# Android SDK Platform Tools

You can download the SDK Platform Tools [here](https://developer.android.com/studio/releases/platform-tools#downloads.html). These include:

## adb

A tool that lets you **manage the state of an emulator instance** or Android-powered device. You can use it to install and debug apps on a device.

## fastboot

**Flashes a device** with platform and other system images.

## logcat

Invoked by adb to **view logs of app and system**.

## etc1tool

A command-line utility that lets you **encode PNG images to the ETC1 compression** standard and **decode** ETC1 compressed images back to PNG.

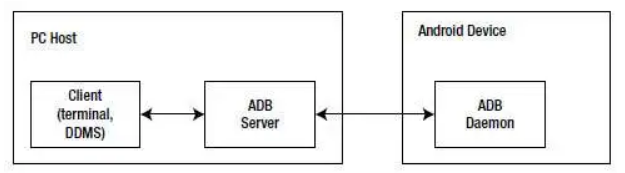
# adb

## Introduction

Android Debug Bridge (ADB) is a **command-line tool** that lets you **communicate with Android devices**. It provides access to a **Unix shell** that you can use to run various commands.

It is a client-server program that includes 3 components:

* A client, which sends commands. It runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.
* A daemon (adbd), which runs commands on a device. It runs as a background process on each device.
* A server, which manages communication between the client and the daemon. It runs as a background process on your development machine.



## How adb Works

When you start an adb client, it first checks whether there is an adb server already running. If there isn't, it starts the server. When the server starts, it binds to **local TCP port 5037** and listens for commands sent from adb clients. Note: All adb clients use port 5037 to communicate with the adb server.

Then the server sets up connections to all running devices. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, which is the range used by the first 16 emulators. Where the server finds an adb daemon (adbd), it sets up a connection to that port.

Each emulator uses a pair of sequential ports — a**n even-numbered port for console connections** and an **odd-numbered port for adb connections**. For example:

* Emulator 1, console: 5554
* Emulator 1, adb: 5555
* Emulator 2, console: 5556
* Emulator 2, adb: 5557
* Etc.

Once the server has set up connections to all devices, you can use adb commands to access those devices. Because the server manages connections to devices and handles commands from multiple adb clients, you can control any device from any client or from a script.

## Setup

First, you need to connect your device and your PC host with a USB. To verify that your device is connected, you can run command "adb devices". If connected, you'll see the device name listed as a "device."

Note: In addition to USB, adb allows connection via WiFi. For more details, check [here](https://developer.android.com/tools/adb#connect-to-a-device-over-wi-fi).

Second, you now can run any adb command.

## Commands

<https://developer.android.com/tools/adb>

<https://devhints.io/adb>

<https://www.linuxtopia.org/online_books/android/devguide/guide/developing/tools/android_adb_commandsummary.html>

<https://gist.github.com/Pulimet/5013acf2cd5b28e55036c82c91bd56d8>

|  |  |
| --- | --- |
| **Command** | **Usage** |
| adb devices -l | **Get a list of attached devices**.  In response, it prints status information for each device:   * Serial number: adb creates a string to uniquely identify the device by its port number. For example, emulator-5554 * State: Connection state of the device can be one of the following:   + offline: The device is not connected to adb or is not responding.   + device: The device is connected to the adb server. Note that this state does not imply that the Android system is fully booted and operational, because the device connects to adb while the system is still booting. After boot-up, this is the normal operational state of a device.   + no device: There is no device connected. * Description: If you include the -l option, the devices command tells you what the device is. This information is helpful when you have multiple devices connected so that you can tell them apart.   **Example:**  $ adb devices  List of devices attached  emulator-5556 device product:sdk\_google\_phone\_x86\_64 model:Android\_SDK\_built\_for\_x86\_64 device:generic\_x86\_64  emulator-5554 device product:sdk\_google\_phone\_x86 model:Android\_SDK\_built\_for\_x86 device:generic\_x86  0a388e93 device usb:1-1 product:razor model:Nexus\_7 device:flo  There are three devices running. The first two lines are emulators, and the third line is a hardware device that is attached to the computer.  **Note:**  In rare cases, this command does not show up any device even though the emulators are visible on your desktop. [Here](https://developer.android.com/tools/adb#notlisted) is what you need to check to resolve the issue. |
| adb kill-server |  |
| adb forward | Set up arbitrary port forwarding, which **forwards requests on a specific host port to a different port on a device**.  **Example:**  The following example sets up forwarding of host port 6100 to device port 7100:  $ adb forward tcp:6100 tcp:7100  The following example sets up forwarding of host port 6100 to local:logd:  $ adb forward tcp:6100 local:logd |
| adb kill-server | **Stop the adb server**.  In some cases, you might need to terminate the adb server process and then restart it to resolve the problem. For example, when adb does not respond to a command.  After stopping the server, you can then restart it by issuing any other adb command |
| adb [-d | -e | -s *serial\_number*] *command* | **Issue any adb command from a command line or from a script**.  If there's only one emulator running or only one device connected, the adb command is sent to that device by default. If multiple emulators are running and/or multiple devices are attached, you need to use the -d, -e, or -s option to specify the target device to which the command should be directed. |
| adb [-d |-e | -s *serial\_number*] shell *shell\_command* | **Issue device commands through adb or to start an interactive shell**.  **Example**:  The following example lists all directories and files in /system/bin in the device:  $ adb shell ls /system/bin |
| adb [-d | -e | -s *serial\_number*] shell | **Start an interactive shell on a device.**  If you don’t want to type "adb [-d |-e | -s serial\_number] shell" every time you want to issue device commands, you can start an interactive shell and simply execute any device command from it.  **Example**:  The following example lists all directories and files in /system/bin in the device, and create a new directory called "abc" in the current directory:  $ adb shell  # ls /system/bin  # mkdir abc |

# fastboot

# logcat

The logcat **display logs from device in real time**.

## Command Line Syntax

To run logcat through the adb shell, the general usage is:

$ adb shell logcat [<option>] ... [<filter-spec>] ...

There is a shorthand of adb logcat, which just expands to adb shell logcat.

For all options and filter specs, run adb logcat --help.

## Logging System Overview

The Android logging system is a set of structured **circular buffers** maintained by the system process logd.

The set of buffers is fixed and defined by the system. The most relevant ones are:

* main: Stores most application logs.
* system: Stores messages originating from the Android OS.
* crash: Stores crash logs. Each log entry has a priority, a tag that identifies the origin of the log, and the actual log message.

The C/C++ interface to the logging system is the shared library liblog and its header <android/log.h>. All language-specific logging facilities (including android.util.Log) eventually call the function \_\_android\_log\_write.

By default, it calls the function \_\_android\_log\_logd\_logger, which **sends the log entry to logd using a socket**.

## Log Levels

The priority is one of the following character values, ordered from lowest to highest priority:

* V: Verbose (lowest priority)
* D: Debug
* I: Info
* W: Warning
* E: Error
* F: Fatal
* S: Silent (highest priority, where nothing is ever printed)

## Filter log output

<https://developer.android.com/tools/logcat#filteringOutput>

## Control Log Output Format

To format the output, use the -v option:

$ adb logcat [-v <format>]

Following output formats are supported:

* brief: Displays priority, tag, and PID of the process issuing the message.
* long: Displays all metadata fields and separate messages with blank lines.
* process: Displays PID only.
* raw: Displays the raw log message with no other metadata fields.
* tag: Displays the priority and tag only.
* thread: A legacy format that shows priority, PID, and TID of the thread issuing the message.
* threadtime (default): Displays the date, invocation time, priority, tag, PID, and TID of the thread issuing the message.
* time: Displays the date, invocation time, priority, tag, and PID of the process issuing the message.

## Format Modifiers

To change the logcat output, specify a format modifier via the -v option:

$ adb logcat -b all -v color -d

Following is modifier details (you can see them all with logcat -v --help):

* color: Shows each priority level with a different color.
* descriptive: Shows log buffer event descriptions. This modifier affects event log buffer messages only and has no effect on the other non-binary buffers. The event descriptions come from the event-log-tags database.
* epoch: Displays time in seconds starting from Jan 1, 1970.
* monotonic: Displays time in CPU seconds starting from the last boot.
* printable: Ensures that any binary logging content is escaped.
* uid: If permitted by access controls, displays the UID or Android ID of the logged process.
* usec: Displays the time, with precision in microseconds.
* UTC: Displays the time as UTC.
* year: Adds the year to the displayed time.
* zone: Adds the local time zone to the displayed time.

## View Alternative Log Buffers

The Android logging system keeps **multiple circular buffers** for log messages, and not all of log messages are sent to the default circular buffer. To see additional log messages, use the -b option:

$ adb logcat [-b <buffer>]

You can view any of these alternate buffers:

* radio: Views the buffer that contains radio/telephony related messages.
* events: Views the interpreted binary system event buffer messages.
* main: Views the main log buffer (default), which doesn't contain system and crash log messages.
* system: Views the system log buffer (default).
* crash: Views the crash log buffer (default).
* all: Views all buffers.
* default: Reports main, system, and crash buffers.

To specify multiple -b flags for all the buffers you want to print, run:

$ logcat -b main -b radio -b events

Or

$ logcat -b main,radio,events

Detailed guideline: <https://developer.android.com/tools/logcat>