

ADB: Android Debug Bridge

and other Android command line tools

[How ADB Works](#) - from developer.android.com; includes all available commands

[How ADB Works](#) - from Google Git; adb overview

[How ADB Works](#) - from dummies.com / do not take it personally!

[Killing the adb server - why](#)

[Always starting with adb devices - why](#)

[Starting an emulator from command line](#)

[Connecting a physical device via TCP/IP](#)

[Command redirection](#)

[Install, update and uninstall an application](#)

[How to find a package name](#)

[Collecting application logs](#)

[Recording video from terminal/command prompt](#)

[Taking a screenshot from terminal/command prompt](#)

[Pull and push files from and to Android device](#)

[Changing runtime permissions from terminal/command prompt](#)

[Using dumpsys tool to diagnose the device \(when needed\)](#)

[Some other useful commands](#)

[Monkey tool](#)

[And where to look for more](#)

Killing the adb server - why

“In some cases, you might need to terminate the adb server process and then restart it to resolve the problem (e.g., if adb does not respond to a command).

To stop the adb server, use

```
adb kill-server
```

You can then restart the server by issuing any other adb command.”

Always starting with adb devices - why

To make sure the device you are going to manipulate with is actually connected, always start with

adb devices

And then issue your next adb command

Starting an emulator from command line

emulator -list-avds - returns the list of created emulators on your machine (their names)

emulator -avd <emulator_name> - start the emulator

Connecting a physical device via TCP/IP

Note: the very first link [How ADB Works](#) has the steps as well

Steps:

1. Your phone and computer are **ON THE SAME WiFi**
2. Connect your Android phone via USB
3. adb devices
4. adb tcpip <port_number_for_server>
6. Disconnect device from USB
6. adb connect 192.168.4.198:5559 - <phone_ip>:<port_number_for_server>
7. adb disconnect - disconnects every physical device connected this way

How to find ip from command line (alternatively, search in the phone settings)

```
adb -d shell ip addr show wlan0
```

Command redirection

adb -d <command> - sends a command to **the only** connected physical device
(CONNECTED via USB)

adb -e <command> - sends a command **to the only** connected emulator
(CONNECTED via TCP/IP)

Note: *Once a physical device is connected via TCP/IP, use -e or -s <serial_number> command redirection option as -d sends command to devices connected via USB*

if more than one device or more than one emulator connected, use

```
adb -s <serial> <command>
```

For ex.,

```
adb -s emulator-5554 install .apk
```

Install, update and uninstall an application

```
adb install Downloads/<file_name>.apk - use your path to .apk
```

```
adb devices | grep device | grep -v devices | cut -f 1 | xargs -l {} adb -s {} install
```

Reinstal (updated) application

```
adb install -r Downloads/<file_name>.apk - add -r before your path to .apk
```

Uninstall

```
adb uninstall com.adjoy.standalone.test2 - use package name
```

How to find a package name

```
adb shell pm list packages - returns the list of packages installed on the device
```

```
adb shell pm list packages -f <app_name> - returns the package for the specific app
```

```
adb shell pm list packages -3 -returns third party packages that were installed from Play store or as .apk (not pre-installed)
```

Alternatively, with dumpsys. First start the app do that it is running on the foreground, then

```
adb shell dumpsys window windows | grep 'mCurrentFocus'
```

Returns

```
mCurrentFocus=Window{2c0df8a u0
```

```
com.adjoy.standalone.test2/com.adjoy.standalone.features.auth.AuthActivity}
```

Note: *Does not work for Android 10 devices*

Collecting application logs

[Logcat](#) is a command-line tool for debugging Android applications

adb logcat - command to start logging

useful options:

adb logcat -c clears all the info that might be in buffer from the previous sessions

adb logcat | grep 'adjoy' - filter the log for a particular application

*Note: On Windows machine, please use **find** instead of **grep** - for ex., find “adjoy”*

Note: If you need to grep more than one word, please do the following:

adb logcat | grep -E "(adjoy|dabbl)"

adb logcat > file_name.txt - writes the log a text file (or -f <file_name>), for example,

adb logcat | grep 'adjoy' > zip_code_crash.txt

*Note: On Windows machine, please use **find** instead of **grep** - for ex., find “adjoy”*

adb logcat | tee logfile.txt - prints the output in the console and also saves to a file

adb logcat **tag:priority** - filtering by priority; for example,

```
adb logcat *:W
```

Note: *Tags are defined by an app developer, use * in a tag place*

Note: *if you use **zsh**, you need to use single quotes around the expression ***:W***

More about filtering from the [official website](#)

```
adb logcat '*:W' | grep 'adjoy' > zip_code_crash.txt
```

Note: *On Windows machine, please use **find** instead of **grep** - for ex., find “adjoy”*

Recording video from terminal/command prompt

```
adb shell screenrecord /sdcard/ErrorMsgRegistrationScreen.mp4
```

give your files a meaningful name. You may use a bug # as a file name, too.

Default recording time is 180 seconds (3 minutes). You may, however, change the that by adding following the arguments

```
adb shell screenrecord --time-limit <TIME> /sdcard/ErrorMsgRegistrationScreen.mp4,
```

instead of <TIME> placeholder, insert the needed time in seconds: --time-limit 120 will produce a 2-minute video.

Since video is saved to sdcard, we need to "pull" it from the device

```
adb pull /sdcard/ErrorMsgRegistrationScreen.mp4 - pulls to current working directory
```

`adb pull /sdcard/ErrorMsgRegistrationScreen.mp4 /Users/tanya/Desktop` - pulls to a specified destination

If no destination directory specified, the file will be stored at your current working directory (to check - `pwd` on Mac, `cd` on Windows)

To remove a file from your device, run

```
adb shell rm /sdcard/ErrorMsgRegistrationScreen.mp4
```

Recording the video in Android Studio -

<https://developer.android.com/studio/debug/am-video.html?hl=en>

Taking a screenshot from terminal/command prompt

```
adb shell screencap -p > ~/Desktop/screenshot.png
```

`-p` forces screencap to use PNG format

`adb shell screencap /sdcard/screenshot.png` - saves the screenshot to the device's sdcard; always `.png`! You have to say it explicitly

To pull it from device

`adb pull /sdcard/screenshot.png` - is destination is not specified, goes to current working directory

`adb pull /sdcard/screenshot.png /Users/tanya/Desktop` - pulls to the Desktop

Remember - we can also do it in the emulator settings and in Android Studio (under Logcat) with buttons.

Pull and push files from and to Android device

`adb pull /sdcard/screenshot.png /Users/tanya/Desktop` - pulls (copies) the file to the Desktop

`adb push /Users/tanya/Desktop/profile_image.png /sdcard/` - push (copies) the file to device sdcard

Note: *You may drag and drop file from your computer to your emulator, that includes .apk files - an easy way to install the app*

Changing runtime permissions from terminal/command prompt

To find out, what runtime permissions your app is using, run

`adb shell dumpsys package <package_name> | grep permission`

The following (partial) output under runtime permissions

`android.permission.CAMERA: granted=true`

means the user is currently allowing the app to access the camera

To revoke permission, run

`adb shell pm revoke <package_name> android.permission.CAMERA`

To grant permission, run

`adb shell pm grant <package_name> android.permission.CAMERA`

The same way you can revoke and grant any **RUNTIME permission** your application may require; saves time especially if working with an emulator.

Using dumpsys tool to diagnose the device (when needed)

“`dumpsys` is a tool that runs on Android devices and provides information about system services”

Sometimes it could be useful to get information about a device memory or battery usage. You can do it this (and not only with `dumpsys`)

How to get an info related to the battery consumption, find out [here](#)

How to get an info about the memory is [there](#)

Some other useful commands

`adb shell getprop` - returns the list of the devices properties; could be useful when you need to get, say, a device model or Android version

`adb shell getprop ro.build.version.release` - returns Android version installed on the device, for example, Android 8.0

`adb shell getprop ro.build.version.sdk` - returns API level, for example, 27

`adb shell pm clear com.adjoy.standalone.test2` - clears all package data from the device; after running this command, the app is as if it were just installed

`adb shell am force-stop com.adjoy.standalone.test2` - `am` stands for activity manager; the command `force` closes the application

`adb shell am start <package_name>/<activity_name>` - starts the application from command line; you'd need to know what is the app's first activity that is created on launch

For example,

```
adb shell am start
com.adjoy.standalone.test2/com.adjoy.standalone.features.auth.AuthActivity
```

adb reboot - reboots the device

Monkey tool

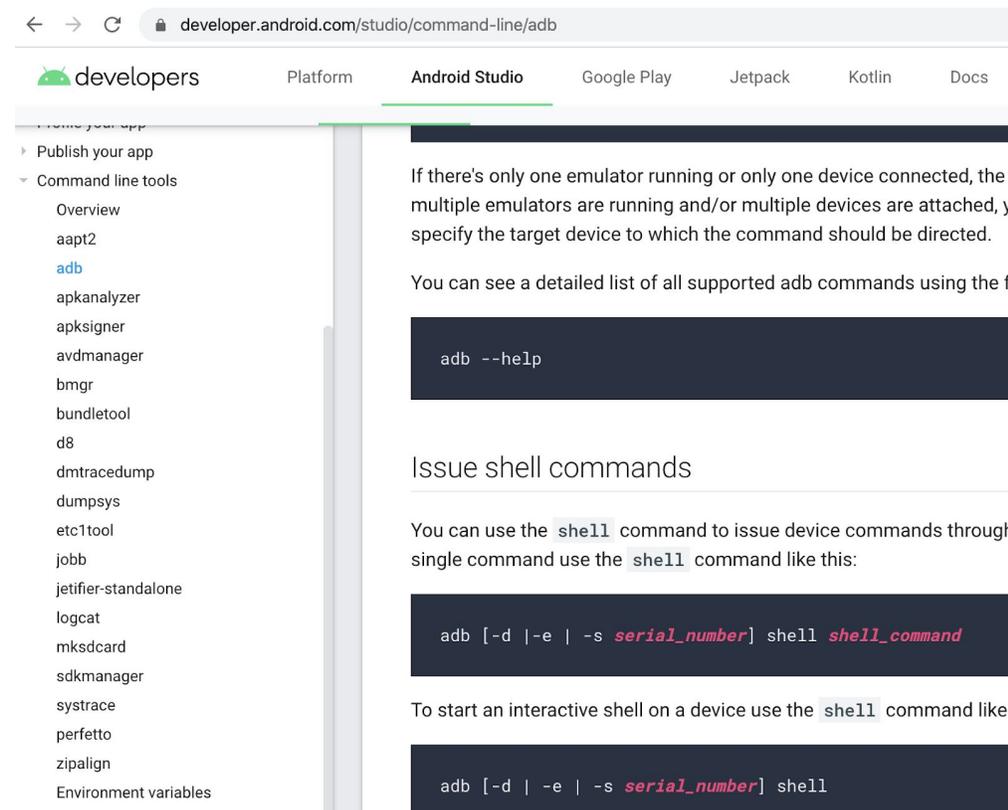
“The [Monkey](#) is a program that runs on your emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events.”

```
adb shell monkey -p com.adjoy.standalone -v 500 - sends 500 random clicks and
touches to Dabbl app
```

And where to look for more

Android command lines tools <https://developer.android.com/studio/command-line>

[Magic of ADB](#) - a talk by **Wojciech Sadurski**, Google at droidcon San Francisco 2019



The screenshot shows the Android Studio website's documentation for ADB (Android Debug Bridge) command line tools. The page is titled "developer.android.com/studio/command-line/adb". The navigation menu includes "Platform", "Android Studio", "Google Play", "Jetpack", "Kotlin", and "Docs". The left sidebar lists various tools, with "adb" highlighted. The main content area explains that if there's only one emulator or device, the target is specified. It provides a code block for the command `adb --help`. Below that, it discusses "Issue shell commands" and provides a code block for the command `adb [-d | -e | -s serial_number] shell shell_command`. It also mentions that to start an interactive shell, the `shell` command is used, with a corresponding code block: `adb [-d | -e | -s serial_number] shell`.