Click to verify

Terminal emulator for Android Not to be confused with Tmux. Termuxnano running on TermuxOriginal author(s)Fredrik FornwallInitial release30 May 2015; 9 years ago (2015-05-30)Stable release0.118.2[1] (29 March 2025; 45 days ago (29 March 2025)) Repository inJava, C, C++Operating systemAndroidPlatformx86-64, ARM64, i686, ARMv7Size105 MBTypeTerminal emulator, command-line interfaceLicenseGPLv3 onlyWebsitetermux.dev Termux is a free and open-source terminal emulator for Android which allows for running a Linux environment on an Android device. Termux installs a minimal base system automatically; additional packages are available using its package manager, based on Debian's.[2] Most commands available in Linux are accessible in Termux, as well as built-in Bash commands. There are several other shells available, such as Zsh and tcsh.[3] Packages are cross-compiled with Android NDK and have compatibility patches to get them working on Android. Since all files are installed in the application directory, rooting is not required.[4] There are more than one thousand packages available, and users can submit requests for new ones. Alternatively, packages can be compiled from source, as Termux supports a variety of build tools including CMake, Meson, GNU Autotools, as well as compilers for C++, Rust, Go, Swift, and other programming languages. Termux can also install interpreters for languages like Ruby, Python, and JavaScript. Terminal-based text editors such as Emacs and Vim can be installed. It is also possible to execute GUI applications in Termux by using a VNC server and installing a desktop environment (Xfce, LXQt, MATE) or window manager.[5] Another popular use case for Termux is its ability to run containers of various Linux distributions which would normally use the chroot command without requiring root privileges by using PRoot.[6] This provides access to many more packages through the package manager of the distribution of choice than would directly be accessible in Termux, though popular application container formats such as Flatpak and Snap cannot be used without root access as they require filesystem mounting permissions. Termux's user interface is fairly simple, only displaying the extra keys row and the terminal output. Color scheme and font can be changed through Termux: Styling. The extra-keys row can also be customized. Users can add more function keys and controls by editing ~/.termux/termux.properties. Termux has mouse/touch support which can be used to interact with programs such as htop and other ncurses-based applications. Scrolling is done by swiping up or down in the terminal buffer. Users configure Termux by editing ~/.termux/termux.properties. Termux also includes 7 add-ons: Termux:API: exposes Android functionality to CLI applications Termux:Styling: allows changing the color scheme and font of the terminal Termux:Boot: executes Termux commands at boot Termux:GUI: allows for some Termux apps to have a GUI using default Android resources; does not work with X11/Wayland apps Termux:Widget: lets users run scripts in a dedicated widget or a shortcut in the Home screen Termux:Float: runs terminal session in a floating window Termux:Tasker: integrates Tasker with Termux Add-ons must be installed from the same source as the application so that the same User ID is used. Termux was initially released in 2015. Support for requesting packages and features was added through GitHub issues in the app's repository. People can also contribute to the project by adding new features and packages. In January 2020, the Termux development team ended support for devices running Android 5-6, making Android version 7 the minimum OS requirement. Termux v0.101 was the last version to be updated in the Google Play Store. Since November 2020, Google Play has enforced apps targeting API level 29, which breaks the execution of binaries in private application directories. According to Google:Untrusted apps that target Android 10 cannot invoke exec() on files within the app's home directory. This execution of files from the writable app home directory is a W^X violation. Apps should load only the binary code that's embedded within an app's APK file.[7] The Termux development team suggests moving to F-Droid in order to continue getting updates, as F-Droid does not impose such restrictions. It is also possible to download APK files from the project's GitHub repository. In May 2021, Bintray, which had been the primary host for the Termux packages, shut down its services.[8] Termux migrated to Hetzner, another hosting service.[9] The installation process extracts the bootstrap archive from the APK file, sets correct permissions for the executable, and sets up directories like the home directory. The Play Store version of Termux is not being updated anymore; users are encouraged to install Termux from F-Droid or GitHub in order to receive the latest updates. Packages in Termux are installed through the application's package manager (pkg) and use the .deb format. However, normal Debian packages cannot be installed as Termux is not FHS compliant.[10] Users can also build and submit packages. Termux had more than 1000 packages available as of 2021. Termux has three repositories available. Repositories included in the default Termux bootstrap installation include: main is the main repository containing all CLI utilities and other popular Linux tools and language compilers/interpreters. x11-repo contains X11-based packages and graphical applications. root-repo contains packages useful for rooted devices. Some of these packages can be used without root, but functionality may be limited. ^ "Release v0.118.2 · termux/termux-app". Retrieved 29 March 2025. ^ Seth Kenlon (August 11, 2020). "Use a Linux terminal on your Android phone". opensource.com. Retrieved August 17, 2021. ^ "ZSH - Termux Wiki". Termux. July 30, 2020. Retrieved August 17, 2021. ^ "The Termux Wiki". Termux. July 30, 2021. Retrieved August 17, 2021. ^ "Graphical Environment - Termux Wiki". Termux. July 30, 2021. Retrieved August 17, 2021. ^ termux/proot-distro, Termux, 2024-11-03, retrieved 2024-11-03 ^ "Behavior changes: apps targeting API 29+". Android Developers. Retrieved 2021-12-01. ^ "Into the Sunset: Bintray, JCenter, GoCenter, and ChartCenter". JFrog. 2021-02-03. Retrieved 2021-12-01. ^ "Termux". Termux. Retrieved 1 June 2023. ^ "Differences from Linux - Termux Wiki". termux.com. 30 July 2021. Retrieved 18 August 2021. Media related to Termux at Wikimedia Commons Official website Termux Android package at the F-Droid repository Termux on GitHub Termux packages on GitHub Termux wiki Download Termux APK Latest Version Retrieved from " Termux is an Android terminal application and Linux environment. Note that this repository is for the app itself (the user interface and the terminal emulation). For the packages installable inside the app, see termux/termux-packages. Quick how-to about Termux package management is available at Package Management. It also has info on how to fix repository is under maintenance or down errors when running apt or pkg commands. We are looking for Termux Android application maintainers. NOTICE: Termux may be unstable on Android 12+. Android OS will kill any (phantom) processes greater than 32 (limit is for all apps combined) and also kill any processes using excessive CPU. You may get [Process completed (signal 9) - press Enter] message in the terminal without actually exiting the shell process yourself. Check the related issue #2366, issue tracker, phantom cached and empty processes docs and this TLDR comment on how to disable trimming of phantom and excessive cpu usage processes. A proper docs page will be added later. An option to disable the killing should be available in Android 12L or 13, so upgrade at your own risk if you are on Android 11, specially if you are not rooted. The core Termux app comes with the following optional plugin apps. Latest version is v0.118.3. NOTICE: It is highly recommended that you update to v0.118.0 or higher ASAP for various bug fixes, including a critical world-readable vulnerability reported here. See below for information regarding Termux on Google Play. Termux can be obtained through various sources listed below for only Android >= 7 with full support for apps and packages. Support for both app and packages was dropped for Android 5 and 6 on 2020-01-01 at v0.83, however it was re-added just for the app without any support for package updates on 2022-05-24 via the GitHub sources. Check here for the details. The APK files of different sources are signed with different signature keys. The Termux app and all its plugins use the same sharedUserId com.termux and so all their APKs installed on a device must have been signed with the same signature key to work together and so they must all be installed from the same source. Do not attempt to mix them together, i.e do not try to install an app or plugin from F-Droid and another one from a different source like GitHub. Android Package Manager will also normally not allow installation of APKs with different signatures and you will get errors on installation like App not installed, Failed to install due to an unknown error, INSTALL_FAILED_UPDATE_INCOMPATIBLE, INSTALL_FAILED_SHARED_USER_INCOMPATIBLE, signatures do not match previously installed version, etc. This restriction can be bypassed with root or with custom roms. If you wish to install from a different source, then you must uninstall any and all existing Termux or its plugin app APKs from your device first, then install all new APKs from the same new source. Check Uninstallation section for details. You may also want to consider Backing up Termux before the uninstallation so that you can restore it after re-installing from Termux different source. In the following paragraphs, "bootstrap" refers to the minimal packages that are shipped with the termux-app itself to start a working shell environment. Its zips are built and released here. Termux application can be obtained from F-Droid from here. You do not need to download the F-Droid app (via the Download F-Droid link) to install Termux. You can download the Termux APK directly from the site by clicking the Download APK link at the bottom of each version section. It usually takes a few days (or even a week or more) for updates to be available on F-Droid once an update has been released on GitHub. The F-Droid releases are built and published by F-Droid once they detect a new GitHub release. The Termux maintainers do not have any control over the building and publishing of the Termux apps on F-Droid. Moreover, the Termux maintainers also do not have access to the APK signing keys of F-Droid releases, so we cannot release an APK ourselves on GitHub that would be compatible with F-Droid releases. The F-Droid app often may not notify you of updates and you will manually have to do a pull down swipe action in the Updates tab of the app for it to check updates. Make sure battery optimizations are disabled for the app, check for details on how to do that. Only a universal APK is released, which will work on all supported architectures. The APK and bootstrap installation size will be ~180MB. F-Droid does not support architecture specific APKs. Termux application can be obtained on GitHub either from GitHub Releases for version >= 0.118.0 or from GitHub Build Action workflows. For android >= 7, only install apt-android-7 variants. For android 5 and 6, only install apt-android-5 variants. The APKs for GitHub Releases will be listed under Assets drop-down of a release. These are automatically attached when a new version is released. The APKs for GitHub Build action workflows will be listed under Artifacts section of a workflow run. These are created for each commit/push done to the repository and can be used by users who don't want to wait for releases and want to try out the latest features immediately or want to test their pull requests. Note that for action workflows, you need to be logged into a GitHub account for the Artifacts links to be enabled/clickable. If you are using the GitHub app, then make sure to open workflow link in a browser like Chrome or Firefox that has your GitHub account logged in since the in-app browser may not be logged in. The APKs for both of these are debuggable and are compatible with each other but they are not compatible with other sources. Both universal and architecture specific APKs are released. The APK and bootstrap installation size will be ~180MB if using universal and ~120MB if using architecture specific. Check here for details. Security warning: APK files on GitHub are signed with a test key that has been shared with community. This IS NOT an official developer key and everyone can use it to generate releases for own testing. Be very careful when using Termux GitHub builds obtained elsewhere except . Everyone is able to use it to forge a malicious Termux update installable over the GitHub build. Think twice about installing Termux builds distributed via Telegram or other social media. If your device get caught by malware, we will not be able to help you. The test key shall not be used to impersonate @termux and can't be used for this anyway. This key is not trusted by us and it is quite easy to detect its use in user generated content. Keystore information Alias name: alias Creation date: Oct 4, 2019 Entry type: PrivateKeyEntry Certificate chain length: 1 Certificate[1]: Owner: CN=APK Signer, OU=Earth, O=Earth Issuer: CN=APK Signer, OU=Earth, O=Earth Serial number: 29be297b Valid from: Wed Sep 04 02:03:24 EEST 2019 until: Tue Oct 26 02:03:24 EEST 2049 Certificate fingerprints: SHA1: 51:79:55:EA:BF:69:FC:05:7C:41:C7:D3:79:DB:BC:EF:20:AD:85:F2 SHA256: B6:DA:01:48:0E:EF:D5:FB:F2:CD:37:71:B8:D1:02:1E:C7:91:30:4B:DD:6C:4B:F4:1D:3F:AA:BA:D4:8E:E5:E1 Signature algorithm name: SHA1withRSA (disabled) Subject Public Key Algorithm: 2048-bit RSA key Version: 3 There is currently a build of Termux available on Google Play for Android 11+ devices, with extensive adjustments in order to pass policy requirements there. This is under development and has missing functionality and bugs (see here for status updates) compared to the stable F-Droid build, which is why most users who can should still use F-Droid or GitHub build as mentioned above. Currently, Google Play will try to update installations away from F-Droid ones. Updating will still fail as sharedUserId has been removed. A planned 0.118.1 F-Droid release will fix this by setting a higher version code than used for the PlayStore app. Meanwhile, to prevent Google Play from attempting to download and then fail to install the Google Play releases over existing installations, you can open the Termux apps pages on Google Play and then click on the 3 dots options button in the top right and then disable the Enable auto update toggle. However, the Termux apps updates will still show in the PlayStore app updates list. If you want to help out with testing the Google Play build (or cannot install Termux from other sources), be aware that it's built from a separate repository (- be sure to report issues there, as any issues encountered might very well be specific to that repository. Uninstallation may be required if a user doesn't want Termux installed in their device anymore or is switching to a different install source. You may also want to consider Backing up Termux before the uninstallation. To uninstall Termux completely, you must uninstall any and all existing Termux or its plugin app APKs listed in Termux App and Plugins. Go to Android Settings -> Applications and then look for those apps. You can also use the search feature if it's available on your device and search termux in the applications list. Even if you think you have not installed any of the plugins, it's strongly suggested to go through the application list in Android settings and double-check. All community links are available here. The main ones are the following. Termux Wiki Termux App Wiki Termux Packages Wiki You can help debug problems of the Termux app and its plugins by setting appropriate logcat Log Level in Termux app settings -> -> Debugging -> Log Level (Requires Termux app version >= 0.118.0). The Log Level defaults to Normal and log level Verbose currently logs additional information. Its best to revert log level to Normal after you have finished debugging since private data may otherwise be passed to logcat during normal operation and moreover, additional logging increases execution time. The plugin apps do not execute the commands themselves but send execution intents to Termux app, which has its own log level which can be set in Termux app settings -> Termux -> Debugging -> Log Level. So you must set log level for both Termux and the respective plugin app settings to get all the info. Once log levels have been set, you can run the logcat command in Termux app terminal to view the logs in realtime (Ctrl+c to stop) or use logcat -d > logcat.txt to take a dump of the log. You can also view the logs from a PC over ADB. For more information, check official android logcat guide here. Moreover, users can generate termux files stat info and logcat dump automatically too with terminal's long hold options menu More -> Report Issue option and selecting YES in the prompt shown to add debug info. This can be helpful for reporting and debugging other issues. If the report generated is too large, then Save To File option in context menu (3 dots on top right) of ReportActivity can be used and the file viewed/shared instead. Users must post complete report (optionally without sensitive info) when reporting issues. Issues opened with (partial) screenshots of error reports instead of text will likely be automatically closed/deleted. Off - Log nothing. Normal - Start logging error, warn and info messages and stacktraces. Debug - Start logging debug messages. Verbose - Start logging verbose messages. The termux-shared library was added in v0.109. It defines shared constants and utils of the Termux app and its plugins. It was created to allow for the removal of all hardcoded paths in the Termux app. Some of the termux plugins are using this as well and rest will in future. If you are contributing code that is using a constant or a util that may be shared, then define it in termux-shared library if it currently doesn't exist and reference it from there. Update the relevant changelogs as well. Pull requests using hardcoded values will/should not be accepted. Termux app and plugin specific classes must be added under com.termux.shared.termux package and general classes outside it. The termux-shared LICENSE must also be checked and updated if necessary when contributing code. The licenses of any external library or code must be honoured. The main Termux constants are defined by TermuxConstants class. It also contains information on how to fork Termux or build it with your own package name. Changing the package name will require building the bootstrap zip packages and other packages with the new $PREFIX, check Building Packages for more info. Check Termux Libraries for how to import termux libraries in plugin apps and Forking and Local Development for how to update termux libraries for plugins. The versionName in build.gradle files of Termux and its plugin apps must follow the semantic version 2.0.0 spec in the format major.minor.patch(-prerelease)(+buildmetadata). When bumping versionName in build.gradle files and when creating a tag for new releases on GitHub, make sure to include the patch number as well, like v0.1.0 instead of just v0.1. The build.gradle files and attach_debug_apks_to_release workflow validates the version as well and the build/attachment will fail if versionName does not follow the spec. Commit messages must use the Conventional Commits spec so that chagelogs as per the Keep a Changelog spec can automatically be generated by the create-conventional-changelog script, check its repo for further details on the spec. The first letter for type and description must be capital and description should be in the present tense. The space after the colon : is necessary. For a breaking change, add an exclamation mark ! before the colon :, so that it is highlighted in the chagelog automatically. [optional scope]: [optional body] [optional footer(s)] Only the types listed below must be used exactly as they are used in the changelog headings. For example, Added: Add foo, Added|Fixed: Add foo and fix bar, Changed!: Change baz as a breaking change, etc. You can optionally add a scope as well, like Fixed(terminal): Fix some bug. Do not use anything else as type, like add instead of Added, etc. Added for new features. Changed for changes in existing functionality. Deprecated for soon-to-be removed features. Removed for now removed features. Fixed for any bug fixes. Security in case of vulnerabilities. Check TermuxConstants javadocs for instructions on what changes to make in the app to change package name. You also need to recompile bootstrap zip for the new package name. Check building bootstrap, here and here. Currently, not all plugins use TermuxConstants from termux-shared library and have hardcoded com.termux values and will need to be manually patched. If forking termux plugins, check Forking and Local Development for info on how to use termux libraries for plugins. You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session. You switched accounts on another tab or window. Reload to refresh your session. Dismiss alert Instantly share code, notes, and snippets. Clone this repository at <script src=" quot;></script> Save gwenhael-le-moine/57409ffed9066ce7f4d4 to your computer and use it in GitHub Desktop. Clone this repository at <script src=" quot;></script> Save gwenhael-le-moine/57409ffed9066ce7f4d4 to your computer and use it in GitHub Desktop. run termux programs as root.. You can't perform that action at this time. 100%(1)100% found this document useful (1 vote)5K viewsThis document provides instructions for becoming a root user in Termux by installing the tsu and ncurses-utils packages and copying the sudo file to the usr/bin directory. It explains that s…AI-enhanced title and descriptionSaveSave ROOT EN TERMUX For Later100%100% found this document useful, undefined As a Linux user I use the sudo command very much to access the root file system of my desktop.But in Android when I started using Termux I found out that it has a beautiful Linux interface .But there was some problems in accessing some commands and changing owners or groups through the termux so I have made a simple script to make a sudo command.This command needs the su binary to be installed in your device for the command script to work.I also have made a permission fix script for termux. Installation: Extract the given zip. Then copy the setperm script and sudo to the following location /data/data/com.termux/files/usr/bin Set permission 755 rwxr-xr-x To both the files. Changelog : Full Magisk Support. Now no problems using and other type of rooting methods. Debugged error in script. Any bugs found could be reported with the last log of the session by taking a screenshot of the session and attaching the png file. Otherwise the bugs will be hard to fix. Thank you for your cooperation. the script was written by @st42 thanks to st42. Last edited: Aug 18, 2018 Reactions: Tracyorama, BeyondIQ, JohnBobDoe3rd and 9 others How to install su binary? Can you provide some detail as I am a newbie? Can you fix this? I am rooted with latest magisk v16.0.plz help sudo su CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22132 Aborted $SU -c "$MOUNT_RW" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22136 Aborted $SU -c "mkdir $ROOT_HOME" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22140 Aborted $SU -c "chmod 700 $ROOT_HOME" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22147 Aborted $SU -c "echo -e $BASHRC > $ROOT_HOME/.bashrc" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22152 Aborted $SU -c "chmod 700 $ROOT_HOME/.bashrc" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22156 Aborted $SU -c "$MOUNT_RO" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 87: 22169 Aborted $SU -c "$CMDLINE" Can you fix this? I am rooted with latest magisk v16.0.plz help sudo su CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22132 Aborted $SU -c "$MOUNT_RW" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22136 Aborted $SU -c "mkdir $ROOT_HOME" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22140 Aborted $SU -c "chmod 700 $ROOT_HOME" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22147 Aborted $SU -c "echo -e $BASHRC > $ROOT_HOME/.bashrc" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22152 Aborted $SU -c "chmod 700 $ROOT_HOME/.bashrc" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 65: 22156 Aborted $SU -c "$MOUNT_RO" CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... CANNOT LINK EXECUTABLE "/sbin/su": cannot locate symbol "XzUnpacker_Construct" referenced by "/system/lib64/libunwind.so"... /data/data/com.termux/files/usr/bin/sudo: line 87: 22169 Aborted $SU -c "$CMDLINE" Sorry support for Magisk is disabled.Support for Magisk will be brought in future. Anyone know how to install fdisk ? I am not using termux anymore but I think in the sudo file I changed these values to SU=su and it worked for magisk if [-x $SYSXBIN/su]; then SU=$SYSXBIN/su elif [-x /su/bin/su]; then SU=/su/bin/su elif [-x /sbin/su]; then SU=/sbin/su Sorry but I am a noob and don't know much, I can't understand what to do with the sudo file present in. ../termux-root Sorry but I am a noob and don't know much, I can't understand what to do with the sudo file present in. ../termux-root Just follow this procedure in the first post zip is given Installation: Extract the given zip. Then copy the setperm script and sudo to the following location /data/data/com.termux/files/usr/bin Set permission 755 rwxr-xr-x Then just open sudo file and change the values I mentioned it should work Sent from my [device_name] using XDA-Developers Legacy app Reactions: Anas Rahman Just follow this procedure in the first post zip is given Installation: Extract the given zip. Then copy the setperm script and sudo to the following location /data/data/com.termux/files/usr/bin Set permission 755 rwxr-xr-x Then just open sudo file and change the values I mentioned it should work Sent from my [device_name] using XDA-Developers Legacy app Magisk Support is back. here is no problem in magisk su. Fully supporting magisk. Reactions: munny_sun Magisk Support is back. here is no problem in magisk su. Fully supporting magisk. Hey I need ur help it was working fine in custom rom but now in stock ROM its showing this error:- $ sudo bash: /data/data/com.termux/files/usr/bin/sudo: Permission denied $ setperm.sh bash: /data/data/com.termux/files/usr/bin/setperm.sh: Permission denied Even though both files permission are set to 755 (rwx r-x r-x) Sent from my [OnePlus 3] using XDA-Developers Legacy app Hey I need ur help it was working fine in custom rom but now in stock ROM its showing this error:- $ sudo bash: /data/data/com.termux/files/usr/bin/sudo: Permission denied $ setperm.sh bash: /data/data/com.termux/files/usr/bin/setperm.sh: Permission denied Even though both files permission are set to 755 (rwx r-x r-x) I find that that I need to run restorecon on files that I've copied from my internal storage to /data otherwise I get permission problems. Sent from my OnePlus3T using XDA Labs Thanks a lot for sharing. I think you quite mistyped in sudo's script, cos on Android permission 700 is not possible to apply I think. So all occurences must be changed to 755 am I right?. Abstract: Learn how to install the su binary using Termux in this comprehensive step-by-step guide. Gain root access on your Android device and unlock advanced functionalities easily. 2023-09-14 by UserComp.com Editors Termux is a powerful terminal emulator app for Android devices that allows you to access and run Linux commands on your phone. One useful feature of Termux is the ability to install the su binary, which gives you root access on your device. In this step-by-step guide, we will walk you through the process of installing the su binary using Termux. Step 1: Install Termux If you haven't already, start by installing the Termux app from the Google Play Store. Simply search for "Termux" and tap on the install button to download and install the app on your device. Step 2: Launch Termux Once Termux is installed, open the app to launch the terminal emulator. You will be greeted with a command prompt where you can start running commands. Step 3: Update Termux Before proceeding with the installation of the su binary, it is recommended to update Termux to ensure you have the latest version. To update Termux, run the following command: apt update && apt upgrade This command will update the package lists and upgrade any outdated packages on your device. Step 4: Install the Required Packages To install the su binary, we need to install a few additional packages. Run the following command to install them: pkg install git tsu This command will install the Git version control system and the tsu utility, which is a wrapper for su that allows us to run commands as root. Step 5: Clone the su Repository Next, we need to clone the su repository from GitHub. Run the following command to clone the repository: git clone This command will download the su repository to your device. Step 6: Navigate to the su Directory Once the repository is cloned, navigate to the su directory using the following command: cd su This command will change your current directory to the su directory. Step 7: Build and Install su Now, we are ready to build and install the su binary. Run the following command to start the installation process: make This command will compile the su binary from the source code and install it on your device. Step 8: Grant Superuser Access After the installation is complete, we need to grant superuser access to the su binary. Run the following command: tsu -c ./su --install This command will prompt you to grant superuser access to the su binary. Follow the on-screen instructions to complete the process. Step 9: Verify the Installation To verify that the su binary is installed correctly, run the following command: su --version If the installation was successful, you should see the version number of the su binary displayed in the terminal. Congratulations! You have successfully installed the su binary using Termux. You now have root access on your Android device. Conclusion Installing the su binary using Termux allows you to gain root access on your Android device, giving you more control and flexibility. However, it is important to use this power responsibly and only perform actions that are necessary and within your knowledge. References Explore our wide range of tech support articles to enhance your knowledge and troubleshoot various issues efficiently. Find solutions to your tech-related queries and stay updated with the latest trends in the tech world. Termux is an Android terminal emulator and Linux environment app that works directly with no rooting or setup required. A minimal base system is installed automatically - additional packages are available using the APT package manager. Secure. Access remote servers using the ssh client from OpenSSH. Termux combines standard packages with accurate terminal emulation in a beautiful open source solution. Feature packed. Take your pick between Bash, fish or Zsh and nano, Emacs or Vim. Grep through your SMS inbox. Access API endpoints with curl and use rsync to store backups of your contact list on a remote server. Customizable. Install what you want through the APT package management system known from Debian and Ubuntu GNU/Linux. Why not start with installing Git and syncing your dotfiles? Explorable. Have you ever sat on a bus and wondered exactly which arguments tar accepts? Packages available in Termux are the same as those on Mac and Linux - install man pages on your phone and read them in one session while experimenting with them in another. Portable - With batteries included. Can you imagine a more powerful yet elegant pocket calculator than a readline-powered Python console? Up-to-date versions of Perl, Python, Ruby and Node.js are all available. Ready to scale up. Connect a Bluetooth keyboard and hook up your device to an external display if you need to - Termux supports keyboard shortcuts and has full mouse support. Tinkerable. Develop by compiling Go, Rust, Swift, or C files with Clang and build your own projects with CMake and pkg-config. Both lldb/GDB and strace are available if you get stuck and need to debug. The default su from Termux is simple. When run it brings /system/bin/sh for you instead of Termux's shell (typically bash). To solve this issue, you can install the custom su wrapper for Termux by apt install tsu And then you can run tsu so that your bash and everything is now under root privilege. Further information: Use an editor (like vim) to see /data/data/com.termux/files/usr/bin/su and you'll find that at line 10, it calls exec su $@ directly while in tsu, it performs shell detection and at line 80, it runs exec su --preserve-environment -c "LD_LIBRARY_PATH=$PREFIX/lib $ROOT_SHELL" to preserve the Termux enviroment. Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. The default su from Termux is simple. When run it brings /system/bin/sh for you instead of Termux's shell (typically bash). To solve this issue, you can install the custom su wrapper for Termux by apt install tsu And then you can run tsu so that your bash and everything is now under root privilege. Further information: Use an editor (like vim) to see /data/data/com.termux/files/usr/bin/su and you'll find that at line 10, it calls exec su $@ directly while in tsu, it performs shell detection and at line 80, it runs exec su --preserve-environment -c "LD_LIBRARY_PATH=$PREFIX/lib $ROOT_SHELL" to preserve the Termux enviroment. install Root in the Termux have you ever wanted to run scripts but they need root access or for some reason you want to enter your commands in the real linux environment? in this tutorial, it is executed only by running a Termux script in root mode. # install git pkg install git -y # clone from github with git git clone && cd root-termux && chmod +x * # install packages pkg install wget proot -y then enter the following command and go through the installation process: # run bash script `install.sh` yes | bash install.sh after installing and creating the executable file, run the root environment with the following command: # run bash script `start.sh` bash start.sh # or `./start.sh` Note: after running Termux as root, you need to install all the packages from the beginning, and be sure to update the terminal with the following commands after installation so that you don't have any problems installing the packages: command to install sudo after installing Ubuntu: You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session. You switched accounts on another tab or window. Reload to refresh your session. Dismiss alert Instantly share code, notes, and snippets. Clone this repository at <script src=" quot;></script> Save GabrielMMelo/0e146f32d73978bf0d0a06786bcbc96c to your computer and use it in GitHub Desktop. Clone this repository at <script src=" quot;></script> Save GabrielMMelo/0e146f32d73978bf0d0a06786bcbc96c to your computer and use it in GitHub Desktop. Install sudo in Termux (Android) You can't perform that action at this time. rust-script rust-script Public Run Rust files and expressions as scripts without any setup or compilation step. Rust 1.4k 47 ELF parsing library in java. Java 160 48 apksigner apksigner Public A lightweight APK signing tool that can be run on Android devices. Java 121 31 rust-static-builder rust-static-builder Public Docker image to build statically linked Linux executables from Rust projects. Dockerfile 49 8 eclipsescript eclipsescript Public archive Eclipse plugin for scripting using javascript Java 29 5 Reddit and its partners use cookies and similar technologies to provide you with a better experience. By accepting all cookies, you agree to our use of cookies to deliver and maintain our services and site, improve the quality of Reddit, personalize Reddit content and advertising, and measure the effectiveness of advertising. By rejecting non-essential cookies, Reddit may still use certain cookies to ensure the proper functionality of our platform. For more information, please see our Cookie Notice and our Privacy Policy. You can't perform that action at this time. Family of computer architectures "ARM architecture" redirects here. For the Australian architectural firm, see ARM Architecture (company). ARMDesigner Sophie Wilson Steve Furber Acorn Computers/Arm Holdings Bits32-bit, 64-bitIntroduced1985; 40 years ago (1985)DesignRISCTypeLoad–storeBranchingCondition code, compare and branchOpenProprietary ARM AArch64 (64/32-bit)Introduced2011; 14 years ago (2011)VersionARMv8-R, ARMv8-A, ARMv8.1-A, ARMv8.2-A, ARMv8.3-A, ARMv8.4-A, ARMv8.5-A, ARMv8.6-A, ARMv8.7-A, ARMv8.8-A, ARMv8.9-A, ARMv9.0-A, ARMv9.1-A, ARMv9.2-A, ARMv9.3-A, ARMv9.4-A, ARMv9.5-A, ARMv9.6-AEncodingAArch64/A64 and AArch32/A32 use 32-bit instructions, AArch32/T32 (Thumb-2) uses mixed 16- and 32-bit instructions[1]EndiannessBi (little as default)ExtensionsSVE, SVE2, SME, AES, SM3, SM4, SHA, CRC32, RNDR, TME; All mandatory: Thumb-2, Neon, VFPv4-D16, VFPv4; obsolete: Thumb and JazelleRegistersGeneral-purpose31 × 64-bit integer registers[1]Floating point32 × 128-bit registers[1] for scalar 32- and 64-bit FP or SIMD FP or integer; or cryptography ARM AArch32 (32-bit)VersionARMv9-R, ARMv9-M, ARMv8-R, ARMv8-M, ARMv7-A, ARMv7-R, ARMv7E-M, ARMv7-MEncoding32-bit, except Thumb-2 extensions use mixed 16- and 32-bit instructions.EndiannessBi (little as default)ExtensionsThumb, Thumb-2, Neon, Jazelle, AES, SM3, SM4, SHA, CRC32, RNDR, DSP, Saturated, FPv4-SP, FPv5, Helium; obsolete since ARMv8: Thumb and JazelleRegistersGeneral-purpose15 × 32-bit integer registers, including R14 (link register), but not R15 (PC)Floating pointUp to 32 × 64-bit registers,[2] SIMD/floating-point (optional) ARM 32-bit (legacy)VersionARMv6, ARMv5, ARMv4T, ARMv3, ARMv2Encoding32-bit, except Thumb extension uses mixed 16- and 32-bit instructions.EndiannessBi (little as default) in ARMv3 and aboveExtensionsThumb, JazelleRegistersGeneral-purpose15 × 32-bit integer registers, including R14 (link register), but not R15 (PC, 26-bit addressing in older)Floating pointNone ARM (stylised in lowercase as arm, formerly an acronym for Advanced RISC Machines and originally Acorn RISC Machine) is a family of RISC instruction set architectures (ISAs) for computer processors. Arm Holdings develops the ISAs and licenses them to other companies, who build the physical devices that use the instruction set. It also designs and licenses cores that implement these ISAs. Due to their low costs, low power consumption, and low heat generation, ARM processors are useful for light, portable, battery-powered devices, including smartphones, laptops, and tablet computers, as well as embedded systems.[3][4][5] However, ARM processors are also used for desktops and servers, including Fugaku, the world's fastest supercomputer from 2020[6] to 2022. With over 230 billion ARM chips produced,[7][8] since at least 2003, and with its dominance increasing every year[update], ARM is the most widely used family of instruction set architectures.[9][4][10][11][12] There have been several generations of the ARM design. The original ARM1 used a 32-bit internal structure but had a 26-bit address space that limited it to 64 MB of main memory. This limitation was removed in the ARMv3 series, which has a 32-bit address space, and several additional generations up to ARMv7 remained 32-bit. Released in 2011, the ARMv8-A architecture added support for a 64-bit address space and 64-bit arithmetic with its new 32-bit fixed-length instruction set.[13] Arm Holdings has also released a series of additional instruction sets for different roles: the "Thumb" extensions add both 32- and 16-bit instructions for improved code density, while Jazelle added instructions for directly handling Java bytecode. More recent changes include the addition of simultaneous multithreading (SMT) for improved performance or fault tolerance.[14] Main article: BBC Micro Acorn Computers' first widely successful design was the BBC Micro, introduced in December 1981. This was a relatively conventional machine based on the MOS Technology 6502 CPU but ran at roughly double the performance of competing designs like the Apple II due to its use of faster dynamic random-access memory (DRAM). Typical DRAM of the era ran at about 2 MHz; Acorn arranged a deal with Hitachi for a supply of faster 4 MHz parts.[15] Machines of the era generally shared memory between the processor and the framebuffer, which allowed the processor to quickly update the contents of the screen without having to perform separate input/output (I/O). As the timing of the video display is exacting, the video hardware had to have priority access to that memory. Due to a quirk of the 6502's design, the CPU left the memory untouched for half of the time. Thus by running the CPU at 1 MHz, the video system could read data during those down times, taking up the total 2 MHz bandwidth of the RAM. In the BBC Micro, the use of 4 MHz RAM allowed the same technique to be used, but running at twice the speed. This allowed it to outperform any similar machine on the market.[16] Main article: Acorn Business Computer 1981 was also the year that the IBM Personal Computer was introduced. Using the recently introduced Intel 8088, a 16-bit CPU compared to the 6502's 8-bit design, it offered higher overall performance. Its introduction changed the desktop computer market radically: what had been largely a hobby and gaming market emerging over the prior five years began to change to a must-have business tool where the earlier 8-bit designs simply could not compete. Even newer 32-bit designs were also coming to market, such as the Motorola 68000[17] and National Semiconductor NS32016.[18] Acorn began considering how to compete in this market and produced a new paper design named the Acorn Business Computer. They set themselves the goal of producing a machine with ten times the performance of the BBC Micro, but at the same price.[19] This would outperform and underprice the PC. At the same time, the recent introduction of the Apple Lisa brought the graphical user interface (GUI) concept to a wider audience and suggested the future belonged to machines with a GUI.[20] The Lisa, however, cost $9,995, as it was packed with support chips, large amounts of memory, and a hard disk drive, all very expensive then.[21] The engineers then began studying all of the CPU designs available. Their conclusion about the existing 16-bit designs was that they were a lot more expensive and were still "a bit crap",[22] offering only slightly higher performance than their BBC Micro design. They also almost always demanded a large number of support chips to operate even at that level, which drove up the cost of the computer as a whole. These systems would simply not hit the design goal.[22] They also considered the new 32-bit designs, but these cost even more and had the same issues with support chips.[23] According to Sophie Wilson, all the processors tested at that time performed about the same, with about a 4 Mbit/s bandwidth.[24][a] Two key events led Acorn down the path to ARM. One was the publication of a series of reports from the University of California, Berkeley, which suggested that a simple chip design could nevertheless have extremely high performance, much higher than the latest 32-bit designs on the market.[25] The second was a visit by Steve Furber and Sophie Wilson to the Western Design Center, a company run by Bill Mensch and his sister, which had become the logical successor to the MOS team and was offering new versions like the WDC 65C02. The Acorn team saw high school students producing chip layouts on Apple II machines, which suggested that anyone could do it.[26][27] In contrast, a visit to another design firm working on modern 32-bit CPU revealed a team with over a dozen members who were already on revision H of their design and yet it still contained bugs.[b] This cemented their late 1983 decision to begin their own CPU design, the Acorn RISC Machine.[28] The original Berkeley RISC designs were in some sense teaching systems, not designed specifically for outright performance. To the RISC's basic register-heavy and load/store concepts, ARM added a number of the well-received design notes of the 6502. Primary among them was the ability to quickly serve interrupts, which allowed the machines to offer reasonable input/output performance with no added external hardware. To offer interrupts with similar performance as the 6502, the ARM design limited its physical address space to 64 MB of total addressable space, requiring 26 bits of address. As instructions were 4 bytes (32 bits) long, and required to be aligned on 4-byte boundaries, the lower 2 bits of an instruction address were always zero. This meant the program counter (PC) only needed to be 24 bits, allowing it to be stored along with the eight bit processor flags in a single 32-bit register. That meant that upon receiving an interrupt, the entire machine state could be saved in a single operation, whereas had the PC been a full 32-bit value, it would require separate operations to store the PC and the status flags. This decision halved the interrupt overhead.[29] Another change, and among the most important in terms of practical real-world performance, was the modification of the instruction set to take advantage of page mode DRAM. Recently introduced, page mode allowed subsequent accesses of memory to run twice as fast if they were roughly in the same location, or "page", in the DRAM chip. Berkeley's design did not consider page mode and treated all memory equally. The ARM design added special vector-like memory access instructions, the "S-cycles", that could be used to fill or save multiple registers in a single page using page mode. This doubled memory performance when they could be used, and was especially important for graphics performance.[30] The Berkeley RISC designs used register windows to reduce the number of register saves and restores performed in procedure calls; the ARM design did not adopt this. Wilson developed the instruction set, writing a simulation of the processor in BBC BASIC that ran on a BBC Micro with a second 6502 processor.[31][32] This convinced Acorn engineers they were on the right track. Wilson approached Acorn's CEO, Hermann Hauser, and requested more resources. Hauser gave his approval and assembled a small team to design the actual processor based on Wilson's ISA.[33] The official Acorn RISC Machine project started in October 1983. ARM1 2nd processor for the BBC Micro Acorn chose VLSI Technology as the "silicon partner", as they were a source of ROMs and custom chips for Acorn. Acorn provided the design and VLSI provided the layout and production. The first samples of ARM silicon worked properly when first received and tested on 26 April 1985.[3] Known as ARM1, these versions ran at 6 MHz.[34] The first ARM application was as a second processor for the BBC Micro, where it helped in developing simulation software to finish development of the support chips (VIDC, IOC, MEMC), and sped up the CAD software used in ARM2 development. Wilson subsequently rewrote BBC BASIC in ARM assembly language. The in-depth knowledge gained from designing the instruction set enabled the code to be very dense, making ARM BBC BASIC an extremely good test for any ARM emulator. The result of the simulations on the ARM1 boards led to the late 1986 introduction of the ARM2 design running at 8 MHz, and the early 1987 speed-bumped version at 10 to 12 MHz.[c] A significant change in the underlying architecture was the addition of a Booth multiplier, whereas formerly multiplication had to be carried out in software.[36] Further, a new Fast Interrupt reQuest mode, FIQ for short, allowed registers 8 through 14 to be replaced as part of the interrupt itself. This meant FIQ requests did not have to save out their registers, further speeding interrupts.[37] The first use of the ARM2 were in ARM Evaluations systems, supplied as a second processor for BBC Micro and Master machines, from July 1986,[38] internal Acorn A500 development machines,[39] and the Acorn Archimedes personal computer models A305, A310, and A440, launched on the 6th June 1987. According to the Dhrystone benchmark, the ARM2 was roughly seven times the performance of a typical 7 MHz 68000-based system like the Amiga or Macintosh SE. It was twice as fast as an Intel 80386 running at 16 MHz, and about the same speed as a multi-processor VAX-11/784 superminicomputer. The only systems that beat it were the Sun SPARC and MIPS R2000 RISC-based workstations.[40] Further, as the CPU was designed for high-speed I/O, it dispensed with many of the support chips seen in these machines; notably, it lacked any dedicated direct memory access (DMA) controller which was often found on workstations. The graphics system was also simplified based on the same set of underlying assumptions about memory and timing. The result was a dramatically simplified design, offering performance on par with expensive workstations but at a price point similar to contemporary desktops.[40] The ARM2 featured a 32-bit data bus, 26-bit address space and 27 32-bit registers, of which 16 are accessible at any one time (including the PC).[41] The ARM2 had a transistor count of just 30,000,[42] compared to Motorola's six-year-older 68000 model with around 68,000. Much of this simplicity came from the lack of microcode, which represents about one-quarter to one-third of the 68000's transistors, and the lack of (like most CPUs of the day) a cache. This simplicity enabled the ARM2 to have a low power consumption and simpler thermal packaging by having fewer powered transistors. Nevertheless, ARM2 offered better performance than the contemporary 1987 IBM PS/2 Model 50, which initially utilised an Intel 80286, offering 1.8 MIPS @ 10 MHz, and later in 1987, the 2 MIPS of the PS/2 70, with its Intel 386 DX @ 16 MHz.[43][44] A successor, ARM3, was produced with a 4 KB cache, which further improved performance.[45] The address bus was extended to 32 bits in the ARM6, but program code still had to lie within the first 64 MB of memory in 26-bit compatibility mode, due to the reserved bits for the status flags.[46] Microprocessor-based system on a chip Die of an ARM610 microprocessor In the late 1980s, Apple Computer and VLSI Technology started working with Acorn on newer versions of the ARM core. In 1990, Acorn spun off the design team into a new company named Advanced RISC Machines Ltd.,[47][48][49] which became ARM Ltd. when its parent company, Arm Holdings plc, floated on the London Stock Exchange and Nasdaq in 1998.[50] The new Apple–ARM work would eventually evolve into the ARM6, first released in early 1992. Apple used the ARM6-based ARM610 as the basis for their Apple Newton PDA. In 1994, Acorn used the ARM610 as the main central processing unit (CPU) in their RiscPC computers. DEC licensed the ARMv4 architecture and produced the StrongARM.[51] At 233 MHz, this CPU drew only one watt (newer versions draw far less). This work was later passed to Intel as part of a lawsuit settlement, and Intel took the opportunity to supplement their i960 line with the StrongARM. Intel later developed its own high performance implementation named XScale, which it has since sold to Marvell. Transistor count of the ARM core remained essentially the same throughout these changes; ARM2 had 30,000 transistors,[52] while ARM6 grew only to 35,000.[53] In 2005, about 98% of all mobile phones sold used at least one ARM processor.[54] In 2010, producers of chips based on ARM architectures reported shipments of 6.1 billion ARM-based processors, representing 95% of smartphones, 35% of digital televisions and set-top boxes, and 10% of mobile computers. In 2011, the 32-bit ARM architecture was the most widely used architecture in mobile devices and the most popular 32-bit one in embedded systems.[55] In 2013, 10 billion were produced[56] and "ARM-based chips are found in nearly 60 percent of the world's mobile devices".[57] See also: Arm Holdings § Licensees Die of a STM32F103VGT6 ARM Cortex-M3 microcontroller with 1 MB flash memory by STMicroelectronics Arm Holdings's primary business is selling IP cores, which licensees use to create microcontrollers (MCUs), CPUs, and systems-on-chips based on those cores. The original design manufacturer combines the ARM core with other parts to produce a complete device, typically one that can be built in existing semiconductor fabrication plants (fabs) at low cost and still deliver substantial performance. The most successful implementation has been the ARM7TDMI with hundreds of millions sold. Atmel has been a precursor design center in the ARM7TDMI-based embedded system. The ARM architectures used in smartphones, PDAs and other mobile devices range from ARMv5 to ARMv8-A. In 2009, some manufacturers introduced netbooks based on ARM architecture CPUs, in direct competition with netbooks based on Intel Atom.[58] Arm Holdings offers a variety of licensing terms, varying in cost and deliverables. Arm Holdings provides to all licensees an integratable hardware description of the ARM core as well as complete software development toolset (compiler, debugger, software development kit), and the right to sell manufactured silicon containing the ARM CPU. SoC packages integrating ARM's core designs include Nvidia Tegra's first three generations, CSR plc's Quatro family, ST-Ericsson's Nova and NovaThor, Silicon Labs's Precision32 MCU, Texas Instruments's OMAP products, Samsung's Hummingbird and Exynos products, Apple's A4, A5, and A5X, and NXP's i.MX. Fabless licensees, who wish to integrate an ARM core into their own chip design, are usually only interested in acquiring a ready-to-manufacture verified semiconductor intellectual property core. For these customers, Arm Holdings delivers a gate netlist description of the chosen ARM core, along with an abstracted simulation model and test programs to aid design integration and verification. More ambitious customers, including integrated device manufacturers (IDM) and foundry operators, choose to acquire the processor IP in synthesizable RTL (Verilog) form. With the synthesizable RTL, the customer has the ability to perform architectural level optimisations and extensions. This allows the designer to achieve exotic design goals not otherwise possible with an unmodified netlist (high clock speed, very low power consumption, instruction set extensions, etc.). While Arm Holdings does not grant the licensee the right to resell the ARM architecture itself, licensees may freely sell manufactured products such as chip devices, evaluation boards and complete systems. Merchant foundries can be a special case; not only are they allowed to sell finished silicon containing ARM cores, they generally hold the right to re-manufacture ARM cores for other customers. Arm Holdings prices its IP based on perceived value. Lower performing ARM cores typically have lower licence costs than higher performing cores. In implementation terms, a synthesisable core costs more than a hard macro (blackbox) core. Complicating price matters, a merchant foundry that holds an ARM licence, such as Samsung or Fujitsu, can offer fab customers reduced licensing costs. In exchange for acquiring the ARM core through the foundry's in-house design services, the customer can reduce or eliminate payment of ARM's upfront licence fee. Compared to dedicated semiconductor foundries (such as TSMC and UMC) without in-house design services, Fujitsu/Samsung charge two- to three-times more per manufactured wafer.[citation needed] For low to mid volume applications, a design service foundry offers lower overall pricing (through subsidisation of the licence fee). For high volume mass-produced parts, the long term cost reduction achievable through lower wafer pricing reduces the impact of ARM's NRE (non-recurring engineering) costs, making the dedicated foundry a better choice. Companies that have developed chips with cores designed by Arm include Amazon.com's Annapurna Labs subsidiary,[59] Analog Devices, Apple, AppliedMicro (now: MACOM Technology Solutions[60]), Atmel, Broadcom, Cavium, Cypress Semiconductor, Freescale Semiconductor (now NXP Semiconductors), Huawei, Intel,[dubious – discuss] Maxim Integrated, Nvidia, NXP, Qualcomm, Renesas, Samsung Electronics, ST Microelectronics, Texas Instruments, and Xilinx. In February 2016, ARM announced the Built on ARM Cortex Technology licence, often shortened to Built on Cortex (BoC) licence. This licence allows companies to partner with ARM and make modifications to ARM Cortex designs. These design modifications will not be shared with other companies. These semi-custom core designs also have brand freedom, for example Kryo 280. Companies that are current licensees of Built on ARM Cortex Technology include Qualcomm.[61] Companies can also obtain an ARM architectural licence for designing their own CPU cores using the ARM instruction sets. These cores must comply fully with the ARM architecture. Companies that have designed cores that implement an ARM architecture include Apple, AppliedMicro (now: Ampere Computing), Broadcom, Cavium (now: Marvell), Digital Equipment Corporation, Intel, Nvidia, Qualcomm, Samsung Electronics, Fujitsu, and NUVIA Inc. (acquired by Qualcomm in 2021). On 16 July 2019, ARM announced ARM Flexible Access. ARM Flexible Access provides unlimited access to included ARM intellectual property (IP) for development. Per product licence fees are required once a customer reaches foundry tapeout or prototyping.[62][63] 75% of ARM's most recent IP over the last two years are included in ARM Flexible Access. As of October 2019: CPUs: Cortex-A5, Cortex-A7, Cortex-A32, Cortex-A34, Cortex-A35, Cortex-A53, Cortex-R5, Cortex-R8, Cortex-R52, Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4, Cortex-M7, Cortex-M23, Cortex-M33 GPUs: Mali-G52, Mali-G31. Includes Mali Driver Development Kits (DDK). Interconnect: CoreLink NIC-400, CoreLink NIC-450, CoreLink CCI-400, CoreLink CCI-500, CoreLink CCI-550, ADB-400 AMBA, XHB-400 AXI-AHB System Controllers: CoreLink GIC-400, CoreLink GIC-500, PL192 VIC, BP141 TrustZone Memory Wrapper, CoreLink TZC-400, CoreLink L2C-310, CoreLink MMU-500, BP140 Memory Interface Security IP: CryptoCell-312, CryptoCell-712, TrustZone True Random Number Generator Peripheral Controllers: PL011 UART, PL022 SPI, PL031 RTC Debug & Trace: CoreSight SoC-400, CoreSight SDC-600, CoreSight STM-500, CoreSight System Trace Macrocell, CoreSight Trace Memory Controller Design Kits: Corstone-101, Corstone-201 Physical IP: Artisan PIK for Cortex-M33 TSMC 22ULL including memory compilers, logic libraries, GPIOs and documentation Tools & Materials: Socrates IP ToolingARM Design Studio, Virtual System Models Support: Standard ARM Technical support, ARM online training, maintenance updates, credits toward onsite training and design reviews Main article: List of ARM processors Architecture Corebit-width Cores Profile Refe-rences Arm Ltd. Third-party ARMv1 32 ARM1 Classic [a 1] ARMv2 32 ARM2, ARM250, ARM3 Amber, STORM Open Soft Core[64] Classic [a 1] ARMv3 32 ARM6, ARM7 Classic [a 2] ARMv4 32 ARM8 StrongARM, FA526, ZAP Open Source Processor Core Classic [a 2] [65] ARMv4T 32 ARM7TDMI, ARM9TDMI, SecurCore SC100 Classic [a 2] ARMv5TE 32 ARM7EJ, ARM9E, ARM10E XScale, FA626TE, Feroceon, PJ1/Mohawk Classic ARMv6 32 ARM11 Classic ARMv6-M 32 ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000 Microcontroller ARMv7-M 32 ARM Cortex-M3, SecurCore SC300 Apple M7 motion coprocessor Microcontroller ARMv7E-M 32 ARM Cortex-M4, ARM Cortex-M7 Microcontroller ARMv8-M 32 ARM Cortex-M23,[66] ARM Cortex-M33[67] Microcontroller [68] ARMv8.1-M 32 ARM Cortex-M55, ARM Cortex-M85 Microcontroller [69] ARMv7-R 32 ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7, ARM Cortex-R8 Real-time ARMv8-R 32 ARM Cortex-R52 Real-time [70][71][72] 64 ARM Cortex-R82 Real-time ARMv7-A 32 ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17 Qualcomm Scorpion/Krait, PJ4/Sheeva, Apple Swift (A6, A6X) Application ARMv8-A 32 ARM Cortex-A32[73] Application 64/32 ARM Cortex-A35,[74] ARM Cortex-A53, ARM Cortex-A57,[75] ARM Cortex-A72,[76] ARM Cortex-A73[77] X-Gene, Nvidia Denver 1/2, Cavium ThunderX, AMD K12, Apple Cyclone (A7)/Typhoon (A8, A8X)/Twister (A9, A9X)/Hurricane+Zephyr (A10, A10X), Qualcomm Kryo, Samsung M1/M2 ("Mongoose") /M3 ("Meerkat") Application [78][1][79][80][81][82] 64 ARM Cortex-A34[83] Application ARMv8.1-A 64/32 Cavium ThunderX2 Application [84] ARMv8.2-A 64/32 ARM Cortex-A55,[85] ARM Cortex-A75,[86] ARM Cortex-A76,[87] ARM Cortex-A77, ARM Cortex-A78, ARM Cortex-X1, ARM Neoverse N1 Nvidia Carmel, Samsung M4 ("Cheetah"), Fujitsu A64FX (ARMv8 SVE 512-bit) Application [88][89][90] 64 ARM Cortex-A65, ARM Neoverse E1 with simultaneous multithreading (SMT), ARM Cortex-A65AE[91] (also having e.g. ARMv8.4 Dot Product; made for safety critical tasks such as advanced driver-assistance systems (ADAS)) Apple Monsoon+Mistral (A11) (September 2017) Application ARMv8.3-A 64/32 Application 64 Apple Vortex+Tempest (A12, A12X, A12Z), Marvell ThunderX3 (v8.3+)[92] Application ARMv8.4-A 64/32 Application 64 ARM Neoverse V1 Apple Lightning+Thunder (A13), Apple Firestorm+Icestorm (A14, M1) Application ARMv8.5-A 64/32 Application 64 Application ARMv8.6-A 64 Apple Avalanche+Blizzard (A15, M2), Apple Everest+Sawtooth (A16),[93] Apple Coll (A17), Apple Ibiza/Lobos/Palma (M3) Application ARMv8.7-A 64 Application [94] ARMv8.8-A 64 Application ARMv8.9-A 64 Application ARMv9.0-A 64 ARM Cortex-A510, ARM Cortex-A710, ARM Cortex-A715, ARM Cortex-X2, ARM Cortex-X3, ARM Neoverse E2, ARM Neoverse N2, ARM Neoverse V2 Application [95][96] ARMv9.1-A 64 Application ARMv9.2-A 64 ARM Cortex-A520, ARM Cortex-A720, ARM Cortex-X4, ARM Neoverse V3,[97] ARM Cortex-X925,[98] ARM Cortex-A320[99] Apple Donan/BravaChop/Brava (Apple M4),[100] Apple Tupai/Tahiti (A18) Application ARMv9.3-A 64 TBA Application [101] ARMv9.4-A 64 TBA Application [102] ARMv9.5-A 64 TBA Application [103] ARMv9.6-A 64 TBA Application [104] ^ a b Although most datapaths and CPU registers in the early ARM processors were 32-bit, addressable memory was limited to 26 bits; with upper bits, then, used for status flags in the program counter register. ^ a b c ARMv3 included a compatibility mode to support the 26-bit addresses of earlier versions of the architecture. This compatibility mode optional in ARMv4, and removed entirely in ARMv5. Arm provides a list of vendors who implement ARM cores in their design (application specific standard products (ASSP), microprocessor and microcontrollers).[105] Tronsmart MK908, a Rockchip-based quad-core Android "mini PC", with a microSD card next to it for a size comparison Main article: List of products using ARM processors ARM cores are used in a number of products, particularly PDAs and smartphones. Some computing examples are Microsoft's first generation Surface, Surface 2 and Pocket PC devices (following 2002), Apple's iPads, and Asus's Eee Pad Transformer tablet computers, and several Chromebook laptops. Others include Apple's iPhone smartphones and iPod portable media players, Canon PowerShot digital cameras, Nintendo Switch hybrid, the Wii security processor and 3DS handheld game consoles, and TomTom turn-by-turn navigation systems. In 2005, Arm took part in the development of Manchester University's computer SpiNNaker, which used ARM cores to simulate the human brain.[106] ARM chips are also used in Raspberry Pi, BeagleBoard, BeagleBone, PandaBoard, and other single-board computers, because they are very small, inexpensive, and consume very little power. An ARMv7 was used to power older versions of the popular Raspberry Pi single-board computers like this Raspberry Pi 2 from 2015. An ARMv7 is also used to power the CuBox family of single-board computers. See also: Comparison of ARMv7-A processors The 32-bit ARM architecture (ARM32), such as ARMv7-A (implementing AArch32; see section on Armv8-A for more on it), was the most widely used architecture in mobile devices as of 2011[update].[55] Since 1995, various versions of the ARM Architecture Reference Manual (see § External links) have been the primary source of documentation on the ARM processor architecture and instruction set, distinguishing interfaces that all ARM processors are required to support (such as instruction semantics) from implementation details that may vary. The architecture has evolved over time, and version seven of the architecture, ARMv7, defines three architecture "profiles": A-profile, the "Application" profile, implemented by 32-bit cores in the Cortex-A series and by some non-ARM cores R-profile, the "Real-time" profile, implemented by cores in the Cortex-R series M-profile, the "Microcontroller" profile, implemented by most cores in the Cortex-M series Although the architecture profiles were first defined for ARMv7, ARM subsequently defined the ARMv6-M architecture (used by the Cortex M0/M0+/M1) as a subset of the ARMv7-M profile with fewer instructions. Except in the M-profile, the 32-bit ARM architecture specifies several CPU modes, depending on the implemented architecture features. At any moment in time, the CPU can be in only one mode, but it can switch modes due to external events (interrupts) or programmatically.[107] User mode: The only non-privileged mode. FIQ mode: A privileged mode that is entered whenever the processor accepts a fast interrupt request. IRQ mode: A privileged mode that is entered whenever the processor accepts an interrupt. Supervisor (svc) mode: A privileged mode entered whenever the CPU is reset or when an SVC instruction is executed. Abort mode: A privileged mode that is entered whenever a prefetch abort or data abort exception occurs. Undefined mode: A privileged mode that is entered whenever an undefined instruction exception occurs. System mode (ARMv4 and above): The only privileged mode that is not entered by an exception. It can only be entered by executing an instruction that explicitly writes to the mode bits of the Current Program Status Register (CPSR) from another privileged mode (not from user mode). Monitor mode (ARMv6 and ARMv7 Security Extensions, ARMv8 EL3): A monitor mode is introduced to support TrustZone extension in ARM cores. Hyp mode (ARMv7 Virtualization Extensions, ARMv8 EL2): A hypervisor mode that supports Popek and Goldberg virtualization requirements for the non-secure operation of the CPU.[108][109] Thread mode (ARMv6-M, ARMv7-M, ARMv8-M): A mode which can be specified as either privileged or unprivileged. Whether the Main Stack Pointer (MSP) or Process Stack Pointer (PSP) is used can also be specified in CONTROL register with privileged access. This mode is designed for user tasks in RTOS environment but it is typically used in bare-metal for super-loop. Handler mode (ARMv6-M, ARMv7-M, ARMv8-M): A mode dedicated for exception handling (except the RESET which are handled in Thread mode). Handler mode always uses MSP and works in privileged level. The original (and subsequent) ARM implementation was hardwired without microcode, like the much simpler 8-bit 6502 processor used in prior Acorn microcomputers. The 32-bit ARM architecture (and the 64-bit architecture for the most part) includes the following RISC features: Load–store architecture. No support for unaligned memory accesses in the original version of the architecture. ARMv6 and later, except some microcontroller versions, support unaligned accesses for half-word and single-word load/store instructions with some limitations, such as no guaranteed atomicity.[110][111] Uniform 16 × 32-bit register file (including the program counter, stack pointer and the link register). Fixed instruction width of 32 bits to ease decoding and pipelining, at the cost of decreased code density. Later, the Thumb instruction set added 16-bit instructions and increased code density. Mostly single clock-cycle execution. To compensate for the simpler design, compared with

processors like the Intel 80286 and Motorola 68020, some additional design features were used: Conditional execution of most instructions reduces branch overhead and compensates for the lack of a branch predictor in early chips. Arithmetic instructions alter condition codes only when desired. 32-bit barrel shifter can be used without performance penalty with most arithmetic instructions and address calculations. Has powerful indexed addressing modes. A link register supports fast leaf function calls. A simple, but fast, 2-priority-level interrupt subsystem has switched register banks. ARM includes integer arithmetic operations for add, subtract, and multiply; some versions of the architecture also support divide operations. ARM supports 32-bit × 32-bit multiplies with either a 32-bit result or 64-bit result, though Cortex-M0 / M0+ / M1 cores do not support 64-bit results.[112] Some ARM cores also support 16-bit × 16-bit and 32-bit × 16-bit multiplies. The divide instructions are only included in the following ARM architectures: Armv7-M and Armv7E-M architectures always include divide instructions.[113] Armv7-R architecture always includes divide instructions in the Thumb instruction set, but optionally in its 32-bit instruction set.[114] Armv7-A architecture optionally includes the divide instructions. The instructions might not be implemented, or implemented only in the Thumb instruction set, or implemented in both the Thumb and ARM instruction sets, or implemented if the Virtualization Extensions are included.[114] Registers across CPU modes usr sys svc abt und irq fiq R0 R1 R2 R3 R4 R5 R6 R7 R8 R8_fiq R9 R9_fiq R10 R10_fiq R11 R11_fiq R12 R12_fiq R13 R13_svc R13_abt R13_und R13_irq R13_fiq R14 R14_svc R14_abt R14_und R14_irq R14_fiq R15 CPSR SPSR_svc SPSR_abt SPSR_und SPSR_irq SPSR_fiq Registers R0 through R7 are the same across all CPU modes; they are never banked. Registers R8 through R12 are the same across all CPU modes except FIQ mode. FIQ mode has its own distinct R8 through R12 registers. R13 and R14 are banked across all privileged CPU modes except system mode. That is, each mode that can be entered because of an exception has its own R13 and R14. These registers generally contain the stack pointer and the return address from function calls, respectively. Aliases: R13 is also referred to as SP, the stack pointer. R14 is also referred to as LR, the link register. R15 is also referred to as PC, the program counter. The Current Program Status Register (CPSR) has the following 32 bits.[115] M (bits 0–4) is the processor mode bits. T (bit 5) is the Thumb state bit. F (bit 6) is the FIQ disable bit. I (bit 7) is the IRQ disable bit. A (bit 8) is the imprecise data abort disable bit. E (bit 9) is the data endianness bit. IT (bits 10–15 and 25–26) is the if-then state bits. GE (bits 16–19) is the greater-than-or-equal-to bits. DNM (bits 20–23) is the do not modify bits. J (bit 24) is the Java state bit. Q (bit 27) is the sticky overflow bit. V (bit 28) is the overflow bit. C (bit 29) is the carry/borrow/extend bit. Z (bit 30) is the zero bit. N (bit 31) is the negative/less than bit. Almost every ARM instruction has a conditional execution feature called predication, which is implemented with a 4-bit condition code selector (the predicate). To allow for unconditional execution, one of the four-bit codes causes the instruction to be always executed. Most other CPU architectures only have condition codes on branch instructions.[116] Though the predicate takes up four of the 32 bits in an instruction code, and thus cuts down significantly on the encoding bits available for displacements in memory access instructions, it avoids branch instructions when generating code for small if statements. Apart from eliminating the branch instructions themselves, this preserves the fetch/decode/execute pipeline at the cost of only one cycle per skipped instruction. An algorithm that provides a good example of conditional execution is the subtraction-based Euclidean algorithm for computing the greatest common divisor. In the C programming language, the algorithm can be written as: int gcd(int a, int b) { while (a != b) // We enter the loop when a < b or a > b, but not when a == b if (a > b) // When a > b we do this a -= b; else // When a < b we do that (no "if (a < b)" needed since a != b is checked in while condition) b -= a; return a; } The same algorithm can be rewritten in a way closer to target ARM instructions as: loop: // Compare a and b GT = a > b; LT = a < b; NE = a != b; // Perform operations based on flag results if (GT) a -= b; // Subtract *only* if greater-than if (LT) b -= a; // Subtract *only* if less-than if (NE) goto loop; // Loop *only* if compared values were not equal return a; and coded in assembly language as: ; assign a to register r0, b to r1 loop: CMP r0, r1 ; set condition "NE" if (a ≠ b), ; "GT" if (a > b), ; or "LT" if (a < b) SUBGT r0, r0, r1 ; if "GT" (Greater Than), then a = a − b SUBLT r1, r1, r0 ; if "LT" (Less Than), then b = b − a BNE loop ; if "NE" (Not Equal), then loop B lr ; return which avoids the branches around the then and else clauses. If r0 and r1 are equal then neither of the SUB instructions will be executed, eliminating the need for a conditional branch to implement the while check at the top of the loop, for example had SUBLE (less than or equal) been used. One of the ways that Thumb code provides a more dense encoding is to remove the four-bit selector from non-branch instructions. Another feature of the instruction set is the ability to fold shifts and rotates into the data processing (arithmetic, logical, and register-register move) instructions, so that, for example, the statement in C language: a += (j