Session 8

It’s time to get familiar with commands lines to copy Files between PC/Laptop and Device

Copy **Fish.txt** and **Bear.txt** from your Laptop/PC to your Mobile Device:

1. Create a directory in C Drive. Name it a

C:/a

2. Open Notepad

Type FISH

Save it as fish.txt

3. Open another Notepad

Type Bear

Save it as bear.txt (in C:/a )

4. Create directory on your C drive and name it b

C:/b

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

 Check directories on your Real device (let’s practice and see what’s available. You may do the same with emulator, use –e if so )

adb -d shell

ls

exit

( usually the most  common folder is sdcard , just like temp directory on PC )

adb -d push C:/a/fish.txt /sdcard/    ( space after .txt)

check if fish.txt is in sdcard

adb -d shell

ls

cd sdcard

ls  ( to check content )

check for file

type exit to be out of shell

10. now do the same for bear.txt

**xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx**

How to copy bear.txt from your Device to your Laptop/PC to directory C:/b

adb -d pull /sdcard/bear.txt C:/b/ (also space after .txt and C:/b/)

check Directory C:/b/ if bear.txt is there

Repeat for fish.txt

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

How to make a screenshot and transfer this file to directory C:/a

1. Open some app , so you can distinguish the screenshot ( if device is locked, you are going to have black screen photo)

adb -d shell screencap -p /sdcard/vasya.png   ( or any other image format like .jpg, jpeg, bmp, etc )

2. check if vasya.png    is in sdcard

adb –d shell

cd sdcard

ls (to check content)

check for file

exit

3. copy vasya.png    to C:/a/

adb -d pull /sdcard/vasya.png C:/a/

check Directory C:/a/ if vasya.png  is there

How to record a 1 min video and move this file to C:/a/

this command only accepts seconds (not minute listing)

Make sure device is unlocked and you are doing something to record.

If device is locked, your video is going to have black screen

Most common Video formats are : .mpeg, .mpeg4, avi...etc

1.  adb -d shell screenrecord --time-limit 60 /sdcard/happy.mp4

2. check if happy.mp4  is in sdcard

adb -d shell

ls

cd sdcard

ls  ( to check content and look up file )

3.  copy happy.mp4 to C:/a/

adb -d pull /sdcard/happy.mp4 C:/a/

check Directory C:/a/ if happy.mp4 is there

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

How to create directory on your Real Device

adb -d shell

ls  ( if you wish to create sub-directory in already existing , exp sdcard )

cd sdcard

mkdir Class1

if not subdirectory needed , then

adb -d shell

mkdir Natalia

1. Create sub-directory in sdcard  ( sdcard/Natalia)

ls

cd sdcard

mkdir Natalia

exit shell (type exit)

2. Copy fish.txt to sdcard/Class1

adb -d push C:/a/fish.txt /sdcard/Class1/

3. Find fish.txt using ls

adb -d shell

ls

cd sdcard

cd Natalia

How to remove sub-directory  ( if it does have files, then you have to remove files first , then remove sub-directory)

adb -d shell

ls

cd sdcard

cd Natalia

rm fish.txt

rmdir Natalia

or you may just use :

adb -d shell

ls

cd sdcard

rm –r Class1 or any other name of the directory that you created ( current 3.1.2 and older)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Dumpsys Commands**

[**https://stackoverflow.com/questions/11201659/whats-the-android-adb-shell-dumpsys-tool-and-what-are-its-benefits**](https://stackoverflow.com/questions/11201659/whats-the-android-adb-shell-dumpsys-tool-and-what-are-its-benefits)

**dumpsys** is a tool that runs on Android devices and provides information about system services.

You can call **dumpsys** from the command line using the [Android Debug Bridge (ADB)](https://developer.android.com/studio/command-line/adb.html) to get diagnostic output for all system services running on a connected device.

 *(Please note : Some of the services might not work on your device )*

For a complete list of system services that you can use with **dumpsys**, use the following command:

**adb -d shell dumpsys –l**

**adb shell dumpsys <and option>**

the command below provides system data for input components, such as *touchscreens or built-in keyboards*:

**adb -d shell dumpsys input**

**adb -d shell dumpsys battery > C:/a/battery.txt**

Test UI performance

Specifying the **gfxinfo** service provides output with performance information relating to frames of animation that are occurring during the recording phase. The following command uses **gfxinfo** to gather UI performance data for a specified package name:

*You have to have your app open in order to run this command properly. We are using United APP*

**adb shell dumpsys gfxinfo *com.united.mobile.android***

**To find the UID for your app, run this command:**

 ***We are using our United APP that has a package name: com.united.mobile.android***

**adb shell dumpsys package *com.united.mobile.android***

Then look for the line labeled userId.

For example, to find network usage for the app 'com.example.myapp', run the following command:

**adb shell dumpsys package *com.united.mobile.android* | grep userId**

***if the above command line is not recognizing “grep” :***

**adb shell**

**dumpsys package *com.united.mobile.android* | grep userId ( *make sure in ID: I– is cap. d-lowercase)***

Output should be similar to the following: **userId=10007 gids=[3003, 1028, 1015]**

Using the sample dump above, look for lines that have uid=10007.

( note: dumpsys netstats )

*Two such lines exist—the first indicates a mobile connection and the second indicates a Wi-Fi connection.*

*This is an Output example that I will be using:*

 ident=[[type=WIFI, subType=COMBINED, networkId="MySSID"]] **uid=10007** set=DEFAULT tag=0x0

 NetworkStatsHistory: bucketDuration=7200000

 bucketStart=1406138400000 activeTime=7200000 **rxBytes**=17086802 **rxPackets**=15387 **txBytes**=1214969 **txPackets**=8036 operations=28

Below each line, you can see the following information for each two-hour window (which bucketDuration specifies in milliseconds):

* set=DEFAULT indicates foreground network usage, while set=BACKGROUND indicates background usage. set=ALL implies both.
* tag=0x0 indicates the socket tag associated with the traffic.
* rxBytes and rxPackets represent received bytes and received packets in the corresponding time interval.
* txBytes and txPackets represent sent (transmitted) bytes and sent packets in the corresponding time interval.

**Let’s check our battery**

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell dumpsys **battery**

*This command line will give you a complete info about your battery.*

***OUTPUT for battery :***

Current Battery Service state:

 AC powered: false

 USB powered: true

 Wireless powered: false

 Max charging current: 500000

 Max charging voltage: 5000000

 Charge counter: 1938174

 status: 2

 health: 2

 present: true

 level: 65

 scale: 100

 voltage: 4016

 temperature: 274

 technology: Li-ion

***To save battery info into a log file ;***

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell dumpsys **battery > C:\a\battery.txt**

***We can run a command line to extract only info that we need. In this example : temperature***

2. E:\AndroidStudio\android-sdk-windows\platform-tools>

adb shell

/$ dumpsys battery | grep ***temperature*** (extract data . In our case its “temperature”)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Inspect battery diagnostics**

Specifying the **batterystats** service generates interesting statistical data about battery usage on a device, organized by unique user ID (UID).

To learn how to use **dumpsys** to test your app for Doze and App Standby, go to Testing with Doze and App Standby.

The command for **batterystats** is as follows:

**adb shell dumpsys batterystats *options ( like: --charged or --checkin)***

To see a list of additional options available to **batterystats**, include the **-h** option.

The example below outputs battery usage statistics for a United app package since the device was last charged:

**adb shell dumpsys batterystats --charged *com.united.mobile.android***

The output typically includes the following:

* History of battery-related events
* Global statistics for the device
* Approximate power use per UID and system component
* Per-app mobile milliseconds per packet
* System UID aggregated statistics
* App UID aggregated statistics

Inspecting machine-friendly output

You can generate **batterystats** output in machine-readable CSV format by using the following command:

adb shell dumpsys batterystats --checkin

The following is an example of the output you should see:

9,0,i,vers,11,116,K,L

9,0,i,uid,1000,android

9,0,i,uid,1000,com.android.providers.settings

9,0,i,uid,1000,com.android.inputdevices

9,0,i,uid,1000,com.android.server.telecom

Battery-usage observations may be per-UID or system-level; data is selected for inclusion based on its usefulness in analyzing battery performance. Each row represents an observation with the following elements:

* A dummy integer
* The user ID associated with the observation
* The aggregation mode:
* "i" for information not tied to charged/uncharged status.
* "l" for --charged (usage since last charge).
* "u" for --unplugged (usage since last unplugged). Deprecated in Android 5.1.1.
* Section identifier, which determines how to interpret subsequent values in the line.

meminfo

You can record a snapshot of how your app's memory is divided between different types of RAM allocation with the following command:

The -d flag prints more info related to Dalvik and ART memory usage.

**Example :**

**adb shell dumpsys meminfo com.united.mobile.android -d**

**Testing with Doze and App Standby**

*To ensure a great experience for your users, you should test your app fully in Doze and App Standby.*

**Testing your app with Doze**

You can test Doze mode by following these steps:

* Configure a hardware device or virtual device with an Android 6.0 (API level 23) or higher system image.
* Connect the device to your development machine and install your app.
* Run your app and leave it active.
* Force the system into idle mode by running the following command:
* **$ adb shell dumpsys deviceidle force-idle**
* When ready, exit idle mode by running the following command:
* **$ adb shell dumpsys deviceidle unforce**
* Observe the behavior of your app after you reactivate the device. Make sure the app recovers gracefully when the device exits Doze.

**Testing your app with App Standby**

To test the App Standby mode with your app:

* Configure a hardware device or virtual device with an Android 6.0 (API level 23) or higher system image.
* Connect the device to your development machine and install your app.
* Run your app and leave it active.
* Force the app into App Standby mode by running the following commands:
* **$ adb shell dumpsys battery unplug**

**How to bring your battery back**

* adb shell dumpsys battery reset

$ adb shell am set-inactive <packageName> true

* Simulate waking your app using the following commands:
* $ adb shell am set-inactive <packageName> false

$ adb shell am get-inactive <packageName>

* Observe the behavior of your app after waking it. Make sure the app recovers gracefully from standby mode. In particular, you should check if your app's Notifications and background jobs continue to function as expected.

Acceptable use cases for whitelisting

The table below highlights the acceptable use cases for requesting or being on the Battery Optimizations exceptions whitelist. In general, your app should not be on the whitelist unless Doze or App Standby break the core function of the app or there is a technical reason why your app cannot use FCM high-priority messages.

For more information, see Support for other use cases .

Try dumpsys command lines and practice . Examples : cpuinfo, meminfo, wifi, location, etc

adb –d shell dumpsys cpuinfo





How to use Dumpsys for an application ( we need to find out a package name first )

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell

OnePlus3T:/ $ pm list packages | grep calculator

package:com.oneplus.calculator ( your package name has a different name )

OnePlus3T:/ $ exit

**Here we go :**

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell dumpsys meminfo com.oneplus.calculator