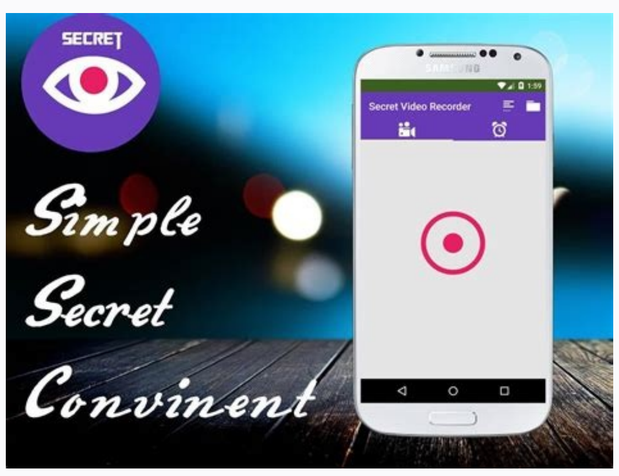
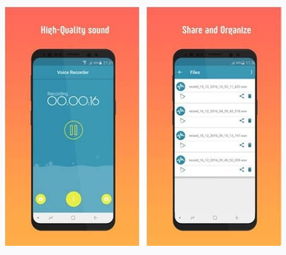
Best hidden call recorder app for android

Continue





Best offline call recording app for android. Best hidden call recorder app for android quora. Best secret call recording app for android. Best call recorder app for android phone. Best call recorder app for android free. Best hidden call recorder for android. Best auto call recorder hidden app for android.

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a 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, which installs it at android_sdk/platform-tools/. Or if you want the standalone Android SDK Platform-Tools package, you can download it here. For information on connecting a device for use over ADB, including how to use the Connection Assistant to troubleshoot common problems, see Run apps on a hardware device. 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. To use adb with a device connected over Wi-Fi, see Connect to a device over Wi-Fi. 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. Connect to a device over Wi-Fi (Android 11+) Note: The instructions below do not apply to Wear devices running Android 11. See the guide to debugging a Wear OS app for more information. Android 11 and higher supports deploying and debugging your app wirelessly from your workstation using Android Debug Bridge (adb). For example, you can deploy your debuggable app to multiple remote devices without physically connecting your device via USB. This eliminates the need to deal with common USB connection issues, such as driver installation. Before you begin using wireless debugging, you must complete the following steps: Ensure that your workstation and device are connected to the same wireless network. Ensure that your device is running Android 11 or higher. For more informaton, see Check & update your Android version. Ensure that you have Android Studio Bumblebee. You can download it here. On your workstation, update to the latest version of the SDK Platform-Tools. To use wireless debugging, you must pair your device to your workstation using a QR Code or a pairing code. Your workstation and device must be connected to the same wireless network. To connect to your device, follow these steps: Enable developer options on your device: On your device, find the Build number option. You can find this in these locations for the following devices: Device Setting Google Pixel Settings > About phone > Build number Samsung Galaxy S8 and later Settings > About phone > Software information > Build number LG G6 and later Settings > About phone > Software info > Build number HTC U11 and later Settings > About > Software information > More > Build number or Settings > System > About phone > Software information > More > Build number OnePlus 5T and later Settings > About phone > Build number Tap the Build Number option seven times until you see the message You are now a developer! This enables developer options on your phone. Enable debugging over Wi-Fi on your device: On your device, find Developer options. You can find this option in these locations for the following devices: Device Setting Google Pixel, OnePlus 5T and later Settings > System > Developer options Samsung Galaxy S8 and later, LG G6 and later, HTC U11 and later Settings > Developer options In Developer options, scroll down to the Debugging section and turn on Wireless debugging. On the Allow wireless debugging on this network? popup, select Allow. Open Android Studio and select Pair Devices Using Wi-Fi from the run configurations dropdown menu. Figure 1. Run configurations dropdown menu. The Pair devices over Wi-Fi window pops up, as shown below. Figure 2. Popup window to pair devices using QR code or pairing code On your device, tap on Wireless debugging and pair your device: Figure 3. Screenshot of the Wireless debugging setting on a Google Pixel phone. To pair your device with a QR code, select Pair device with QR code and scan the QR code obtained from the Pair devices over Wi-Fi popup above. To pair your device with a pairing code, select Pair device with pairing code from the Pair devices over Wi-Fi popup above. On your device, select Pair using pairing code and take note of the six digit pin code. Once your device appears on the Pair devices over Wi-Fi window, you can select Pair and enter the six digit pin code shown on your device. Figure 4. Example of six digit pin code entry. After you are paired, you can attempt to deploy your app to your device. To pair a different device or to forget this device on your workstation, navigate to Wireless debugging on your device, tap on your workstation name under Paired devices, and select Forget. If you want to quickly turn on and off wireless debugging, you can utilize the Quick settings developer tiles for Wireless debugging, found in Developer Options > Quick settings developer tiles. Figure 5. The Quick settings developer tiles setting allows you to quickly turn wireless debugging on and off. Alternatively, to connect to your device via command line without Android Studio, follow these steps: Enable developer options on your device, as described above. Enable Wireless debugging on your device, as described above. On your workstation, open a terminal window and navigate to android_sdk/platform-tools. Find your IP address, port number, and pairing code by selecting Pair device with pairing code. Take note of the IP address, port number, and pairing code displayed on the device. On your workstation's terminal, run adb pair ipaddr:port. Use the IP address and port number from above. When prompted, enter the pairing code, as shown below. Figure 6. A message indicates that your device has been successfully paired. Resolve wireless connection issues If you are having issues connecting to your device wirelessly, you can try the following troubleshooting steps to resolve the issue. Check if your workstation and device meet the prerequisites To meet the prerequisites for wireless debugging, ensure that: Your workstation and device are connected to the same wireless network. Your device is running Android 11 or higher. For more information, see Check & update your Android version. You have Android Studio Bumblebee. You can download it here. You have the latest version of the SDK Platform Tools on your workstation. Check for other known issues The following is a list of current known issues with wireless debugging in Android Studio and how to resolve them. Wi-Fi is not connecting: Some Wi-Fi networks, such as corporate Wi-Fi networks, may block p2p connections and not allow you to connect over Wi-Fi.Try connecting with a cable or another Wi-Fi network. ADB over Wi-Fi sometimes turns off automatically: This can happen if the device either switches Wi-Fi networks or disconnects from the network. Connect to a device over Wi-Fi (Android 10 and lower) Note: The instructions below do not apply to Wear devices running Android 10 (or lower). See the guide to debugging a Wear OS app for more information. adb usually communicates with the device over USB, but you can also use adb over Wi-Fi. To connect a device running Android 10 or lower, there are some initial steps you must do over USB, as described below: 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. If you are connecting to a Wear OS device, turn off Bluetooth on the phone that's paired with the device. Connect the device to the host computer with a USB cable. Set the target device to listen for a TCP/IP connection on port 5555. adb tcpip 5555 Disconnect the USB cable from the target device. 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 a Wear OS device, you can find the IP address at Settings > Wi-Fi Settings > Advanced > IP address. Connect to the device by its IP address. adb connect device_ip_address:5555 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! If the adb connection is ever lost: Make sure that your host is still connected to the same Wi-Fi network your Android device is. Reconnect by executing the adb connect step again. 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 hardware device that is attached to the computer. $ 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 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: The adb server is not running, and You use the emulator command with the -port or -ports option with an odd-numbered port value between 5554 and 5584, and 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 You start the adb server after you start the emulator. One way to avoid this situation is to 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 the emulator command and before using the adb devices command, as follows: $ 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. 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, 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 You can use adb to install an APK on an emulator or connected device with the install command: adb install path_to_apk You must use the -t option with the install command when you install a test APK. For more information, see -t. For more information about how to create an APK file that you can install on an emulator/device instance, see Build and Run Your App. Note that, if you are using Android Studio, you do not need to use adb directly to install your app on the emulator/device. Instead, Android Studio handles the packaging and installation of the app for you. 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 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. Issuing adb commands 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. You can see a detailed list of all supported adb commands using the following command: adb --help Issue shell commands You can use the shell command to issue device commands through adb, or to start an interactive shell. To issue a single command use the shell command like this: adb [-d |-e | -s serial_number] shell shell_command To start an interactive shell on a device use the shell command like this: adb [-d | -e | -s serial_number] shell To exit an interactive shell, press Control + D or type exit. 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 and makes it possible to now safely execute commands that contain shell metacharacters, such as adb install Let\'sGo.apk. But, this change means that the interpretation of any command that contains shell metacharacters has also changed. For example, the adb shell setprop foo 'a 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'". Android provides most of the usual Unix command-line tools. For a list of available tools, use the following command: adb shell ls /system/bin Help is available for most of the commands via the --help argument. Many of the shell commands are provided by toybox. General help applicable to all toybox commands is available via toybox --help. See also Logcat Command-Line Tool which is useful for monitoring the system log. 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 specified by intent. See the Specification for intent arguments. 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 activity. --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 specified by intent. See the Specification for intent arguments. 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. 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 instance. Typically the target component is the form test_package/runner_class. Options are: -r: Print raw results (otherwise decode 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. profile start process file Start profiler on process, write results to file. profile stop process Stop profiler on process. dumpheap [options] process file 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. set-debug-app [options] package Set app package to debug. Options are: -w: Wait for debugger when app starts. --persistent: Retain this value. clear-debug-app Clear the package previous set for debugging with set-debug-app. monitor [options] Start monitoring for crashes or ANRs. Options are: --gdb: Start gdbserv on the given port at crash/ANR. screen-compat {on | off} package Control screen compatibility 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. to-intent-uri intent Print the given intent specification as an intent: URI. See the Specification for intent arguments. 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. -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(). --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 Add a component name, which is converted and passed as a ComponentName 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. --grant-write-uri-permission Include the flag FLAG_GRANT_WRITE_URI_PERMISSION. --debug-log-resolution Include the flag FLAG_DEBUG_LOG_RESOLUTION. --exclude-stopped-packages Include the flag FLAG_EXCLUDE_STOPPED_PACKAGES. --include-stopped-packages Include the flag FLAG_INCLUDE_STOPPED_PACKAGES. --activity-brought-to-front Include the flag FLAG_ACTIVITY_BROUGHT_TO_FRONT. --activity-clear-top Include the flag FLAG_ACTIVITY_CLEAR_TOP. --activity-clear-when-task-reset Include the flag FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET. --activity-exclude-from-recents Include the flag FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS. --activity-launched-from-history Include the flag FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY. --activity-multiple-task Include the flag FLAG_ACTIVITY_MULTIPLE_TASK. --activity-no-animation Include the flag FLAG_ACTIVITY_NO_ANIMATION. --activity-no-history Include the flag FLAG_ACTIVITY_NO_HISTORY. --activity-no-user-action Include the flag FLAG_ACTIVITY_NO_USER_ACTION. --activity-previous-is-top Include the flag FLAG_ACTIVITY_PREVIOUS_IS_TOP. --activity-reorder-to-front Include the flag FLAG_ACTIVITY_REORDER_TO_FRONT. --activity-reset-task-if-needed Include the flag FLAG_ACTIVITY_RESET_TASK_IF_NEEDED. --activity-single-top Include the flag FLAG_ACTIVITY_SINGLE_TOP. --activity-clear-task Include the flag FLAG_ACTIVITY_CLEAR_TASK. --activity-task-on-home Include the flag FLAG_ACTIVITY_TASK_ON_HOME. --receiver-registered-only Include the flag FLAG_RECEIVER_REGISTERED_ONLY. --receiver-replace-pending Include the flag 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 the above options. When an argument is unqualified, the tool assumes the argument is a URI if it contains a ":" (colon); it assumes the argument is a component name if it contains a "/" (forward-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 app 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. list permission-groups Prints all known permission groups. 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: -r: Reinstall an existing app, keeping its data. -t: Allow test APKs to be installed. Gradle generates a test APK when you have only run or debugged your app or have used the Android Studio Build > Build APK command. If the APK is built using a developer preview SDK (if the targetSdkVersion is a letter instead of a number), you must include the -t option with the install command if you are installing a test APK. -i installer_package_name: Specify the installer package name. --install-location location: Sets the install location using one of the following values: 0: Use the default install location 1: Install on internal device storage 2: Install on external media -f: Install package on the internal system memory. -d: Allow version code downgrade. -g: Grant all permissions listed in the app manifest. --fastdeploy: Quickly update an installed package by only updating the parts of the APK that changed. --incremental: Installs enough of the APK to launch the app while streaming the remaining data in the background. To use this feature, you must sign the APK, create an APK Signature Scheme v4 file, and place this file in the same directory as the APK. This feature is only supported on certain devices. This option forces adb to use the feature or fail if it is not supported (with verbose information on why it failed). Append the --wait option to wait until the APK is fully installed before granting access to the APK. --no-incremental prevents adb from using this feature. 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 permission defined by the app. 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 apps 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. get-app-links [options] [package] Prints the domain verification state for the given package, or for all packages if none is specified. State codes are defined as follows: none: nothing has been recorded for this domain verified: the domain has been successfully verified approved: force approved, usually through shell denied: force denied, usually through shell migrated: preserved verification from a legacy response restored: preserved verification from a user data restore legacy_failure: rejected by a legacy verifier, unknown reason system_configured: automatically approved by the device config >= 1024: Custom error code which is specific to the device verifier Options are: --user user_id: include user selections (includes all domains, not just autoVerify ones). reset-app-links [options] [package] Resets domain verification state for the given package, or for all packages if none is specified. package: the package to reset, or "all" to reset all packages Options are: --user user_id: include user selections (includes all domains, not just autoVerify ones). verify-app-links [--re-verify] [package] Broadcasts a verification request for the given package, or for all packages if none is specified. Only sends if the package has previously not recorded a response. --re-verify: send even if the package has recorded a response set-app-links [--package package] state domains Manually set the state of a domain for a package. The domain must be declared by the package as autoVerify for this to work. This command will not report a failure for domains that could not be applied. --package package: the package to set, or "all" to set all packages state: the code to set the domains to, valid values are: STATE_NO_RESPONSE (0): reset as if no response was ever recorded. STATE_SUCCESS (1): treat domain as successfully verified by domain verification agent. Note that the domain verification agent can override this. STATE_APPROVED (2): treat domain as always approved, preventing the domain verification agent from changing it. STATE_DENIED (3): treat domain as always denied, preveting the domain verification agent from changing it. domains: space separated list of domains to change, or "all" to change every domain. set-app-links-user-selection --user user_id [--package package] enabled domains Manually set the state of a host user selection for a package. The domain must be declared by the package for this to work. This command will not report a failure for domains that could not be applied. --user user_id: the user to change selections for --package package/code>: the package to set< enabled: whether or not to approve the domain domains: space separated list of domains to change, or "all" to change every domain. set-app-links-user-selection --user user_id [--package package] enabled domains Manually set the state of a host user selection for a package. The domain must be declared by the package for this to work. This command will not report a failure for domains that could not be applied. --user user_id: the user to change selections for --package package: the package to set enabled: whether or not to approve the domain domains: space separated list of domains to change, or "all" to change every domain. set-app-links-allowed --user user_id [--package package] allowed Toggle the auto-verified link-handling setting for a package. --user user_id: the user to change selections for --package package: the package to set, or "all" to set all packages; packages will be reset if no one package is specified. allowed: true to allow the package to open auto-verified links, false to disable get-app-link-owners --user user_id [--package package] domains Print the owners for a specific domain for a given user in low to high priority order. --user user_id: the user to query for --package package: optionally also print for all web domains declared by a package, or "all" to print all packages domains: space separated list of domains to query for Call device policy manager (dpm) To help you develop and test your device management (or other enterprise) apps, you can issue commands to the device policy manager (dpm) tool. Use the tool to control the active admin app or change a policy's status data on the device. While in a shell, the syntax is: dpm command You can also issue a device policy manager command directly from adb without entering a remote shell: adb shell dpm command Table 4. Available device policy manager commands Command Description set-active-admin [options] component Sets component as active admin. Options are: --user user_id: Specify the target user. You can also pass --user current to select the current user. set-profile-owner [options] component Sets component as active admin and its package as profile owner for an existing user. Options are: --user user_id: Specify the target user. You can also pass --user current to select the current user. --name name: Specify the human-readable organization name. set-device-owner [options] component Sets component as active admin and its package as device owner. Options are: --user user_id: Specify the target user. You can also pass --user current to select the current user. --name name: Specify the human-readable organization name. remove-active-admin [options] component Disables an active admin. The app must declare android:testOnly in the manifest. This command also removes device and profile owners. Options are: --user user_id: Specify the target user. You can also pass --user current to select the current user. clear-freeze-period-record Clears the device's record of previously-set freeze periods for system OTA updates. This is useful to avoid the device's scheduling restrictions when developing apps that manage freeze-periods. See Manage system updates. Supported on devices running Android 9.0 (API level 28) and higher. force-network-logs Forces the system to make any existing network logs ready for retrieval by a DPC. If there are connection or DNS logs available, the DPC receives the onNetworkLogsAvailable() callback. See Network activity logging. This command is rate-limited. Supported on devices running Android 9.0 (API level 28) and higher. force-security-logs Forces the system to make any existing security logs available to the DPC. If there are logs available, the DPC receives the onSecurityLogsAvailable() callback. See Log enterprise device activity. This command is rate-limited. Supported on devices running Android 9.0 (API level 28) and higher. Take a screenshot 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. You can use this file to create promotional or training videos or for debugging and testing. In a shell, use the following syntax: 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 (Command + C on Mac); 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 shell@ $ screenrecord --verbose /sdcard/demo.mp4 (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. Limitations of the screenrecord utility: Audio is not recorded with the video file. Video recording is not available for devices running Wear OS. 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 5. screenrecord options Options Description --help Displays command syntax and options --size widthxheight 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 understand which methods are determined to be frequently executed and which classes are used during 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 Reset test devices If you test your app across multiple test devices, it may be useful to reset your device between tests, for example, to remove user data and reset the test environment. You can perform a factory reset of a test device running Android 10 (API level 29) or higher using the testharness adb shell command, as shown below. adb shell cmd testharness enable When restoring the device using testharness, the device automatically backs up the RSA key that allows debugging through the current workstation in a persistent location. That is, after the device is reset, the workstation can continue to debug and issue adb commands to the device without manually registering a new key. Additionally, to help make it easier and more secure to keep testing your app, using the testharness to restore a device also changes the following device settings: The device sets up certain system settings so that initial device setup wizards do not appear. That is, the device enters a state from which you can quickly install, debug, and test your app. Settings: Disables lock screen Disables emergency alerts Disables auto-sync for accounts Disables automatic system updates Other: Disables preinstalled security apps If you app needs to detect and adapt to the default settings of the testharness command, you can use the ActivityManager.isRunningInUserTestHarness(). sqlite sqlite3 starts the sqlite command-line program for examining sqlite databases. It includes commands such as .dump to print the contents of a table, and .schema to print the SQL CREATE statement for an existing table. You can also execute SQLite commands from the command line, as shown below. $ adb -s emulator-5554 shell $ sqlite3 /data/data/com.example.app/databases/rssitems.db SQLite version 3.3.12 Enter ".help" for instructions For more information, see the sqlite3 command line documentation.