Android Developers

Android Debug Bridge

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device (an emulator or a connected Android device). The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device. It is a client-server program that includes three components:

- **A client**, which sends commands. The client 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. The daemon runs as a background process on each device.

- **A server**, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

`adb` is included in the Android SDK Platform-Tools package. You can download this package with the SDK Manager (https://developer.android.com/studio/intro/update.html#sdk-manager), which installs it at `android_sdk`/`platform-tools/`. Or if you want the standalone Android SDK Platform-Tools package, you can download it here (https://developer.android.com/studio/releases/platform-tools.html).

See also

Control the Emulator from the Command Line

How adb works

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

The server then sets up connections to all running devices. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, the range used by the first 16 emulators. Where the server finds an adb daemon (adbd), it sets up a connection to that port. Note that each emulator uses a pair of sequential ports — an 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
>
> and so on...

As shown, the emulator connected to adb on port 5555 is the same as the emulator whose console listens on port 5554.

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).

Enable adb debugging on your device

To use adb with a device connected over USB, you must enable **USB debugging** in the device system settings, under **Developer options**.

On Android 4.2 and higher, the Developer options screen is hidden by default. To make it visible, go to **Settings > About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options** at the bottom.

On some devices, the Developer options screen might be located or named differently.

You can now connect your device with USB. You can verify that your device is connected by executing `adb devices` from the `android_sdk`/`platform-tools/` directory. If connected, you'll see the device name listed as a "device."

> **Note:** When you connect a device running Android 4.2.2 or higher, the system shows a dialog asking whether to accept an RSA key that allows debugging through this computer. This security mechanism protects user devices because it ensures that USB debugging and other adb commands cannot be executed unless you're able to unlock the device and acknowledge the dialog.

For more information about connecting to a device over USB, read Run Apps on a Hardware Device

This site uses cookies to store your preferences for site-specific language and display options

Connect to a device over Wi-Fi

adb usually communicates with the device over USB, but you can also use adb over Wi-Fi after some initial setup over USB, as described below. If you're developing for Android Wear, however, you should instead see the guide to debugging an Android Wear app (https://developer.android.com/training/wearables /apps/debugging.html), which has special instructions for using adb with Wi-Fi and Bluetooth.

1. Connect your Android device and adb host computer to a common Wi-Fi network accessible to both. Beware that not all access points are suitable; you might need to use an access point whose firewall is configured properly to support adb.

2. If you are connecting to an Android Wear device, turn off Bluetooth on the phone that's paired with the device.

3. Connect the device to the host computer with a USB cable.

4. Set the target device to listen for a TCP/IP connection on port 5555.

   ```
   adb tcpip 5555
   ```

5. Disconnect the USB cable from the target device.

6. Find the IP address of the Android device. For example, on a Nexus device, you can find the IP address at **Settings** > **About tablet** (or **About phone**) > **Status** > **IP address**. Or, on an Android Wear device, you can find the IP address at **Settings** > **Wi-Fi Settings** > **Advanced** > **IP address**.

7. Connect to the device by its IP address.

   ```
   adb connect device_ip_address
   ```

8. Confirm that your host computer is connected to the target device:

   ```
   $ adb devices
   List of devices attached
   device_ip_address:5555 device
   ```

You're now good to go!

This site uses cookies to store your preferences for site-specific language and display options.   OK

1. Make sure that your host is still connected to the same Wi-Fi network your Android device is.

2. Reconnect by executing the `adb connect` step again.

3. Or if that doesn't work, reset your adb host:

   ```
   adb kill-server
   ```

   Then start over from the beginning.

Query for devices

Before issuing adb commands, it is helpful to know what device instances are connected to the adb server. You can generate a list of attached devices using the `devices` command.

```
adb devices -l
```

In response, adb prints this status information for each device:

- Serial number: A string created by adb to uniquely identify the device by its port number. Here's an example serial number: `emulator-5554`

- State: The 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 now 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. However, after boot-up, this is the normal operational state of an 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.

The following example shows the `devices` command and its output. There are three devices running. The first two lines in the list are emulators, and the third line is a physical device that is attached to the computer.

```
$ adb devices
List of devices attached
```

This site uses cookies to store your preferences for site-specific language and display options. OK

```
emulator-5554 device product:sdk_google_phone_x86 model:Android_SDK_built_for_x86
0a388e93      device usb:1-1 product:razor model:Nexus_7 device:flo
```

Emulator not listed

The `adb devices` command has a corner-case command sequence that causes running emulator(s) to not show up in the `adb devices` output even though the emulator(s) are visible on your desktop. This happens when *all* of the following conditions are true:

1. The adb server is not running, and

2. You use the `emulator` command with the `-port` or `-ports` option with an odd-numbered port value between 5554 and 5584, and

3. The odd-numbered port you chose is not busy so the port connection can be made at the specified port number, or if it is busy, the emulator switches to another port that meets the requirements in 2, and

4. You start the adb server after you start the emulator.

One way to avoid this situation is let the emulator choose its own ports, and don't run more than 16 emulators at once. Another way is to always start the adb server before you use the `emulator` command, as explained in the following examples.

Example 1: In the following command sequence, the `adb devices` command starts the adb server, but the list of devices does not appear.

Stop the adb server and enter the following commands in the order shown. For the avd name, provide a valid avd name from your system. To get a list of avd names, type `emulator -list-avds`. The `emulator` command is in the *android_sdk*/tools directory.

```
$ adb kill-server
$ emulator -avd Nexus_6_API_25 -port 5555
$ adb devices

List of devices attached
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
```

Example 2: In the following command sequence, `adb devices` displays the list of devices because the adb server was started first.

To see the emulator in the `adb devices` output, stop the adb server, and then start it again after using

This site uses cookies to store your preferences for site-specific language and display options. OK

```
$ adb kill-server
$ emulator -avd Nexus_6_API_25 -port 5557
$ adb start-server
$ adb devices

List of devices attached
emulator-5557 device
```

For more information about emulator command-line options, see Using Command Line Parameters (https://developer.android.com/studio/run/emulator-commandline.html#startup-options).

Send commands to a specific device

If multiple devices are running, you must specify the target device when you issue the adb command. To specify the target, use the `devices` command to get the serial number of the target. Once you have the serial number, use the `-s` option with the adb commands to specify the serial number. If you're going to issue a lot of adb commands, you can set the `ANDROID_SERIAL` environment variable to contain the serial number instead. If you use both `-s` and `ANDROID_SERIAL`, `-s` overrides `ANDROID_SERIAL`.

In the following example, the list of attached devices is obtained, and then the serial number of one of the devices is used to install the `helloWorld.apk` on that device.

```
$ adb devices
List of devices attached
emulator-5554 device
emulator-5555 device

$ adb -s emulator-5555 install helloWorld.apk
```

Note: If you issue a command without specifying a target device when multiple devices are available, adb generates an error.

If you have multiple devices available (hardware or emulated), but only one is an emulator, use the `-e` option to send commands to the emulator. Likewise, if there are multiple devices but only one hardware device attached, use the `-d` option to send commands to the hardware device.

Install an app

This site uses cookies to store your preferences for site-specific language and display options   OK

```
adb install path_to_apk
```

To first build your APK, you can build it with Gradle (https://developer.android.com/studio/build/building-cmdline.html) or instead build and install it with Android Studio (https://developer.android.com/studio/run/index.html).

Set up port forwarding

You can use the `forward` command to set up arbitrary port forwarding, which forwards requests on a specific host port to a different port on a device. 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
```

Copy files to/from a device

Use the `pull` and `push` commands to copy files to and from an device. Unlike the `install` command, which only copies an APK file to a specific location, the `pull` and `push` commands let you copy arbitrary directories and files to any location in a device.

To copy a file or directory and its sub-directories *from* the device, do the following:

```
adb pull remote local
```

To copy a file or directory and its sub-directories *to* the device, do the following:

```
adb push local remote
```

Replace *local* and *remote* with the paths to the target files/directory on your development machine (local) and on the device (remote). For example:

```
adb push foo.txt /sdcard/foo.txt
```

Stop the adb server

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 the `adb kill-server` command. You can then restart the server by issuing any other adb command.

adb commands reference

You can issue adb commands from a command line on your development machine or from a script. The usage is:

```
adb [-d | -e | -s serial_number] command
```

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.

The table below lists all of the supported adb commands and explains their meaning and usage.

Table 1. Available adb commands and options

Global Options	Description
`-a`	Listen on all network interfaces instead of only on `localhost`.
`-d`	Direct an adb command to the only attached USB device. Returns an error when more than one USB device is attached.
`-e`	Direct an adb command to the only running emulator. Returns an error when more than one emulator is running.
`-s serial_number`	Direct an adb command to a specific device, referred to by its adb-assigned serial number (such as emulator-5556). Overrides the serial number value stored in the $ANDROID_SERIAL environment variable. See Directing

	Commands to a Specific Emulator or Device (#directingcommands).
`-p` *`product`*	The name of or path to the device product. An example path is (`'angler'`/`'out/target/product/angler');`. The default product is the value in the `$ANDROID_PRODUCT_OUT` environment variable.
`-H` *`server`*	The name of the adb server host. The default value is `localhost`.
`-P` *`port`*	The adb server port number. The default value is `5037`.
`-L` *`socket`*	Listen on the provided adb server socket. The default value is `tcp:localhost:5037`.
General Commands	**Description**
`devices [-l]`	Print a list of all devices. Use the `-l` option to include the device descriptions. For more information, see Querying for Emulator/Device Instances (#devicestatus).
`help`	Print a list of supported adb commands and their descriptions.
`version`	Print the adb version number.
Networking Commands	**Description**
`connect` *`host`*`[:`*`port`*`]`	Connect to a device over TCP/IP. If you do not specify a port, then the default port, `5555`, is used.
`disconnect [`*`host`* `|` *`host`*`:`*`port`*`]`	Disconnect from the specified TCP/IP device running on the specified port. If you do not specify a host or a port, then all devices are disconnected from all TCP/IP ports. If you specify a host, but not a port, the default port, `5555`, is used.
`forward -list`	List all forwarded socket connections.
`forward [--no-rebind]` *`local remote`*	Forward socket connections from the specified local port to the specified remote port on the device. You can specify both ***local*** and ***remote*** ports in the following ways: `tcp:`*`port`*. To choose any open port, make the ***local*** value `tcp:0`. `localabstract:`*`unix_domain_socket_name`*. `localreserved:`*`unix_domain_socket_name`*

- localfilesystem:*unix_domain_socket_name*.

- dev:*character_device_name*.

- jdwp:*pid*.

forward --remove *local*	Remove the specified forwarded socket connection.
reverse -l	List all reverse socket connections from the device.
reverse [--no-rebind] *remote local*	Reverse a socket connection. The --no-rebind option means the reversal fails if the specified socket is already bound through a previous reverse command. You can specify the port for both local and remote arguments in the following ways: • tcp:*port*. To choose any open port, make the **remote** value tcp:0. • localabstract:*unix_domain_socket_name*. • localreserved:*unix_domain_socket_name*. • localfilesystem:*unix_domain_socket_name*.
reverse --remove *remote*	Remove the specified reverse socket connection from the device.
reverse --remove-all	Remove all reverse socket connections from the device.
File Transfer Commands	Description
push *local remote*	Copy files and directories from the local device (computer) to a remote location on the device.
pull [-a] *remote local*	Copy remote files and directories to a device. Use the -a option to preserve the file time stamp and mode.
App Installation Commands	Description
install [*options*] *package*	Push packages to the device and install them. Possible options are the following: • -l: Forward lock application. • -r: Replace the existing application. • -t: Allow test packages. • -s: Install the application on the SD card.

This site uses cookies to store your preferences for site-specific language and display options. OK

- −d: Allow version code downgrade (debugging packages only).

- −g: Grant all runtime permissions.

install−multiple [*options*] *packages*	Same options as `install` with the addition of the following: −p: Partial application install.
uninstall [−k] *package*	Remove this app package from the device. Add the −k option to keep the data and cache directories.
Backup and Restore Commands	**Description**
backup [−f *file*] [−apk|−noapk] [−obb|−noobb] [−shared|−noshared] [−all] [−system| −nosystem] *package_names*	Write an archive of the device's data to ***file***. If you do not specify a file name, the default file is `backup.adb`. The ***package*** list is optional when you specify the −all and −shared options. The following describes the usages for the other options: • −apk | −noapk: Back up or do not back up .apk files. The default value is −noapk. • −obb | −noobb: Back up or do not back up .obb files. The default value is −noobb. • −shared | −noshared: Back up or do not back up shared storage. The default value is −noshared. • −all: Back up all installed applications. • −system | −nosystem: Include or do not include system apps when backing up all installed applications (−all). The default value is −system.
restore *file*	Restore the device contents from ***file***.
Debug Commands	**Description**
bugreport *path*	Print a `bugreport` to the specified path. If ***path*** is a directory, then the bug report is saved to that directory using the default file name, `bugreport.zip`. Devices that do not support zipped bug reports print to `stdout`.
jdwp	Print a list of the available JDWP processes on a given device. Use `forward jdwp:`*pid* to connect to a specific JDWP process. For example:

This site uses cookies to store your preferences for site-specific language and display options. OK

```
jdb –attach localhost:8000
```

`logcat [-help] [`*`option`*`]` `[`*`filter-spec`*`]`	Print log data to the screen. For information about the `logcat` command and the `$ANDROID_LOG_TAGS` environment variable, see Filtering Output (https://developer.android.com/studio/command-line/logcat.html#filteringOutput) on the logcat page. The `$ADB_TRACE` environment variable contains a comma-separated list of the debug information to log. Values can be any combination of the following: `all`, `adb`, `sockets`, `packets`, `rwx`, `usb`, `sync`, `sysdeps`, `transport`, and `jdwp`. See also Logcat Command-Line Tool (https://developer.android.com /studio/command-line/logcat.html).
Security Commands	**Description**
`disable-verity`	Disable `dm-verity` checking on `userdebug` builds. The `dm-verity` option ensures that when a user boots a device that it is in the same state that it was in when it was last used. For more information, see Verified Boot (https://source.android.com /security/verifiedboot/).
`enable-verity`	Re-enable `dm-verity` checking on `userdebug` builds. The `dm-verity` option ensures that when a user boots a device that it is in the same state that it was in when it was last used. For more information, see Verified Boot (https://source.android.com /security/verifiedboot/).
`keygen` *`file`*	Generate adb public and private key encryption. The private key is stored in *`file`*. The public key is stored in *`file`*`.pub`. The `$ANDROID_VENDOR_KEYS` environment variable contains a colon-separated list of keys (files or directories).
Scripting Commands	**Description**
`wait-for [`*`-transport`*`]` *`-state`*	Wait for the device to be in the specified state. • `state`: Values can be `device`, `recovery`, `sideload`, or `bootloader`. • `transport`: Values can be `usb`, `local`, or `any`.
`get-state`	Print the adb state of a device. The adb state can be `print offline`, `bootloader`, or `device`. For more information, see

This site uses cookies to store your preferences for site-specific language and display options. OK

`get-serialno`	Print the adb device serial number string. For more information, see Querying for Emulator/Device Instances (#devicestatus).
`get-devpath`	Print the adb device path.
`remount`	Remount the `/system`, `/vendor`, and `/oem` partitions in read-write mode.
`reboot [bootloader | recovery | sideload | sideload-auto-reboot ]`	Reboot the device. This command defaults to booting the system image, but also supports `bootloader` and `recovery`. - The `bootloader` option reboots into bootloader. - The `recovery` option reboots into recovery. - The `sideload` option reboots into recovery and starts `sideload` mode. - The `sideload-auto-reboot` option is the same as `sideload`, but reboots after side loading completes.
`sideload otapackage`	Side load (install in APK format) the specified full OTA package onto the device.
`root`	Restart adbd with root permissions.
`unroot`	Restart adbd without root permissions.
`usb`	Restart the adb server listening on USB.
`tcpip port-number`	Restart the adb server listening on TCP at the specified port.
Internal Debugging Commands	Description
`start-server`	Check whether the adb server process is running.
`kill-server`	Terminate the adb server process.
`reconnect`	Force a reconnect from the host.
`reconnect device`	Force a reconnect from the device to force a reconnect.
Shell Commands	Description
`shell`	Start a remote interactive shell in the target device. For more information, see Issue shell commands (#shellcommands).
`shell -e escape_char [-n] [-T] [-t] [-x] [command]`	Issue a shell command in the target device and then exit the remote shell. Use any combination of the following options:

not want to use an escape character. If you do not provide a value, the default escape character (a dash (–)), is used.

- `–n`: Do not read from `stdin`.

- `–T`: Disable pseudo-terminal utiity (PTY) allocation.

- `–t`: Force PTY allocation.

- `–x`: Disable remote exit codes and `stdout/stderr` separation.

For more information, see Issue shell commands (#shellcommands).

`emu` *`command`*	Run an emulator console command. For more information, see Control the Emulator from the Command Line (https://developer.android.com/studio/run/emulator-commandline.html)

Issue shell commands

You can use the `shell` command to issue device commands through adb, with or without entering the adb remote shell on the device. To issue a single command without entering a remote shell, use the `shell` command like this:

```
adb [–d |–e | –s serial_number] shell shell_command
```

Or enter a remote shell on a device like this:

```
adb [–d | –e | –s serial_number] shell
```

When you are ready to exit the remote shell, press Control + D or type `exit`.

The shell command binaries are stored in the file system of the device at `/system/bin/`.

> **Note:** With **Android Platform-Tools 23** and higher, adb handles arguments the same way that the `ssh(1)` command does. This change has fixed a lot of problems with command injection (https://en.wikipedia.org/wiki/Code_injection#Shell_injection) and makes it possible to now safely execute commands that contain shell metacharacters (https://en.wikipedia.org/wiki/Metacharacter), such as "adb install Let\'sGo.apk". But this change means that the interpretation of any command that

This site uses cookies to store your preferences for site-specific language and display options.

b'"" command is now an error because the single quotes (') are swallowed by the local shell, and the device sees "adb shell setprop foo a b". To make the command work, quote twice, once for the local shell and once for the remote shell, the same as you do with ssh(1). For example, "adb shell setprop foo "'a b'"".

Call activity manager (am)

Within an adb shell, you can issue commands with the activity manager (am) tool to perform various system actions, such as start an activity, force-stop a process, broadcast an intent, modify the device screen properties, and more. While in a shell, the syntax is:

```
am command
```

You can also issue an activity manager command directly from adb without entering a remote shell. For example:

```
adb shell am start -a android.intent.action.VIEW
```

Table 2. Available activity manager commands

Command	Description
start [options] intent	Start an Activity (https://developer.android.com/reference/android/app/Activity.html) specified by intent. See the Specification for intent arguments (#IntentSpec). Options are: -D: Enable debugging. -W: Wait for launch to complete. --start-profiler file: Start profiler and send results to file. -P file: Like --start-profiler, but profiling stops when the app goes idle. -R count: Repeat the activity launch count times. Prior to each repeat, the top activity will be finished. -S: Force stop the target app before starting the

	• `--opengl-trace`: Enable tracing of OpenGL functions. • `--user` *`user_id`* `| current`: Specify which user to run as; if not specified, then run as the current user.
`startservice [`*`options`*`]` *`intent`*	Start the `Service` (https://developer.android.com /reference/android/app/Service.html) specified by *`intent`*. See the Specification for intent arguments (#IntentSpec). Options are: • `--user` *`user_id`* `| current`: Specify which user to run as; if not specified, then run as the current user.
`force-stop` *`package`*	Force stop everything associated with *`package`* (the app's package name).
`kill [`*`options`*`]` *`package`*	Kill all processes associated with *`package`* (the app's package name). This command kills only processes that are safe to kill and that will not impact the user experience. Options are: • `--user` *`user_id`* `| all | current`: Specify user whose processes to kill; all users if not specified.
`kill-all`	Kill all background processes.
`broadcast [`*`options`*`]` *`intent`*	Issue a broadcast intent. See the Specification for intent arguments (#IntentSpec). Options are: • `[--user` *`user_id`* `| all | current]`: Specify which user to send to; if not specified then send to all users.
`instrument [`*`options`*`]` *`component`*	Start monitoring with an `Instrumentation` (https://developer.android.com/reference/android /app/Instrumentation.html) instance. Typically the target *`component`* is the form *`test_package`*`/`*`runner_class`*. Options are:

This site uses cookies to store your preferences for site-specific language and display options OK

<table>
<tr><td></td><td>

report_key_streamresult). Use with [*-e perf true*] to generate raw output for performance measurements.

- *-e* *name value*: Set argument *name* to *value*. For test runners a common form is *-e* *testrunner_flag value*[*,value...*].

- *-p* *file*: Write profiling data to *file*.

- *-w*: Wait for instrumentation to finish before returning. Required for test runners.

- *--no-window-animation*: Turn off window animations while running.

- *--user* *user_id* | current: Specify which user instrumentation runs in; current user if not specified.

</td></tr>
<tr><td>

`profile start` *`process file`*

</td><td>

Start profiler on *process*, write results to *file*.

</td></tr>
<tr><td>

`profile stop` *`process`*

</td><td>

Stop profiler on *process*.

</td></tr>
<tr><td>

`dumpheap [`*`options`*`]` *`process file`*

</td><td>

Dump the heap of *process*, write to *file*.
Options are:

- *--user* [*user_id* | current]: When supplying a process name, specify user of process to dump; uses current user if not specified.

- *-n*: Dump native heap instead of managed heap.

</td></tr>
<tr><td>

`set-debug-app [`*`options`*`]` *`package`*

</td><td>

Set application *package* to debug.
Options are:

- *-w*: Wait for debugger when application starts.

- *--persistent*: Retain this value.

</td></tr>
<tr><td>

`clear-debug-app`

</td><td>

Clear the package previous set for debugging with `set-debug-app`.

</td></tr>
<tr><td>

`monitor [`*`options`*`]`

</td><td>

Start monitoring for crashes or ANRs.
Options are:

- *--gdb*: Start gdbserv on the given port at crash/ANR.

</td></tr>
</table>

`screen-compat {on | off}` `package`	Control screen compatibility (https://developer.android.com /guide/practices/screen-compat-mode.html) mode of `package`.
`display-size [reset | widthxheight]`	Override device display size. This command is helpful for testing your app across different screen sizes by mimicking a small screen resolution using a device with a large screen, and vice versa. Example: `am display-size 1280x800`
`display-density dpi`	Override device display density. This command is helpful for testing your app across different screen densities on high-density screen environment using a low density screen, and vice versa. Example: `am display-density 480`
`to-uri intent`	Print the given intent specification as a URI. See the Specification for intent arguments (#IntentSpec).
`to-intent-uri intent`	Print the given intent specification as an `intent:` URI. See the Specification for intent arguments (#IntentSpec).

Specification for intent arguments

For activity manager commands that take an `intent` argument, you can specify the intent with the following options:

Show all

`-a action`

Specify the intent action, such as `android.intent.action.VIEW`. You can declare this only once.

`-d data_uri`

Specify the intent data URI, such as `content://contacts/people/1`. You can declare this only once.

`-t mime_type`

Specify the intent MIME type, such as `image/png`. You can declare this only once.

This site uses cookies to store your preferences for site-specific language and display options. OK

-c *category*

Specify an intent category, such as `android.intent.category.APP_CONTACTS`.

-n *component*

Specify the component name with package name prefix to create an explicit intent, such as `com.example.app/.ExampleActivity`.

-f *flags*

Add flags to the intent, as supported by `setFlags()` `(https://developer.android.com/reference/android/content/Intent.html#setFlags(int))`.

--esn *extra_key*

Add a null extra. This option is not supported for URI intents.

-e | --es *extra_key extra_string_value*

Add string data as a key-value pair.

--ez *extra_key extra_boolean_value*

Add boolean data as a key-value pair.

--ei *extra_key extra_int_value*

Add integer data as a key-value pair.

--el *extra_key extra_long_value*

Add long data as a key-value pair.

--ef *extra_key extra_float_value*

Add float data as a key-value pair.

--eu *extra_key extra_uri_value*

Add URI data as a key-value pair.

--ecn *extra_key extra_component_name_value*

This site uses cookies to store your preferences for site-specific language and display options. OK

(https://developer.android.com/reference/android/content/ComponentName.html) object.

`--eia` *extra_key extra_int_value*`[,`*extra_int_value*`...]`

> Add an array of integers.

`--ela` *extra_key extra_long_value*`[,`*extra_long_value*`...]`

> Add an array of longs.

`--efa` *extra_key extra_float_value*`[,`*extra_float_value*`...]`

> Add an array of floats.

`--grant-read-uri-permission`

> Include the flag `FLAG_GRANT_READ_URI_PERMISSION` (https://developer.android.com/reference
> /android/content/Intent.html#FLAG_GRANT_READ_URI_PERMISSION).

`--grant-write-uri-permission`

> Include the flag `FLAG_GRANT_WRITE_URI_PERMISSION` (https://developer.android.com/reference
> /android/content/Intent.html#FLAG_GRANT_WRITE_URI_PERMISSION).

`--debug-log-resolution`

> Include the flag `FLAG_DEBUG_LOG_RESOLUTION` (https://developer.android.com/reference/android
> /content/Intent.html#FLAG_DEBUG_LOG_RESOLUTION).

`--exclude-stopped-packages`

> Include the flag `FLAG_EXCLUDE_STOPPED_PACKAGES` (https://developer.android.com/reference
> /android/content/Intent.html#FLAG_EXCLUDE_STOPPED_PACKAGES).

`--include-stopped-packages`

> Include the flag `FLAG_INCLUDE_STOPPED_PACKAGES` (https://developer.android.com/reference
> /android/content/Intent.html#FLAG_INCLUDE_STOPPED_PACKAGES).

`--activity-brought-to-front`

> Include the flag `FLAG_ACTIVITY_BROUGHT_TO_FRONT` (https://developer.android.com/reference

This site uses cookies to store your preferences for site-specific language and display options. OK

--activity-clear-top

Include the flag `FLAG_ACTIVITY_CLEAR_TOP` (https://developer.android.com /content/Intent.html#FLAG_ACTIVITY_CLEAR_TOP).

--activity-clear-when-task-reset

Include the flag `FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET` (https://developer.android.com /reference/android/content/Intent.html#FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET).

--activity-exclude-from-recents

Include the flag `FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS` (https://developer.android.com /reference/android/content/Intent.html#FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS).

--activity-launched-from-history

Include the flag `FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY` (https://developer.android.com /reference/android/content/Intent.html#FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY).

--activity-multiple-task

Include the flag `FLAG_ACTIVITY_MULTIPLE_TASK` (https://developer.android.com/reference /android/content/Intent.html#FLAG_ACTIVITY_MULTIPLE_TASK).

--activity-no-animation

Include the flag `FLAG_ACTIVITY_NO_ANIMATION` (https://developer.android.com/reference/android /content/Intent.html#FLAG_ACTIVITY_NO_ANIMATION).

--activity-no-history

Include the flag `FLAG_ACTIVITY_NO_HISTORY` (https://developer.android.com/reference/android /content/Intent.html#FLAG_ACTIVITY_NO_HISTORY).

--activity-no-user-action

Include the flag `FLAG_ACTIVITY_NO_USER_ACTION` (https://developer.android.com/reference /android/content/Intent.html#FLAG_ACTIVITY_NO_USER_ACTION).

--activity-previous-is-top

This site uses cookies to store your preferences for site-specific language and display options. OK

`--activity-reorder-to-front`

Include the flag `FLAG_ACTIVITY_REORDER_TO_FRONT` `(https://developer.android.com/reference` `/android/content/Intent.html#FLAG_ACTIVITY_REORDER_TO_FRONT)`.

`--activity-reset-task-if-needed`

Include the flag `FLAG_ACTIVITY_RESET_TASK_IF_NEEDED` `(https://developer.android.com` `/reference/android/content/Intent.html#FLAG_ACTIVITY_RESET_TASK_IF_NEEDED)`.

`--activity-single-top`

Include the flag `FLAG_ACTIVITY_SINGLE_TOP` `(https://developer.android.com/reference/android` `/content/Intent.html#FLAG_ACTIVITY_SINGLE_TOP)`.

`--activity-clear-task`

Include the flag `FLAG_ACTIVITY_CLEAR_TASK` `(https://developer.android.com/reference/android` `/content/Intent.html#FLAG_ACTIVITY_CLEAR_TASK)`.

`--activity-task-on-home`

Include the flag `FLAG_ACTIVITY_TASK_ON_HOME` `(https://developer.android.com/reference/android` `/content/Intent.html#FLAG_ACTIVITY_TASK_ON_HOME)`.

`--receiver-registered-only`

Include the flag `FLAG_RECEIVER_REGISTERED_ONLY` `(https://developer.android.com/reference` `/android/content/Intent.html#FLAG_RECEIVER_REGISTERED_ONLY)`.

`--receiver-replace-pending`

Include the flag `FLAG_RECEIVER_REPLACE_PENDING` `(https://developer.android.com/reference` `/android/content/Intent.html#FLAG_RECEIVER_REPLACE_PENDING)`.

`--selector`

Requires the use of `-d` and `-t` options to set the intent data and type.

URI component package

You can directly specify a URI, package name, and component name when not qualified by one of

This site uses cookies to store your preferences for site-specific language and display options. OK

slash); otherwise it assumes the argument is a package name.

Call package manager (pm)

Within an adb shell, you can issue commands with the package manager (`pm`) tool to perform actions and queries on application packages installed on the device. While in a shell, the syntax is:

```
pm command
```

You can also issue a package manager command directly from adb without entering a remote shell. For example:

```
adb shell pm uninstall com.example.MyApp
```

Table 3. Available package manager commands.

Command	Description
`list packages [options] filter`	Prints all packages, optionally only those whose package name contains the text in `filter`. Options: • `-f`: See their associated file. • `-d`: Filter to only show disabled packages. • `-e`: Filter to only show enabled packages. • `-s`: Filter to only show system packages. • `-3`: Filter to only show third party packages. • `-i`: See the installer for the packages. • `-u`: Also include uninstalled packages. • `--user user_id`: The user space to query.

This site uses cookies to store your preferences for site-specific language and display options.   OK

`list permissions [options] group`	Prints all known permissions, optionally only those in *group*. Options: • `-g`: Organize by group. • `-f`: Print all information. • `-s`: Short summary. • `-d`: Only list dangerous permissions. • `-u`: List only the permissions users will see.
`list instrumentation [options]`	List all test packages. Options: • `-f`: List the APK file for the test package. • `target_package`: List test packages for only this app.
`list features`	Prints all features of the system.
`list libraries`	Prints all the libraries supported by the current device.
`list users`	Prints all users on the system.
`path package`	Print the path to the APK of the given *package*.
`install [options] path`	Installs a package (specified by *path*) to the system. Options: • `-l`: Install the package with forward lock. • `-r`: Reinstall an existing app, keeping its data. • `-t`: Allow test APKs to be installed. • `-i installer_package_name`:

This site uses cookies to store your preferences for site-specific language and display options.   OK

	• `-s`: Install package on the shared mass storage (such as sdcard). • `-f`: Install package on the internal system memory. • `-d`: Allow version code downgrade. • `-g`: Grant all permissions listed in the app manifest.
`uninstall [`*`options`*`]` *`package`*	Removes a package from the system. Options: • `-k`: Keep the data and cache directories around after package removal.
`clear` *`package`*	Deletes all data associated with a package.
`enable` *`package_or_component`*	Enable the given package or component (written as "package/class").
`disable` *`package_or_component`*	Disable the given package or component (written as "package/class").
`disable-user [`*`options`*`]` *`package_or_component`*	Options: • `--user` *`user_id`*: The user to disable.
`grant` *`package_name permission`*	Grant a permission to an app. On devices running Android 6.0 (API level 23) and higher, the permission can be any permission declared in the app manifest. On devices running Android 5.1 (API level 22) and lower, must be an optional permission defined by the app.
`revoke` *`package_name permission`*	Revoke a permission from an app. On devices running Android 6.0 (API level 23) and higher, the permission can be any permission declared in the app manifest. On devices running Android 5.1 (API level 22) and lower, must be an optional

This site uses cookies to store your preferences for site-specific language and display options.   OK

set-install-location *location*	Changes the default install location. Location values: • `0`: Auto: Let system decide the best location. • `1`: Internal: install on internal device storage. • `2`: External: on external media. **Note:** This is only intended for debugging; using this can cause applications to break and other undesireable behavior.
get-install-location	Returns the current install location. Return values: • `0 [auto]`: Lets system decide the best location • `1 [internal]`: Installs on internal device storage • `2 [external]`: Installs on external media
set-permission-enforced *permission* [true | false]	Specifies whether the given permission should be enforced.
trim-caches *desired_free_space*	Trim cache files to reach the given free space.
create-user *user_name*	Create a new user with the given *user_name*, printing the new user identifier of the user.
remove-user *user_id*	Remove the user with the given *user_id*, deleting all data associated with that user
get-max-users	Prints the maximum number of users supported by the device.

Take a screenshot

This site uses cookies to store your preferences for site-specific language and display options.

OK

The `screencap` command is a shell utility for taking a screenshot of a device display. While in a shell, the syntax is:

```
screencap filename
```

To use the `screencap` from the command line, type the following:

```
adb shell screencap /sdcard/screen.png
```

Here's an example screenshot session, using the adb shell to capture the screenshot and the `pull` command to download the file from the device:

```
$ adb shell
shell@ $ screencap /sdcard/screen.png
shell@ $ exit
$ adb pull /sdcard/screen.png
```

Record a video

The `screenrecord` command is a shell utility for recording the display of devices running Android 4.4 (API level 19) and higher. The utility records screen activity to an MPEG-4 file.

> **Note:** Audio is not recorded with the video file.

A developer can use this file to create promotional or training videos. While in a shell, the syntax is:

```
screenrecord [options] filename
```

To use `screenrecord` from the command line, type the following:

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

Stop the screen recording by pressing Control + C, otherwise the recording stops automatically at three minutes or the time limit set by `--time-limit`.

To begin recording your device screen, run the `screenrecord` command to record the video. Then, run the `pull` command to download the video from the device to the host computer. Here's an example recording session:

```
$ adb shell
```

```
(press Control + C to stop)
shell@ $ exit
$ adb pull /sdcard/demo.mp4
```

The `screenrecord` utility can record at any supported resolution and bit rate you request, while retaining the aspect ratio of the device display. The utility records at the native display resolution and orientation by default, with a maximum length of three minutes.

There are some known limitations of the `screenrecord` utility that you should be aware of when using it:

- Some devices might not be able to record at their native display resolution. If you encounter problems with screen recording, try using a lower screen resolution.

- Rotation of the screen during recording is not supported. If the screen does rotate during recording, some of the screen is cut off in the recording.

Table 4. `screenrecord` options

Options	Description
`--help`	Displays command syntax and options
`--size `*`width`*`x`*`height`*	Sets the video size: `1280x720`. The default value is the device's native display resolution (if supported), 1280x720 if not. For best results, use a size supported by your device's Advanced Video Coding (AVC) encoder.
`--bit-rate `*`rate`*	Sets the video bit rate for the video, in megabits per second. The default value is 4Mbps. You can increase the bit rate to improve video quality, but doing so results in larger movie files. The following example sets the recording bit rate to 6Mbps: ```screenrecord --bit-rate 6000000 /sdcard/demo.mp4```
`--time-limit `*`time`*	Sets the maximum recording time, in seconds. The default and maximum value is 180 (3 minutes).
`--rotate`	Rotates the output 90 degrees. This feature is experimental.
`--verbose`	Displays log information on the command-line screen. If you do not set this option, the utility does not display any information while running.

Read ART profiles for apps

Starting in Android 7.0 (API level 24) the Android Runtime (ART) collects execution profiles for installed apps, which are used to optimize app performance. You might want to examine the collected profiles to

This site uses cookies to store your preferences for site-specific language and display options.   OK

app startup.

To produce a text form of the profile information, use the command:

```
adb shell cmd package dump-profiles package
```

To retrieve the file produced, use:

```
adb pull /data/misc/profman/package.txt
```

Other shell commands

For a list of all the available shell programs, use the following command:

```
adb shell ls /system/bin
```

Help is available for most of the commands.

Table 5 lists some of the more common adb shell commands.

Table 5. Some other adb shell commands

Shell Command	Description	Comments
dumpsys	Dumps system data to the screen.	The Dalvik Debug Monitor Server (https://developer.android.com/studio/profile/ddms.html) (DDMS) tool offers an integrated debug environment that you might find easier to use.
dumpstate	Dumps state to a file.	
logcat [option]... [filter-spec]...	Enables system and app logging and prints output to the screen. See also Logcat Command-Line Tool (https://developer.android.com/studio/command-line/logcat.html).	
dmesg	Prints kernel debugging messages to the screen.	
start	Starts (restarts) a device.	
stop	Stops execution of a device.	

This site uses cookies to store your preferences for site-specific language and display options. OK