotFoundError: print("Please ensure 'red_square.png' exists or create one.") except Exception as e: print(f"An error occurred: {e}") ````
```

Run it:

```
bash python image_process.py
```

## 3. Final Mastery: Continuous Learning and Contribution

Achieving "final mastery" in Termux isn't about knowing every command, but about understanding the underlying principles and continuously learning. Here are some tips for continued growth:

- **Read Man Pages:** For any command, type `man <command_name>` (e.g., `man ls`) to read its manual page. This is the most comprehensive source of information.
- **Explore Termux Wiki:** The official Termux Wiki (<https://wiki.termux.com/>) is an invaluable resource.
- **Join Communities:** Engage with other Termux users on forums, Reddit (r/termux), or Telegram groups. Sharing knowledge and asking questions is key.
- **Contribute:** If you find a bug, write a useful script, or discover a new trick, consider contributing back to the community or open-source projects.
- **Experiment:** Don't be afraid to try new things. Create dummy files and directories to experiment with commands. The more you practice, the more comfortable you'll become.
- **Combine Tools:** The real power of Termux comes from combining different tools and languages. Think about how you can use Python to process data, shell scripts to automate workflows, and Termux:API to interact with your phone.

## Important commands in code blocks

```
Install cronie for task scheduling
pkg install cronie

Start the cron daemon
crond

Edit your crontab file
crontab -e

List your cron jobs
crontab -l

Install NLTK for NLP (Python)
pip install nltk
python -c "import nltk; nltk.download('punkt')"

Install Pillow for image processing (Python)
pip install Pillow

Read manual page for a command
man <command_name>
```

## Output examples if possible

### crontab -l output (example):

```
DO NOT EDIT THIS FILE - edit the master and reinstall.
(/tmp/crontab.XXXXXXX/crontab installed on Sat Jul 12 15:00:00 2025)
(Cronie version 3.0.1)
0 9 * * * /data/data/com.termux/files/home/my_daily_script.sh
```

### nlp\_example.py output:

```
Original text: Termux is an amazing tool for mobile development and learning Linux.
Tokenized words: ["Termux", "is", "an", "amazing", "tool", "for", "mobile", "development", "and", "learning", "Linux", "."]
```

### image\_process.py output:

```
Created red_square.png
Created blurred_red_square.png
```

## Practice task

Your final practice task is to set up a simple automated task and experiment with a basic AI-related library.

### 1. Automate a daily message:

- Create a simple shell script (e.g., `daily_message.sh`) that uses `termux-toast` to display a message like "Good morning from your automated Termux!" (You'll need `pkg install termux-api` if you haven't already).
- Make the script executable.
- Use `crontab -e` to schedule this script to run every day at a specific time (e.g., 7:00 AM).
- Verify your cron job with `crontab -l`.

### 2. Experiment with `nltk`:

- Write a Python script that takes a sentence as input (using `input()`) and then uses `nltk.word_tokenize` to break it into words. Print the original sentence and the tokenized words.

## Chapter summary

---

This chapter concludes your journey through "Beginners To Advance Termux." You've learned how to automate tasks using `cron`, explored basic AI-related functionalities with Python libraries like NLTK and Pillow, and gained insights into achieving true mastery through continuous learning and community engagement. Termux is a dynamic and evolving platform, and the skills you've acquired provide a strong foundation for endless exploration and development on your Android device. Keep experimenting, keep learning, and unlock the full potential of Linux in your pocket!

---

## 50-Termux Knowledge Checklist

---

Instructions: ✓ Tick each box when you know or have practiced it.

### Basics (1–10)

---

1. ☐ What is Termux and why is it useful?
2. ☐ How to install Termux from F-Droid?
3. ☐ What is the use of `termux-setup-storage` ?
4. ☐ Can you use `ls` , `cd` , and `pwd` correctly?
5. ☐ Do you know how to install packages using `pkg` ?
6. ☐ How do you create a new directory with `mkdir` ?
7. ☐ What is the difference between `cp` and `mv` ?
8. ☐ How do you make a shell script executable?
9. ☐ What is the purpose of the `~` (tilde) symbol in Termux?
10. ☐ How do you update your Termux packages?

### Intermediate (11–20)

---

1. ☐ How do you customize your Termux prompt (PS1)?
2. ☐ What is the shebang line in a shell script and why is it important?
3. ☐ How do you pass arguments to a shell script?
4. ☐ What is Git and why is it used for version control?
5. ☐ How do you clone a repository from GitHub?
6. ☐ What is the purpose of `git add` and `git commit` ?
7. ☐ How do you push your local changes to a remote GitHub repository?
8. ☐ Where should you place your custom CLI tools to make them accessible from anywhere?
9. ☐ How do you check the status of your Git repository?
10. ☐ Can you explain the basic structure of a `for` loop in Bash?

### Programming (21–30)

---

1. ☐ How do you install Python in Termux?
2. ☐ How do you run a Python script in Termux?
3. ☐ What is `pip` and how do you use it to install Python packages?
4. ☐ What is a Python virtual environment and why is it useful?

5. ☐ How do you install Node.js and npm in Termux?
6. ☐ How do you run a Node.js script in Termux?
7. ☐ What is `npm` and how do you use it to manage Node.js packages?
8. ☐ Can you create a simple "Hello World" program in both Python and Node.js?
9. ☐ How do you handle user input in a Python script?
10. ☐ How do you handle command-line arguments in a Node.js script?

## Advanced Tools (31–40)

---

1. ☐ How do you host a simple HTTP server using Python in Termux?
2. ☐ How do you host a simple PHP development server in Termux?
3. ☐ How do you host a simple Node.js Express server in Termux?
4. ☐ What is Nmap and what is its ethical use?
5. ☐ How do you perform a basic network scan using Nmap on your local network?
6. ☐ What is TShark and how can it be used ethically for network analysis?
7. ☐ How do you connect to a remote server using SSH from Termux?
8. ☐ What is the benefit of using SSH key-based authentication?
9. ☐ How do you set up a VNC server in Termux to get a GUI environment?
10. ☐ What is the Termux:API and how do you use it to access device features?

## Real-World Practice (41–50)

---

1. ☐ How do you get your device's battery status using Termux:API?
  2. ☐ How do you take a photo using Termux:API?
  3. ☐ How do you send an SMS message using Termux:API?
  4. ☐ How do you back up your entire Termux setup?
  5. ☐ What are the key directories to back up in Termux ( `home` and `usr` )?
  6. ☐ How do you restore a Termux backup to a new installation?
  7. ☐ What is `cron` and how do you schedule tasks in Termux?
  8. ☐ Can you explain the basic syntax of a crontab entry?
  9. ☐ How can Python libraries like NLTK or Pillow be used for AI-related tasks in Termux?
  10. ☐ What are some resources for continuous learning and community engagement in Termux?
- 

## Additional Features Notes

---

### Quizzes or MCQs

---

**Recommendation:** Include short quizzes or Multiple Choice Questions (MCQs) at the end of important chapters (e.g., after chapters on basic commands, shell scripting, Python, Git, etc.). These can be simple text-based questions with options, and the answers can be provided in a separate section or immediately after the questions.

**Example Question (for Chapter 4: Basic Linux Commands):**

Which command is used to change the current directory?

a) ls b) cd c) pwd d) mkdir

Answer: b) cd

## Dual Language (English + Bengali)

---

**Recommendation:** For key definitions, important commands, and code comments, provide optional Bengali translations. This can be done by placing the Bengali translation directly below the English text, perhaps in italics or a different font to distinguish it.

**Example (for Chapter 1: Introduction to Termux):**

**Termux:** A terminal emulator and Linux environment app for Android. *টার্মাক্স: অ্যান্ড্রয়েডের জন্য একটি টার্মিনাল এমুলেটর এবং লিনাক্স পরিবেশ অ্যাপ।*

**Example (for code comments):**

```
This command lists the contents of the current directory
এই কমান্ডটি বর্তমান ডিরেক্টরির বিষয়বস্তু তালিকাভুক্ত করে
ls
```

## Motivational Quotes or Learning Tips

---

**Recommendation:** Add motivational quotes or learning tips every 5 chapters. These can be short, inspiring messages related to learning, perseverance, or the power of technology.

**Example (after Chapter 5):**

*"Every line of code you write is a step towards mastering your craft. Keep building!"*

## Screenshot Recommendations

---

**Recommendation:** Suggest where screenshots can be added to enhance understanding, especially for visual steps or output. These suggestions should be placed within the relevant chapter content.

**Examples already included in chapters:**

- **Chapter 2:** A screenshot of the Termux app page in the F-Droid store.
- **Chapter 6:** Screenshots showing Android permissions settings for Termux.
- **Chapter 7:** Screenshots of the Termux long-press menu for theme/font selection.
- **General:** Screenshots of command output, file structures, or specific app interfaces (e.g., BotFather in Telegram).

## Resource Suggestions

---

**Recommendation:** Suggest links or resources where users can find more information, code examples, or community support. These should be relevant to the chapter content.

**Examples already included in chapters:**

- **Chapter 9 (Git & GitHub):** Link to GitHub.com for account creation.
- **Chapter 20 (Final Mastery):** Link to Termux Wiki (<https://wiki.termux.com/>) and mention Reddit (r/termux) or Telegram groups.
- **General:** Suggest linking to official documentation for specific tools or libraries (e.g., Python, Node.js, Nmap, etc.).