Android Phone as a Dedicated Server

Complete Setup Guide: Termux + Shizuku

Based on a Samsung Galaxy S22 Ultra setup, but applicable to most modern Android phones

Table of Contents

1. Prerequisites

What You Need

Item | Notes

Android phone | Android 11+ recommended. Flagship = more RAM/cores

USB cable | For initial ADB setup

Computer | Temporary - for wireless ADB pairing

WiFi | Stable connection for server use

Power source | Keep phone plugged in 24/7

Why This Setup?

The Problem: Termux runs as an "untrusted app" on Android, which means:

- Limited to ~3 CPU cores (Android throttles background apps)
- No GPU access (SELinux blocks /dev/dri/*)
- Can't access many system features

The Solution: Shizuku gives you shell user privileges, unlocking:

- All CPU cores (8 on our S22 Ultra)
- GPU compute access (Vulkan, OpenCL)
- System management commands (pm, dumpsys, etc.)

Our Reference Hardware

```
Device: Samsung Galaxy S22 Ultra (S908B)
CPU: Snapdragon 8 Gen 1 (8 cores)
RAM: 8GB (3.8GB available after optimization)
GPU: Samsung Xclipse 920 (Vulkan 1.1.179)
```

```
Storage: 106GB (85GB free)
Status: Cracked screen, repurposed as 24/7 server
```

2. Termux Setup

Installation

IMPORTANT: Install from F-Droid, NOT Google Play Store!

The Play Store version is outdated and broken.

- Install F-Droid: https://f-droid.org/
- Install Termux from F-Droid: https://f-droid.org/packages/com.termux/
- Install Termux:API (optional but useful): https://f-droid.org/packages/com.termux.api/

First Launch

Open Termux and run:

```
# Update package lists
pkg update
# Upgrade existing packages
pkg upgrade -y
# Install essential tools
pkg install -y git curl wget nano openssh jq
```

Understanding pkg

pkg is Termux's package manager (wrapper around apt):

```
# Search for packages
pkg search <name>
# Install packages
pkg install <package>
# Remove packages
pkg uninstall <package>
# List installed packages
pkg list-installed
# Show package info
pkg show <package>
```

Storage Setup

Grant Termux access to shared storage:

```
termux-setup-storage
```

This creates symlinks in ~/storage/:

- ~/storage/shared Internal storage
- ~/storage/downloads Downloads folder
- ~/storage/dcim Camera photos

Keeping Termux Alive

Android aggressively kills background apps. Prevent this:

- Disable battery optimization for Termux:
  - Settings Apps Termux Battery Unrestricted

- Acquire a wake lock (in Termux):
`bash
termux-wake-lock
`
- Consider running a foreground service:
`bash
Install termux-services
pkg install termux-services
Enable sshd to keep Termux active
sv-enable sshd
`

SSH Access (Recommended)

Set up SSH so you can manage remotely:

```
# Install OpenSSH
pkg install openssh
# Set a password
passwd
# Start SSH server
sshd
# Find your IP
ip addr show wlan0 | grep inet
```

SSH runs on port 8022 by default:

```
# From another machine
ssh -p 8022 user@<phone-ip>
```

3. Shizuku Setup

What is Shizuku?

Shizuku is a clever app that:

- Runs an ADB shell process on your phone
- Lets other apps borrow those shell privileges
- Works WITHOUT root

This gives us access to the shell user (uid 2000), which has far more privileges than regular apps.

Installation

- Install Shizuku from Play Store or F-Droid:
  - https://shizuku.rikka.app/
  - Or: https://play.google.com/store/apps/details?id=moe.shizuku.privileged.api
- Open Shizuku app

Starting Shizuku (Wireless ADB Method)

This is the recommended method - survives reboots better than USB.

Step 1: Enable Developer Options

- Settings About Phone Software Information
- Tap "Build Number" 7 times
- Enter PIN if prompted
- "Developer mode enabled" appears

Step 2: Enable Wireless Debugging

- Settings Developer Options
- Find "Wireless debugging" and enable it
- Tap on "Wireless debugging" to enter its settings

Step 3: Pair with Shizuku

Option A: From the phone itself (easiest)

- In Wireless Debugging settings, tap "Pair device with pairing code"
- Note the pairing code and port (e.g., 123456 on port 37521)
- Open Shizuku app
- Tap "Start via Wireless debugging"
- Enter the pairing code when prompted
- Shizuku will connect and start

Option B: From a computer (if Option A fails)

```
# On your computer, with ADB installed
adb pair <phone-ip>:<pairing-port>
# Enter pairing code when prompted
# Then connect
adb connect <phone-ip>:<debug-port>
# Start Shizuku
adb shell sh /storage/emulated/0/Android/data/moe.shizuku.privileged.api/start.sh
```

Step 4: Verify Shizuku is Running

In the Shizuku app, you should see:

- "Shizuku is running"
- Service version number
- Number of authorized apps

Auto-Start on Boot

Shizuku can auto-start if:

- Wireless debugging stays enabled
- You're on the same WiFi network

In Shizuku settings:

- Enable "Start automatically when wireless debugging is enabled"

Granting Termux Access

- Open Shizuku
- Look for Termux in the list of apps
- Tap to authorize
- If Termux doesn't appear, you need to install rish first (next section)

4. rish Integration

What is rish?

rish (Remote Interactive SHell) is a tool that lets you run commands with Shizuku's shell privileges from within Termux.

Installing rish

```
# Navigate to Termux bin
cd $PREFIX/bin
# Download rish and the DEX file
# Method 1: From Shizuku releases
curl -L -o rish "https://github.com/ArcPart/rish/releases/download/v1.0.0/rish-termux"
curl -L -o rish_shizuku.dex "https://github.com/ArcPart/rish/releases/download/v1.0.0/rish_shizuku.dex"
# Make executable
chmod +x rish
```

Alternative: Manual extraction from Shizuku APK

```
# If the above doesn't work, extract from Shizuku APK
pkg install unzip
# Find Shizuku APK (path varies)
APK_PATH=$(pm path moe.shizuku.privileged.api 2>/dev/null | cut -d: -f2)
# If that doesn't work, try:
APK_PATH="/data/app/~~*/moe.shizuku.privileged.api*/base.apk"
# Extract the DEX
unzip -p $APK_PATH classes.dex > $PREFIX/bin/rish_shizuku.dex
# The rish binary needs to be obtained separately or written
```

Testing rish

```
# Test basic command
echo 'id' | rish
# Expected output:
# uid=2000(shell) gid=2000(shell) groups=...
```

If you see uid=2000(shell), it's working!

The rish-run Helper Script

Create a helper script that makes running binaries through Shizuku easy:

```
cat > $PREFIX/bin/rish-run << 'EOF'
#!/data/data/com.termux/files/usr/bin/bash
#
# rish-run: Run commands/binaries with full hardware access via Shizuku
#
# Usage:
#   rish-run ./binary           # Run a binary
#   rish-run -c "command"       # Run a shell command
#
set -e
SHARED_DIR="/data/local/tmp"
if [ "$1" = "-c" ]; then
    # Run a command directly
    shift
    echo "$*" | rish
    exit $?
fi
if [ -z "$1" ]; then
```

```
    echo "Usage: rish-run ./binary [args]"
    echo "       rish-run -c \"command\""
    exit 1
fi
BINARY="$1"
shift
if [ ! -f "$BINARY" ]; then
    echo "Error: File not found: $BINARY"
    exit 1
fi
# Get absolute path and filename
BINARY_PATH=$(realpath "$BINARY")
BINARY_NAME=$(basename "$BINARY")
DEST="$SHARED_DIR/$BINARY_NAME"
# Copy to shared location and execute
echo "cp \"$BINARY_PATH\" \"$DEST\" && chmod 755 \"$DEST\" && \"$DEST\" $*" | rish
# Cleanup
echo "rm -f \"$DEST\"" | rish 2>/dev/null || true
EOF
chmod +x $PREFIX/bin/rish-run
```

Usage Examples

```
# Check all CPU cores are visible
rish-run -c "cat /proc/cpuinfo | grep processor | wc -l"
# Output: 8 (instead of 3)
# Check GPU access
rish-run -c "ls -la /dev/dri/"
# Should show renderD128, renderD129, etc.
# Run a compiled binary with full hardware
rish-run ./my_benchmark
# Compile in Termux, run via Shizuku
gcc -O3 -o test test.c
rish-run ./test
```

What Shizuku Unlocks (Before/After)

Resource | Normal Termux | Via Shizuku

CPU Cores | 3 | 8

GPU Access | Blocked | Full

CPU Frequencies | Hidden | Visible

System Info | Limited | Full dumpsys

Package Management | Limited | Full pm commands

5. Database Stack

SQLite (Built-in, Simple)

SQLite is perfect for local data storage.

```
# Install
pkg install sqlite
# Create/open database
sqlite3 mydata.db
# Basic commands
sqlite> CREATE TABLE trades (id INTEGER PRIMARY KEY, symbol TEXT, price REAL, timestamp INTEGER);
sqlite> INSERT INTO trades VALUES (1, 'BTC', 50000.0, 1640000000);
```

```
sqlite> SELECT * FROM trades;
sqlite> .quit
```

Limitations: Single-writer, no network access, file-based.

Redis (In-Memory, Fast)

Redis is excellent for caching, real-time data, and pub/sub.

```
# Install
pkg install redis
# Start Redis server
redis-server --daemonize yes
# Test connection
redis-cli ping
# Output: PONG
# Basic usage
redis-cli SET mykey "hello"
redis-cli GET mykey
```

Redis Configuration (optional):

```
# Create config file
mkdir -p ~/.config/redis
cat > ~/.config/redis/redis.conf << 'EOF'
daemonize yes
port 6379
bind 127.0.0.1
maxmemory 512mb
maxmemory-policy allkeys-lru
appendonly yes
appendfilename "appendonly.aof"
dir /data/data/com.termux/files/home/.local/share/redis
EOF
# Create data directory
mkdir -p ~/.local/share/redis
# Start with config
redis-server ~/.config/redis/redis.conf
```

PostgreSQL (via proot-distro)

PostgreSQL doesn't run natively in Termux, but works perfectly in a Linux container.

Installing proot-distro

```
# Install proot-distro
pkg install proot-distro
# List available distros
proot-distro list
# Install Alpine Linux (lightweight, ~10MB)
proot-distro install alpine
```

Setting Up PostgreSQL in Alpine

```
# Enter Alpine
proot-distro login alpine
# Inside Alpine: Install PostgreSQL
apk update
apk add postgresql postgresql-contrib
# Initialize database
mkdir -p /var/lib/postgresql/data
mkdir -p /run/postgresql
```

```
chown -R postgres:postgres /var/lib/postgresql /run/postgresql
# Initialize as postgres user
su postgres -c "initdb -D /var/lib/postgresql/data"
# Start PostgreSQL
su postgres -c "pg_ctl -D /var/lib/postgresql/data start"
# Create a database
su postgres -c "createdb trading"
# Connect
su postgres -c "psql -d trading"
# Exit Alpine
exit
```

Connecting from Termux

```
# Run command in Alpine
proot-distro login alpine -- su postgres -c "psql -d trading -c 'SELECT version();'"
```

Helper Script for PostgreSQL

```
cat > $PREFIX/bin/pg << 'EOF'
#!/bin/bash
# PostgreSQL helper for proot-distro Alpine
ACTION="${1:-shell}"
case "$ACTION" in
    start)
        proot-distro login alpine -- sh -c "
            mkdir -p /run/postgresql
            chown postgres:postgres /run/postgresql
            su postgres -c 'pg_ctl -D /var/lib/postgresql/data start'
        "
        ;;
    stop)
        proot-distro login alpine -- su postgres -c "pg_ctl -D /var/lib/postgresql/data stop"
        ;;
    status)
        proot-distro login alpine -- su postgres -c "pg_ctl -D /var/lib/postgresql/data status"
        ;;
    shell)
        proot-distro login alpine -- su postgres -c "psql -d trading"
        ;;
    exec)
        shift
        proot-distro login alpine -- su postgres -c "psql -d trading -c \"$*\""
        ;;
    *)
        echo "Usage: pg [start|stop|status|shell|exec <SQL>]"
        ;;
esac
EOF
chmod +x $PREFIX/bin/pg
```

Usage:

```
pg start      # Start PostgreSQL
pg stop       # Stop PostgreSQL
pg shell      # Interactive psql
pg exec "SELECT * FROM trades LIMIT 5"
```

TimescaleDB (Time-Series Extension)

TimescaleDB is perfect for trading data, metrics, and time-series workloads.

```
# Enter Alpine
```

```
proot-distro login alpine
# Add TimescaleDB repository
echo "https://dl-cdn.alpinelinux.org/alpine/edge/community" >> /etc/apk/repositories
apk update
# Install TimescaleDB
apk add timescaledb
# Enable the extension
su postgres -c "psql -d trading -c 'CREATE EXTENSION IF NOT EXISTS timescaledb;'"
# Create a hypertable for trades
su postgres -c "psql -d trading" << 'EOF'
CREATE TABLE IF NOT EXISTS trades (
    time        TIMESTAMPTZ NOT NULL,
    symbol      TEXT NOT NULL,
    price       DOUBLE PRECISION,
    volume      DOUBLE PRECISION,
    side        TEXT
);
SELECT create_hypertable('trades', 'time', if_not_exists => TRUE);
-- Create indexes
CREATE INDEX IF NOT EXISTS idx_trades_symbol_time ON trades (symbol, time DESC);
-- Enable compression (optional, saves space)
ALTER TABLE trades SET (
    timescaledb.compress,
    timescaledb.compress_segmentby = 'symbol'
);
EOF
exit
```

Example Queries:

```
-- Insert trade
INSERT INTO trades VALUES (NOW(), 'BTC/USD', 50000.0, 1.5, 'buy');
-- Get OHLCV candles
SELECT
    time_bucket('1 minute', time) AS bucket,
    symbol,
    first(price, time) AS open,
    max(price) AS high,
    min(price) AS low,
    last(price, time) AS close,
    sum(volume) AS volume
FROM trades
WHERE symbol = 'BTC/USD'
  AND time > NOW() - INTERVAL '1 hour'
GROUP BY bucket, symbol
ORDER BY bucket;
```

6. GPU Compute

Understanding Android GPU Access

On Android:

- GPU hardware exists at /dev/dri/renderD128, /dev/dri/renderD129
- Regular apps (including Termux) are blocked by SELinux
- The shell user (via Shizuku) CAN access GPU

Checking GPU Access

```
# Without Shizuku (will fail)
ls -la /dev/dri/
# Permission denied
```

```
# With Shizuku (works!)
rish-run -c "ls -la /dev/dri/"
# crw-rw---- 1 system graphics ... renderD128
# crw-rw---- 1 system graphics ... renderD129
```

Vulkan Compute (Recommended)

Vulkan is the fastest option for GPU compute on Android.

```
# Install Vulkan tools in Termux
pkg install vulkan-tools vulkan-headers shaderc
# Check Vulkan info (via Shizuku)
rish-run -c "vulkaninfo --summary 2>/dev/null | head -30"
```

Sample Vulkan Compute Program:

```
// save as: gpu_compute.c
// Minimal Vulkan compute example
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dlfcn.h>
// Vulkan types (minimal)
typedef uint32_t VkFlags;
typedef uint32_t VkBool32;
typedef uint64_t VkDeviceSize;
typedef struct VkInstance_T* VkInstance;
typedef struct VkPhysicalDevice_T* VkPhysicalDevice;
typedef struct VkDevice_T* VkDevice;
// ... (full implementation would be longer)
int main() {
    printf("Vulkan GPU Compute Test\n");

    // Load Vulkan library
    void* vulkan = dlopen("libvulkan.so", RTLD_NOW);
    if (!vulkan) {
        printf("Failed to load Vulkan: %s\n", dlerror());
        return 1;
    }

    printf("Vulkan library loaded successfully!\n");

    // In a real implementation, you would:
    // 1. Create VkInstance
    // 2. Select VkPhysicalDevice
    // 3. Create VkDevice with compute queue
    // 4. Create shader module from SPIR-V
    // 5. Create compute pipeline
    // 6. Allocate buffers
    // 7. Record and submit command buffer
    // 8. Read results

    dlclose(vulkan);
    return 0;
}
```

```
# Compile
clang -O3 -o gpu_compute gpu_compute.c -ldl
# Run via Shizuku (for GPU access)
rish-run ./gpu_compute
```

OpenCL (Native Samsung/AMD)

Samsung phones with AMD GPUs (like S22 Ultra) have native OpenCL:

```
# Check for OpenCL library
```

```
rish-run -c "ls -la /vendor/lib64/*OpenCL* 2>/dev/null"
# /vendor/lib64/libSGPUOpenCL.so
# The GPU reports as AMD gfx1040 (RDNA2 architecture)
```

OpenCL Kernel Example:

```
// opencl_kernel.cl
__kernel void vector_add(__global const float* a,
                         __global const float* b,
                         __global float* result) {
    int i = get_global_id(0);
    result[i] = a[i] + b[i];
}
```

ML Inference (NNAPI)

Android's Neural Networks API can use GPU/NPU:

```
# Using ONNX Runtime with NNAPI
import onnxruntime as ort
# Create session with NNAPI provider (uses GPU/NPU)
session = ort.InferenceSession(
    "model.onnx",
    providers=['NnapiExecutionProvider', 'CPUExecutionProvider']
)
# Run inference
result = session.run(None, {"input": input_data})
```

GPU Benchmark Results (S22 Ultra)

Test | Result

Vulkan compute (1M float adds) | 1.33 ms

OpenCL compute (1M float adds) | 3.95 ms

Memory bandwidth | 5.2 GB/s

7. Python/Node Stack

Python Setup

```
# Install Python
pkg install python python-pip
# Upgrade pip
pip install --upgrade pip
# Install common packages
pip install numpy requests aiohttp
# For data science (may take a while)
pip install pandas scipy matplotlib
```

NumPy Verification:

```
python3 -c "
import numpy as np
print(f'NumPy version: {np.__version__}')
# Quick benchmark
import time
a = np.random.rand(1000000)
b = np.random.rand(1000000)
start = time.time()
c = a + b
print(f'1M float adds: {(time.time()-start)*1000:.2f}ms')
```

```
"
```

Async PostgreSQL with Python

```
# Install asyncpg (works in Alpine via proot)
proot-distro login alpine -- sh -c "
    apk add python3 py3-pip
    pip install asyncpg
"
```

```
# async_db.py - Run inside Alpine
import asyncio
import asyncpg
async def main():
    conn = await asyncpg.connect(
        user='postgres',
        database='trading',
        host='/run/postgresql'  # Unix socket
    )

    result = await conn.fetch('SELECT * FROM trades LIMIT 5')
    for row in result:
        print(row)

    await conn.close()
asyncio.run(main())
```

Node.js Setup

```
# Install Node.js
pkg install nodejs-lts
# Or latest version
pkg install nodejs
# Verify
node --version
npm --version
# Initialize a project
mkdir ~/myproject && cd ~/myproject
npm init -y
# Install packages
npm install express redis pg
```

Example Express Server:

```
// server.js
const express = require('express');
const app = express();
app.get('/', (req, res) => {
    res.json({
        status: 'running',
        platform: 'Android/Termux',
        timestamp: new Date().toISOString()
    });
});
app.listen(3000, '0.0.0.0', () => {
    console.log('Server running on http://0.0.0.0:3000');
});
```

```
# Run
node server.js
# Test from another device
curl http://<phone-ip>:3000/
```

Development Workflow

```
# Install development tools
pkg install git vim tmux
# tmux for persistent sessions
tmux new -s dev
# Split panes for:
# - Code editing
# - Running server
# - Logs/testing
# Detach: Ctrl+B, D
# Reattach: tmux attach -t dev
```

8. Bloatware Management

Why Disable Bloatware?

Preinstalled apps consume RAM even when "not running":

- Background services
- Receivers listening for broadcasts
- Cached processes

On our S22 Ultra, disabling ~100 packages freed significant RAM now at 3.4GB available (from 7.1GB total).

Listing Installed Packages

```
# List all packages
rish-run -c "pm list packages" | wc -l
# List third-party packages
rish-run -c "pm list packages -3"
# List system packages
rish-run -c "pm list packages -s"
# List disabled packages
rish-run -c "pm list packages -d"
```

Safe Packages to Disable

WARNING: Disabling wrong packages can soft-brick your phone!

Generally Safe to Disable:

```
# Facebook
echo 'pm disable-user --user 0 com.facebook.katana' | rish
echo 'pm disable-user --user 0 com.facebook.appmanager' | rish
echo 'pm disable-user --user 0 com.facebook.services' | rish
# Microsoft
echo 'pm disable-user --user 0 com.microsoft.skydrive' | rish
echo 'pm disable-user --user 0 com.microsoft.office.outlook' | rish
# Google (be careful!)
echo 'pm disable-user --user 0 com.google.android.apps.magazines' | rish
echo 'pm disable-user --user 0 com.google.android.apps.tachyon' | rish  # Duo
echo 'pm disable-user --user 0 com.google.android.videos' | rish
echo 'pm disable-user --user 0 com.google.android.music' | rish
# Samsung
echo 'pm disable-user --user 0 com.samsung.android.game.gamehome' | rish
echo 'pm disable-user --user 0 com.samsung.android.game.gametools' | rish
echo 'pm disable-user --user 0 com.samsung.android.app.tips' | rish
echo 'pm disable-user --user 0 com.samsung.android.voc' | rish  # Samsung Members
echo 'pm disable-user --user 0 com.samsung.android.mobileservice' | rish
```

Disable Script

```
cat > $PREFIX/bin/disable-bloat << 'EOF'
#!/bin/bash
# Disable common bloatware packages
PACKAGES=(
    # === GOOGLE (aggressive - keep GMS if you need Play Store) ===
    "com.google.android.gms"                    # Play Services - big RAM hog
    "com.google.android.gms.location.history"
    "com.google.android.gms.supervision"
    "com.google.android.vending"                # Play Store
    "com.google.android.apps.maps"
    "com.google.android.apps.messaging"
    "com.google.android.apps.tachyon"           # Duo/Meet
    "com.google.android.apps.turbo"             # Device Health
    "com.google.android.gm"                     # Gmail
    "com.google.android.youtube"
    "com.google.android.googlequicksearchbox"   # Google Search/Assistant
    "com.google.android.inputmethod.latin"      # Gboard
    "com.google.android.tts"
    "com.google.android.as"
    "com.google.android.as.oss"
    "com.google.android.ims"
    "com.google.android.feedback"
    "com.google.android.documentsui"
    "com.google.android.ext.services"
    "com.google.android.printservice.recommendation"
    "com.google.android.projection.gearhead"    # Android Auto
    "com.google.android.syncadapters.calendar"
    "com.google.ar.core"
    "com.google.audio.hearing.visualization.accessibility.scribe"
    "com.google.android.apps.accessibility.voiceaccess"
    "com.google.android.adservices.api"
    "com.google.android.federatedcompute"
    "com.google.android.ondevicepersonalization.services"
    "com.google.mainline.adservices"
    "com.google.mainline.telemetry"
    # === FACEBOOK ===
    "com.facebook.appmanager"
    "com.facebook.services"
    # === MICROSOFT ===
    "com.microsoft.skydrive"
    "com.microsoft.appmanager"
    # === SAMSUNG BLOAT ===
    "com.samsung.android.mobileservice"
    "com.samsung.android.bixby.agent"
    "com.samsung.android.app.settings.bixby"
    "com.samsung.android.visionintelligence"
    "com.samsung.android.arzone"
    "com.samsung.android.app.spage"             # Samsung Free
    "com.samsung.android.app.notes"
    "com.samsung.android.app.contacts"
    "com.samsung.android.app.routines"
    "com.samsung.android.app.sharelive"
    "com.samsung.android.app.galaxyfinder"
    "com.samsung.android.app.telephonyui"
    "com.samsung.android.app.aodservice"
    "com.samsung.android.game.gos"              # Game Optimizer
    "com.samsung.android.messaging"
    "com.samsung.android.honeyboard"            # Samsung Keyboard
    "com.samsung.android.themestore"
    "com.samsung.android.stickercenter"
    "com.samsung.android.forest"                # Digital Wellbeing
    "com.samsung.android.lool"                  # Samsung Members
    "com.samsung.android.scloud"                # Samsung Cloud
    "com.samsung.android.samsungpass"
```

```
    "com.samsung.android.smartcallprovider"
    "com.samsung.android.smartface"
    "com.samsung.android.smartsuggestions"
    "com.samsung.android.rubin.app"
    "com.samsung.android.mdx"
    "com.samsung.android.mcfds"
    "com.samsung.android.da.daagent"
    "com.samsung.android.dsms"
    "com.samsung.android.bbc.bbcagent"
    "com.samsung.android.beaconmanager"
    "com.samsung.android.cmfa.framework"
    "com.samsung.android.carkey"
    "com.samsung.android.dkey"
    "com.samsung.android.uwb"
    "com.samsung.android.cameraxservice"
    "com.samsung.android.peripheral.framework"
    "com.samsung.android.scpm"
    "com.samsung.android.scs"
    "com.samsung.android.server.wifi.mobilewips"
    "com.samsung.android.service.aircommand"
    "com.samsung.android.service.airviewdictionary"
    "com.samsung.android.service.pentastic"
    "com.samsung.android.shortcutbackupservice"
    "com.samsung.SMT"
    "com.samsung.accessibility"
    "com.samsung.android.accessibility.talkback"
    "com.samsung.faceservice"
    "com.samsung.ipservice"
    "com.samsung.oda.service"
    "com.samsung.sec.android.teegris.tui_service"
    "com.samsung.cmfa.AuthTouch"
    # === SYSTEM (careful!) ===
    "com.android.bluetooth"                    # Only if not using BT
    "com.android.chrome"
    "com.android.nfc"                          # Only if not using NFC
    "com.android.phone"                        # Only if pure server (no calls!)
    "com.android.settings.intelligence"
    "com.osp.app.signin"                       # Samsung Account
    "com.sec.android.app.launcher"             # Samsung Home (use AOSP)
    "com.sec.android.app.chromecustomizations"
    "com.sec.android.daemonapp"
    "com.sec.android.desktopmode.uiservice"    # DeX
    "com.sec.android.diagmonagent"
    "com.sec.android.sdhms"
    "com.sec.bcservice"
    "com.sec.imsservice"
    "com.sec.location.nsflp2"
    "com.sec.phone"
    "com.sec.sve"
    "com.sec.unifiedwfc"
    "com.wssyncmldm"
)
for pkg in "${PACKAGES[@]}"; do
    echo "Disabling: $pkg"
    echo "pm disable-user --user 0 $pkg" | rish 2>/dev/null
done
echo "Done! Disabled ${#PACKAGES[@]} packages"
echo "Reboot recommended"
EOF
chmod +x $PREFIX/bin/disable-bloat
```

Re-enabling Packages

If something breaks:

```
# Re-enable a specific package
echo 'pm enable com.package.name' | rish
# Re-enable ALL disabled packages
for pkg in $(pm list packages -d | cut -d: -f2); do
    echo "pm enable $pkg" | rish
done
```

Checking Memory After Cleanup

```
# Before/after comparison
cat /proc/meminfo | grep -E "MemTotal|MemAvailable"
# Or via Termux API
termux-battery-status
```

9. Clawdbot Integration (Optional)

What is Clawdbot?

Clawdbot is an AI agent platform that runs Claude (Anthropic's AI) locally with:

- Tool use (file system, web, shell commands)
- Memory and context management
- Discord/messaging integration

Prerequisites

- Node.js 18+
- npm
- Anthropic API key

Installation

```
# Install Clawdbot globally
npm install -g clawdbot
# Set up API key
export ANTHROPIC_API_KEY="your-key-here"
# Add to shell profile
echo 'export ANTHROPIC_API_KEY="your-key-here"' >> ~/.bashrc
```

Termux-Specific Patches

Clawdbot may have hardcoded paths that don't work in Termux:

```
# Check for hardcoded paths
grep -rn "'/tmp\|'/home\|\"/tmp\|\"/home" $PREFIX/lib/node_modules/clawdbot/dist/
# Patch /tmp  Termux temp
# Patch /home  /data/data/com.termux/files/home
```

Running Clawdbot

```
# Start interactively
clawdbot
# Or with specific config
clawdbot --config ~/.config/clawdbot/config.yaml
```

Sharp (Image Processing)

Clawdbot uses sharp for images. The native build may fail:

```
# If native build OOMs, use WASM fallback
cd $PREFIX/lib/node_modules/clawdbot
npm install sharp --build-from-source=false
```

Discord Integration

```
# Set up Discord bot token
export DISCORD_BOT_TOKEN="your-token"
# Configure channels in Clawdbot config
```

10. Troubleshooting

Shizuku Won't Start

Problem: Shizuku shows "not running"

Solutions:

- Check wireless debugging is enabled
- Re-pair the device:
`bash
On computer
adb pair <ip>:<port>
adb connect <ip>:<port>
adb shell sh /storage/emulated/0/Android/data/moe.shizuku.privileged.api/start.sh
`
- Check if another ADB session is blocking:
`bash
adb devices # Should show your device
`

rish: Permission Denied

Problem: rish commands fail with permission errors

Solutions:

- Ensure Shizuku is running (check Shizuku app)
- Authorize Termux in Shizuku
- Check the DEX file exists:
`bash
ls -la $PREFIX/bin/rish_shizuku.dex
`

PostgreSQL Won't Start

Problem: pg_ctl fails in Alpine

Solutions:

```
# Check if already running
proot-distro login alpine -- su postgres -c "pg_ctl status -D /var/lib/postgresql/data"
# Check data directory permissions
proot-distro login alpine -- ls -la /var/lib/postgresql/
# Reinitialize if corrupted
proot-distro login alpine -- sh -c "
    rm -rf /var/lib/postgresql/data/*
    su postgres -c 'initdb -D /var/lib/postgresql/data'
"
```

Termux Keeps Getting Killed

Problem: Android kills Termux after a few minutes

Solutions:

- Disable battery optimization
- Acquire wake lock: termux-wake-lock
- Run a foreground notification service
- Disable "Adaptive battery"
- On Samsung: Disable "Put unused apps to sleep" in Device Care

Out of Memory (OOM) During Compilation

Problem: Compiling large packages fails with OOM

Solutions:

- Disable more bloatware to free RAM
- Add swap (if rooted)
- Compile on another machine and copy binary
- Use pre-built packages when available

GPU Access Still Blocked

Problem: Even with Shizuku, GPU commands fail

Solutions:

- Verify shell SELinux context:
`bash
rish-run -c "cat /proc/self/attr/current"
Should show: u:r:shell:s0
`
- Check device permissions:
`bash
rish-run -c "ls -la /dev/dri/"
`
- Some devices may need additional vendor-specific workarounds

Networking Issues

Problem: Can't connect to services from other devices

Solutions:

- Check phone IP: ip addr show wlan0
- Verify service is binding to 0.0.0.0 (not just localhost)
- Check Android firewall settings

- Some routers block device-to-device traffic

Quick Reference Card

```
# Shizuku commands
echo 'id' | rish                    # Test Shizuku
rish-run ./binary                   # Run binary with full hardware
rish-run -c "command"               # Run command via Shizuku
# Database
redis-server --daemonize yes        # Start Redis
pg start                            # Start PostgreSQL (helper)
pg shell                            # PostgreSQL shell
# System info
rish-run -c "cat /proc/cpuinfo | grep processor | wc -l"  # CPU cores
rish-run -c "cat /proc/meminfo | grep MemAvailable"       # Free RAM
rish-run -c "ls /dev/dri/"          # GPU devices
# Bloatware
rish-run -c "pm list packages -d"   # List disabled
echo 'pm disable-user --user 0 com.package' | rish  # Disable
echo 'pm enable com.package' | rish                  # Re-enable
# Termux
termux-wake-lock                    # Prevent sleep
termux-setup-storage                # Enable storage access
sshd                                # Start SSH server (port 8022)
```

Appendix: Our S22 Ultra Performance

Metric | Value

CPU cores (via Shizuku) | 8

Available RAM (after optimization) | 3.8 GB

Vulkan compute (1M adds) | 1.33 ms

OpenCL compute (1M adds) | 3.95 ms

AES-256-CBC | 491 MB/s

Disk write (synced) | 123 MB/s

Disk read | 2.3 GB/s

API latency (Anthropic) | ~117ms connect, ~477ms total

Guide last updated: January 2026

Based on real-world experience with Samsung Galaxy S22 Ultra running Android 12