# Enable usb debugging using fastboot

Can i enable usb debugging via fastboot.

ADB and FastBoot are different modes. You can insert recovery without USB debugging. Press and hold POWER + VOL. This is the bootloader screen. Now browse to restore with the VOL keys, power to confirm. Wait a moment. When the Android symbol appears, press VOL UP + POWER to enter the recovery. Now make the factory reset. If the device is locked, you may not be able to insert recovery. You can unlock it from FastBoot and try again. The unlock will be restored the Tablet factory. Download platform_tools from Google. Enter the FastBoot mode from the Bootloader screen. Connect the USB cable. Install the drivers. Open the cmd.exe as an administrator. Go to the folder. Check with FastBoot if the serialno is displayed. to unlock. FastBoot FastBoot OEM Unlock devices then start TWRP and see if the partitions can be mounted. At least / system and / cache should be mountable, otherwise something is wrong. / The data is encrypted but must also be mountable. For more information see here. NON TWRP Flash in Restore Partition, First Recovery of Backup Securities! Fastboot boot twrp-3.3.1-0-shieldtablet.img in TWRP, there is an option restarts the restoration that should carry in restore mode. Otherwise, you still have to option to run the option to run the option to back up / delete partitions from TWRP. Start with the cleaning cache. If the phone is not yet started, restart into the TWRP and format data. If an error occurs, you can extract the register from ADB at any time. Publish the log file in the TWRP support thread and ask for help. Try another twrp version (3.1.1-0) Depending on the Android version ADB Pull /Tmp/Recovery.log C: Users Admin Downloads Platform_Tools Recovery.log If you need to enable USB debugging from recovery , you can change / system. (But the other methods connected above should work) Pull the /system/build.prop and change the following rows with note ++ lock (create a copy to backup) persist.service.adb.enable = 1 persist.service.debuggable = 1 persist.sys.usb.config = MTP, ADB Mount / System System, then push it to the phone. If you do not want to change / system you can try pushing it /data/local.prop (but I imagine it is not supported on this rom) or create a /data/property/persist.sys.sb file. Config (I have no exactly read this post above) ADB Push "C: Users Admin Downloads Platform_Tools Build.prop" /system/Build.prop now Place the RSA button somewhere on the phone and join the ADB_KEYS ADB Push "C: Users Admin .Android adbkey.pub" /tmple/adbkey.pub adb shell mkdir -p / data / misc / adb cat /tmp/adbkey.pub >> / data / misc / adb / adb_kyys if Root access is required, it is possible to modify RO.Secure = 0 in build.prop also (but not recommended) Android Debug Bridge (ADB) is a versatile command line tool that allows you to communicate with a device. The ADB command facilitates a variety of device actions, such as app installing and debugging and provides access to a UNIX shell that you can use to perform a variety of commands on a device. It is a client-server program that includes three components: a client, which sends commands. The client works on your development machine. You can call up a client from a command line terminal by issuing an ADB command. A demon (ADBD), which performs commands on a device. The daemon works as a background process on each device. A server, which manages communication between the customer and the daemon. The server runs as a background process on the development machine. ADB is included in the Platform-Tools Android SDK package. You can download this package with the SDK manager, installing it on Android_SDK / Platform-Tools /. Or, You want the Android SDK STD-STR-Tools package, you can download it here. For information on connecting a device for use on ADB, included How to use the connection server to resolve common problems, see App on a hardware device. How ADB works when you start an ADB client, the client checks first if a process of the ADB server is already running. If it is not, the server process starts. When the server starts, binds to the local TCP port 5037 and listen for Sent from ADB clients ... All ADB clients use Port 5037 to communicate with the ADB server. The server therefore sets connections to all execution devices. It identifies the emulars by showing the ports of the odd numbered in the range 5555 to 5585, the range used by the first 16 emulators. Where the server finds a Daemon ADB (ADBD), set a connection to that port. Note that each emulator uses a pair of sequential doors - a uniform numbered port for console connections and an odd port for ADB connections. For example: emulator 1, console: emulator 5554 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 The emulator whose console listens to port 5554. Once the server has configured connections to all devices, you can use ADB commands to access these devices. Because the server manages the connections to the devices and handle commands from plus ADB client, you can control any device from any client (or a script). Enable ADB debugging on the device to use ADB with a device connected via USB, you need to enable USB debugging in device system settings, in developer options. To use ADB with a device connected via Wi-Fi, see Connecting to a device via Wi-Fi. On Android 4.2 and higher, the Developer Options screen is hidden by default. To make it visible, go to Settings> Phone information and tap Build number seven times. Back to the previous screen to find the developer options at the bottom. On some devices, the Developer Options screen may be positioned or named differently. Now you can connect your device with USB. You can check that the device is connected by performing ADB devices from Android_SDK / Platform-Tools / Directory. If connected, you will see the name of the device listed as a "device". Note: When connecting a device running Android 4.2.2 or higher, the system shows a dialog box that requires whether to accept a RSA button that allows you to debug through this computer. This safety mechanism protects user devices because it guarantees that the USB debugging and other ADB commands cannot be performed unless you can unlock the device and recognize the dialog box. For more information on connecting to a USB device, read the apps run on a hardware device. Connect to a device via Wi-Fi (Android 11+) Android 11 and superior support distribution and debugging of your wireless app from your workstation using Android Debug Bridge (ADB). For example, you can distribute your DeboAable app on multiple remote devices without physically connecting the device via USB. This eliminates the need to manage common USB connection problems, such as the driver installation. To use wireless debugging, you need to match the device to the workstation using a coupling code. The workstation and the device must be connected to the same wireless network. To connect to the device, follow these steps: Figure 1. Wireless ADB Coupling dialog. In your workstation, update the latest version of SDK platform tools. On the device, enable developer options. Enable the wireless debugging option. In the dialog asked to allow wireless debugging on this network?, Click Allow. Select the coupling device with the coupling code. Take note of the coupling code, IP address and port number displayed on the device (see image). On the workstation, open a terminal and browse up / Platform-tools. Run ADB Pair iPadDR: door. Use the IP address and port number from step 5. When prompted, enter the coupling code received in step 5. A message indicates that the device has been successfully coupled. Enter the coupling code: 482924 Coupled correctly to 192.168.1.130:37099 [GUID = ADB-235XY] (only for Linux or Microsoft Windows) Run ADB Connect iPadDR: door. Use the IP address and door under wireless debugging. Figure 2. IP wireless ADB and port number. ADB usually communicates with the device on USB, but you can also use ADB on Wi-Fi provided the following: To connect to a device running Android 11 (and later), see see Wi-Fi section in running applications on a hardware device. To connect to a device that runs previous versions of Android, there are some initial steps you have to do via USB. These steps are described below. If you are developing for OS Wear, refer to the Debug Guide of an App Wear OS, which has special instructions for using ADB with Wi-Fi and Bluetooth. Connect the Android device and ADB computer host to a common accessible Wi-Fi network. Attention that not all access points are suitable; You may need to use an access point whose firewall is correctly configured for ADB support. If you connect to a Wear OS device, turn off Bluetooth on your mobile phone that is coupled with the device. Connect the device to the host computer via a USB cable. Set the destination device to listen for a TCP / IP connection on port 5555. ADB TCPIP 5555 Disconnect the USB cable from the destination device. Find the IP address of the Android device. For example, on a Nexus device, you can find the IP address in Settings> Tablet information (or phone info)> Status> IP address. Or, on a Wear OS device, you can find the IP address in Settings> Wi-Fi settings> Advanced address> IP. Connect the device from your IP address. adb device_ip_address connect: 5555 Verify that the host computer is connected to the target device: device_ip_address devices $ adb list of connected devices: 5555 device now six to leave! If the ADB connection is never lost: make sure your guest is still connected to the same Wi-Fi network of the Android device is. Reconnect by performing the ADB connection phase again. Or, if this doesn't work, restore your ADB guest: ADB Kill-Server then start again from the beginning. Query for devices Before issuing ADB commands, it is useful to know which devices instances are connected to the ADB server. You can generate a list of devices connected with the device control. 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 with its port number. Here is a serial example example: emulator-5554 status: the connection status of the device can be one of the following: offline: the device is not connected to the ADB or does not respond. Device: The device is now connected to the ADB server. Note that this state does not imply that the Android system is completely started and operational, because the device connects to ADB while the system is still starting. However, after startup, this is the normal state of operation of a device. No device: there is no connected device. Description: If you include the -L option, the Devices command tells you what the device is. This information is useful when you have more connected devices so you can distinguish them. The following example shows the devices and its output controls. There are three devices in â €