The Complete Termux Troubleshooting Handbook

A practical diagnostic system for Android terminal failures, package management, development tools, networking, and recovery

DEDSEC PROJECT - English

How to use this handbook

This handbook is organized around diagnosis rather than random command lists. Start by identifying the layer that failed: Android, the Termux application, the package manager, the shell, a language runtime, a project dependency, the network, or a remote service. A command that repairs one layer can damage another when used without evidence.

The safest workflow is:

1. Preserve important files before changing repositories, deleting environments, or reinstalling packages.
2. Capture the complete command and complete output, including the first error line.
3. Reproduce the problem with the smallest possible command.
4. Test one hypothesis at a time.
5. Verify the result with an explicit command rather than assuming success.
6. Record what changed so the repair can be reversed.

> **Safety boundary.** This book is for lawful administration, learning, development, and defensive troubleshooting. Never use a repair command you do not understand on another person's device, account, or infrastructure.

Contents

1. Build a diagnostic snapshot

A useful report describes the environment before any repair is attempted. The goal is not to collect everything; it is to collect enough evidence to distinguish an application problem from a package, project, permission, or network problem.

Run the following commands and save the output:

```
mkdir -p "$HOME/termux-diagnostics"
{
  echo '=== termux-info ==='
  termux-info 2>&1
  echo '=== system ==='
  uname -a
  echo '=== architecture ==='
  dpkg --print-architecture
  echo '=== paths ==='
  printf 'HOME=%s\nPREFIX=%s\nPATH=%s\n' "$HOME" "$PREFIX" "$PATH"
  echo '=== disk ==='
  df -h "$HOME" "$PREFIX" 2>&1
  echo '=== package state ==='
  dpkg --audit 2>&1
} | tee "$HOME/termux-diagnostics/environment.txt"
```

Classify the failure

Layer	Typical clues	First evidence to collect
Android	Process killed, storage permission missing, app cannot install	Android release, battery restrictions, free storage
Termux application	Add-on signature mismatch, bootstrap failure, terminal closes	Installation source, `termux-info`
Package manager	`Release file`, GPG, lock, dependency, mirror errors	Full `pkg update` output, source-list files
Shell and filesystem	`command not found`, `permission denied`, wrong path	`pwd`, `ls -la`, `file`, `head -n 1`
Language runtime	Import, syntax, package build, missing compiler	Interpreter path, package list, exact traceback
Network	DNS, timeout, certificate, proxy	`curl -Iv`, `getent hosts`, network type
Remote service	Authentication, rate limit, repository access	Remote URL, account method, service status

Reduce the problem

When a large project fails, prove that the underlying tool works independently. For example, replace a full Python application with `python -c 'print("ok")'`, replace an npm build with `node -e 'console.log("ok")'`, or replace a browser request with `curl -I URL`. A minimal test tells you whether the failure belongs to the project or to the environment.

2. Confirm a trustworthy installation

Termux and its add-ons must come from compatible sources. Builds signed by different distributors cannot be mixed because Android verifies application signatures. A common symptom is an add-on that installs but cannot communicate with Termux, or an update that fails with a signature conflict.

Check the current installation

```
termux-info
```

Look for the application source, package architecture, Android release, and installed add-ons. If the source is obsolete or unknown, plan a backup before changing it.

Source-consistency rule

Install Termux and every Termux add-on from the same distribution source. If changing sources, back up `$HOME` and `$PREFIX`, uninstall the complete Termux application family, then reinstall all components from one trusted source. Do not attempt to bypass Android signature checks.

Baseline health test

```
pkg update
pkg install termux-tools coreutils curl git python
command -v pkg apt bash curl git python
```

A healthy baseline should update package indexes, install without unresolved dependencies, and print absolute paths for all commands. If `pkg update` fails, solve repository access before troubleshooting Python or project scripts.