Assignment 1: ADB Connection & Shell Access

Objective

To understand how Android Debug Bridge (ADB) works and how to interact with an Android device/emulator using shell commands.

Tools Used

- Android Device / Emulator (Genymotion)
- ADB (Android Debug Bridge)
- Kali Linux (Linux System)

Tasks Performed

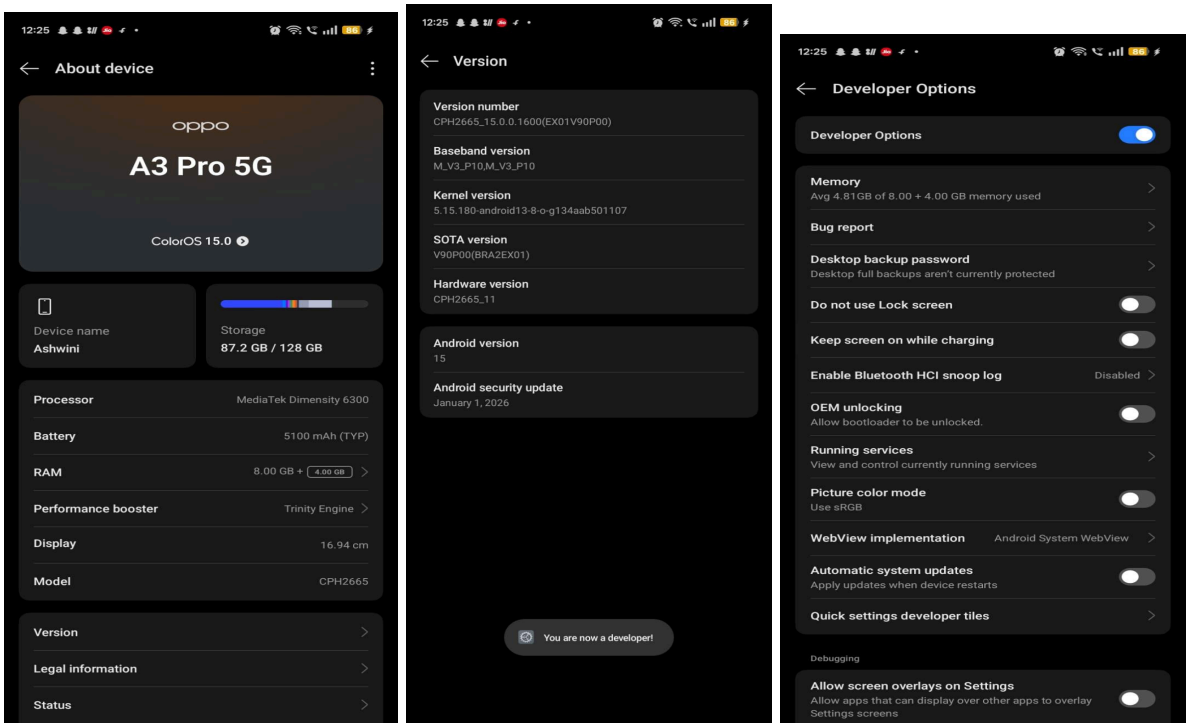
1. Enabled **Developer Options** on the Android device.
2. Enabled **USB Debugging**.
3. Connected the Android emulator (Genymotion) to the Kali Linux system.
4. Verified device connection using ADB.
5. Obtained ADB shell access.
6. Executed basic Linux commands inside the shell.

Steps:

Step 1: Enable Developer Options
Open **Settings** on the Android device/emulator.
Go to **About Device / About Phone**.
Locate **Build Number / Version**.

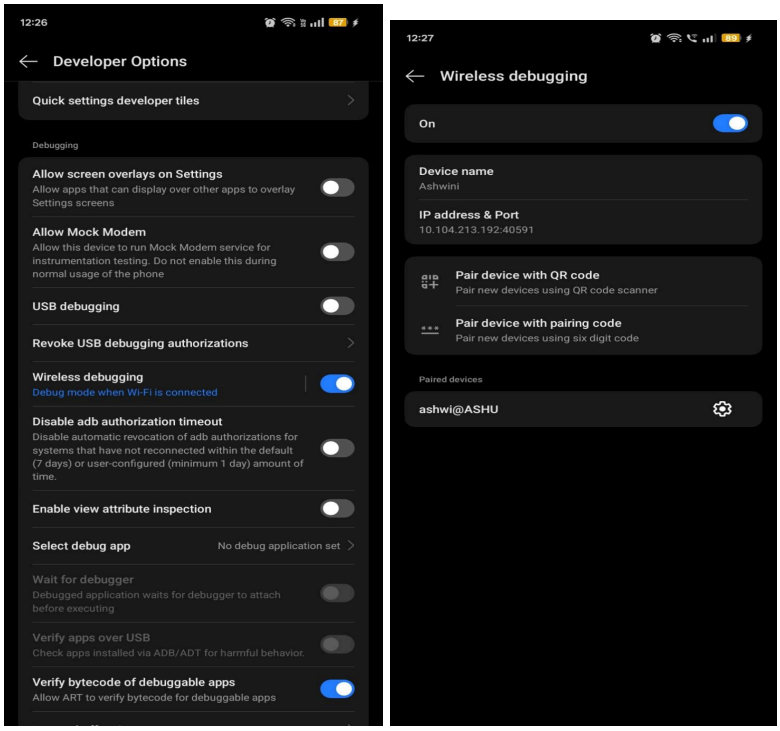
Tap on it **5 times continuously** until the message *“You are now a developer”* appears.

Developer Options will now be enabled.



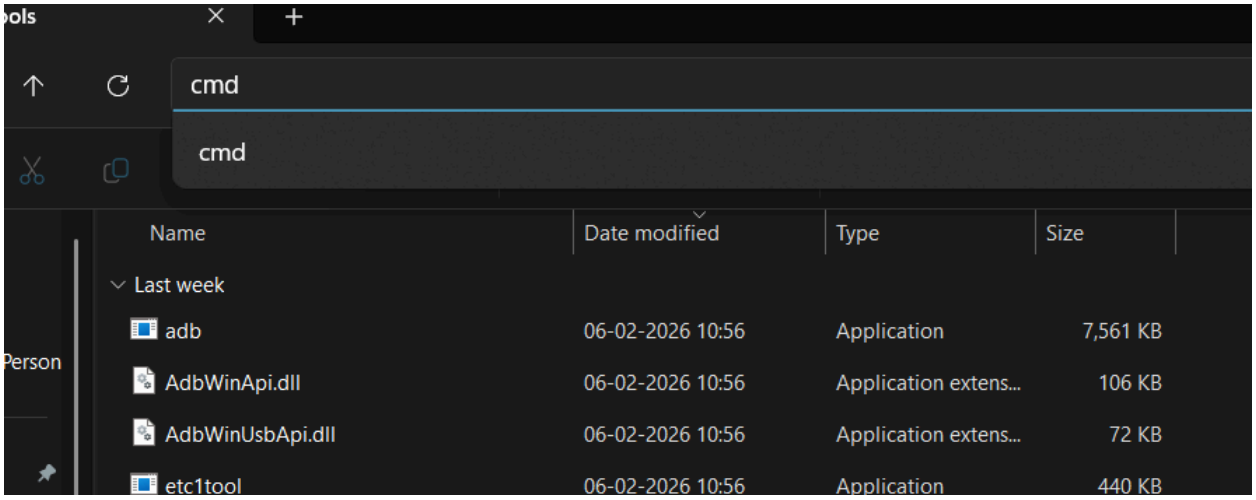
Step 2: Enable Wireless Debugging

- Go to **Settings** → **Developer Options**.
- Enable **Wireless Debugging**.
- Open **Wireless Debugging** and select **Pair device with pairing code**.
- Note down the following details shown on the screen:
  - **IP Address**
  - **Port Number**
  - **Pairing Code**



Step 3: Open ADB Platform Tools

- Open the folder where **ADB (platform-tools)** is installed.
- Click on the **address bar**, type `cmd`, and press **Enter**.
- Command Prompt will open in the ADB directory.



Step 4: Pair ADB with Android Device

Run the following command in CMD:

Command: adb pair ip:port

```
C:\Users\ashwi\Downloads\platform-tools-latest-windows\platform-tools>adb pair 10.104.213.192:45115
Enter pairing code: 696716
Successfully paired to 10.104.213.192:45115 [guid=adb-KVYHA6S8EAEEAASS-rXD796]
```

Enter the **pairing code** when prompted.
Message **“Successfully paired”** will be displayed.

Step 5: Verify Device Connection

- Check whether the device is connected using:

```
C:\Users\ashwi\Downloads\platform-tools-latest-windows\platform-tools>adb devices
List of devices attached
adb-KVYHA6S8EAEEAASS-rXD796._adb-tls-connect._tcp       device
```

the device name or IP address will appear in the list, confirming a successful connection.

Step 6: Access ADB Shell

- Enter the Android shell using:

```
C:\Users\ashwi\Downloads\platform-tools-latest-windows\platform-tools>adb shell
OP5B05L1:/ $ ls
acct            config         etc                   mnt            my_manifest  odm          sdcard                 system_dlkm
apex            d              init                  my_bigball     my_preload   odm_dlkm     second_stage_resources  system_ext
bin             data           init.environ.rc       my_carrier     my_product   oem          special_preload         tmp
bootstrap-apex  data_mirror    init.environ.rc.patch  my_company     my_region    postinstall  storage                 vendor
bugreports      debug_ramdisk  linkerconfig          my_engineering  my_reserve   proc         sys                     vendor_dlkm
cache           dev            metadata              my_heytap      my_stock     product      system
OP5B05L1:/ $
```

You are now inside the Android device shell environment.

Step 7: Execute Shell Commands

Run the following commands inside the ADB shell:

```
/system/bin/sh: sdcard: inaccessible or not found
127|OP5B05L1:/ $ cd sdcard
```

➡ Displays the current user (usually shell)

```
OP5B05L1:/ $ pwd
/
```

➡ Shows the present working directory

Showing ip address of our device:

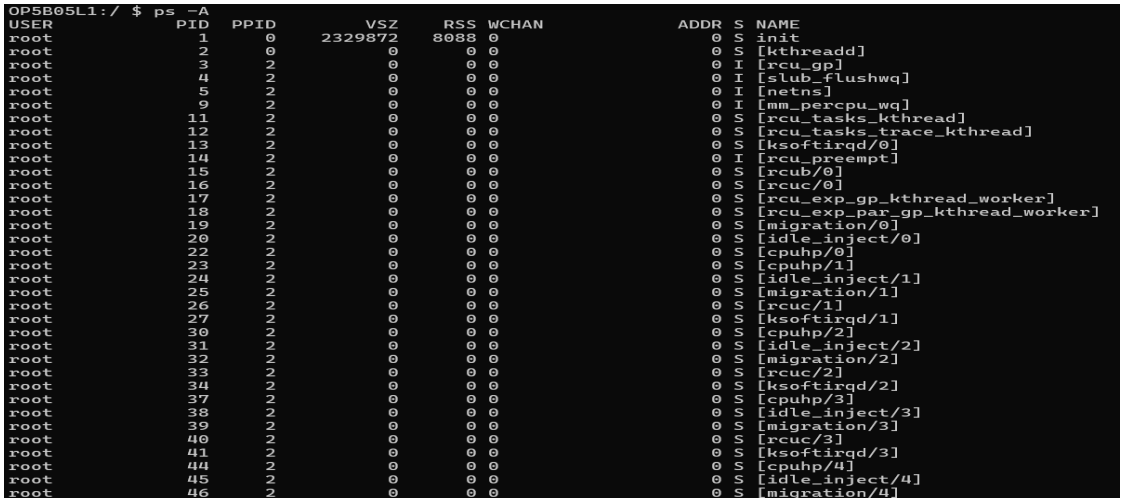
```
OP5B05L1:/ $ ifconfig
lo        Link encap:UNSPEC
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope: Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:33081 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33081 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4927839 TX bytes:4927839

dummy0    Link encap:UNSPEC
          inet6 addr: fe80::34c2:78ff:fec9:61cc/64 Scope: Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49378 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 TX bytes:7698805

ifb0      Link encap:UNSPEC
          inet6 addr: fe80::d8c3:b8ff:fe69:2b1a/64 Scope: Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:51324 errors:0 dropped:49332 overruns:0 frame:0
          TX packets:1992 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9959865 TX bytes:2269885

ifb1      Link encap:UNSPEC
          inet6 addr: fe80::cce3:b8ff:fe84:8a80/64 Scope: Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:54890 errors:0 dropped:48976 overruns:0 frame:0
          TX packets:5914 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14649681 TX bytes:6990085
```

For showing ps-A



```
OP5B05L1:/ $ ps -A
USER          PID  PPID     VSZ    RSS WCHAN              ADDR S NAME
root            1     0 2329872   8088 0                    0 S init
root            2     0       0      0 0                    0 S [kthreadd]
root            3     2       0      0 0                    0 I [rcu_gp]
root            4     2       0      0 0                    0 I [slub_flushwq]
root            5     2       0      0 0                    0 I [netns]
root            9     2       0      0 0                    0 I [mm_percpu_wq]
root           11     2       0      0 0                    0 S [rcu_tasks_kthread]
root           12     2       0      0 0                    0 S [rcu_tasks_trace_kthread]
root           13     2       0      0 0                    0 S [ksoftirqd/0]
root           14     2       0      0 0                    0 I [rcu_preempt]
root           15     2       0      0 0                    0 S [rcub/0]
root           16     2       0      0 0                    0 S [rcuc/0]
root           17     2       0      0 0                    0 S [rcu_exp_gp_kthread_worker]
root           18     2       0      0 0                    0 S [rcu_exp_par_gp_kthread_worker]
root           19     2       0      0 0                    0 S [migration/0]
root           20     2       0      0 0                    0 S [idle_inject/0]
root           22     2       0      0 0                    0 S [cpuhp/0]
root           23     2       0      0 0                    0 S [cpuhp/1]
root           24     2       0      0 0                    0 S [idle_inject/1]
root           25     2       0      0 0                    0 S [migration/1]
root           26     2       0      0 0                    0 S [rcuc/1]
root           27     2       0      0 0                    0 S [ksoftirqd/1]
root           30     2       0      0 0                    0 S [cpuhp/2]
root           31     2       0      0 0                    0 S [idle_inject/2]
root           32     2       0      0 0                    0 S [migration/2]
root           33     2       0      0 0                    0 S [rcuc/2]
root           34     2       0      0 0                    0 S [ksoftirqd/2]
root           37     2       0      0 0                    0 S [cpuhp/3]
root           38     2       0      0 0                    0 S [idle_inject/3]
root           39     2       0      0 0                    0 S [migration/3]
root           40     2       0      0 0                    0 S [rcuc/3]
root           41     2       0      0 0                    0 S [ksoftirqd/3]
root           44     2       0      0 0                    0 S [cpuhp/4]
root           45     2       0      0 0                    0 S [idle_inject/4]
root           46     2       0      0 0                    0 S [migration/4]
```

What is ADB?

ADB (Android Debug Bridge) is a command-line tool that allows communication between a computer and an Android device/emulator for debugging, testing, and analysis.

Purpose of Shell Access:

ADB shell provides direct command-line access to the Android operating system, enabling users to explore files, processes, permissions, and system behavior.

Conclusion:

The Android device was successfully paired using **ADB Wireless Debugging**, and shell access was obtained to execute basic Linux commands.