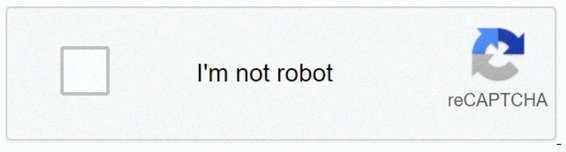
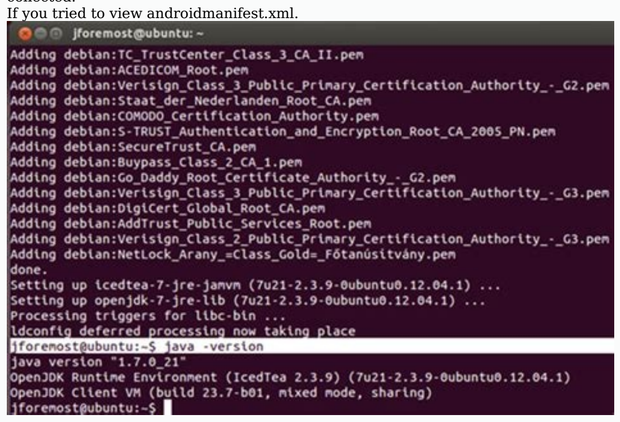


Continue

Apktool how to use

How to use apktool in windows. How to use apktool in windows 10. How to use apktool in kali linux. How to use apktool on android. Apktool how to use mac. How to use apktool m. How to use apktool to decompile apk. How to use apktool in linux.

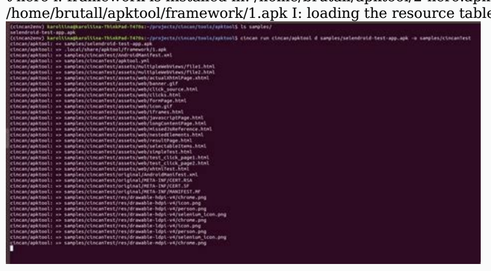
Reverse engineering tool for third -party patented Android application binary files. It can decipher the resources almost in their original form and restore them after a certain change. It also facilitates the app by automating a project, such as file structure and some repeat tasks, such as APK development, etc. It is not for piracy and other illegal purposes. It can be used for localization, adding some features, or supporting the platform for support, app analysis and more. APTOOL is licensed according to the Apache 2.0 license - see the license file for more details. First, let's take the APK. There is nothing more than a zip file with sources and a Java code. If you unpack the APK file as follows, you will be left with files like Class.dex and Resources.arsc. $ Unzip Archive TestApp.apk: TestApp.apk Bloat: Androidmanifest.xml Blot: Class.Dex Extract: RES/Drawable-HDPI/IC_LAUNCHER.PNG Blot: RES/XML/Literals.ml Blot: RES/XML/References.XML : Resources.ARSC But at this point you have simply increased the number of resources collected.
If you tried to view androidmanifest.xml.



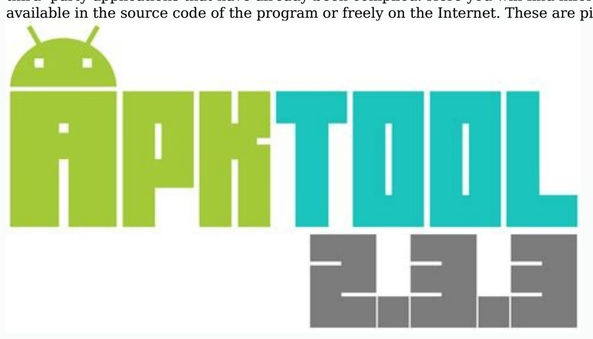
You would see it. P 4 F 0 \ F N V E R S I O N C O D E R S I O N N A M E N D R O I D * H T P: // S C H E M A S. A n d r o i d. C o m / a p k / r e s / a n d r o i d p a c g e p l a t f o r m b u l d v e r s i o n c o n c o d e p l a r m b u l t s s a d. 0 2 1 A p k t o o l, of course, followed by modifying or reviewing the compiled file as far as it is not possible. Aptool is used here. $ aptool d TestApp.apk i: Aptool 2.0.0 TestAPP.Apk Usage I: Reload of Resource Table ... I: Androidmanifest.xml Decoding withA tool to reverse engineer third-party Android application binaries. It can decode resources in almost their original form and restore them after making some changes. Also, it makes the app easier to use due to design like file structure and automation of some repetitive activities like APK development etc.
It is not intended for piracy and other illegal purposes. Adding some features or supporting a custom platform can be used for localization, app analysis, etc. An APK is nothing more than a ZIP file that contains assets and compiled Java code.
If I only mutilate the APK this way, files like Class.Dex and Resources.ARSC will remain. $ Unpack testapp.apk Archive: testapp.apk Extension: Androidmaniifest.xml Extension: Classe.dex. Extract: Res/Drawable-hdpi/ic_launcher.png Extension: res/xml/literals.xml Extension: res/xml/links. :Resources.Arsc, but at this point you've just faked the collected resources. If you tried to view the androidmanifest.xml. You will stay to see it. P 4 f 0 \ f n v e r s i o n c o d e v e r s i o n a m e n d r o i d * h t t p: / s c h e m a s. Android. C O M / A P K / R E S / A N D R O I D P A C K A G E L A T F O R M B U I L D V E R S I O N C O D E P L A T F O R M B U I L D V E R S I O N N N A M A M A N I F E S B R U T. xml withI: Download Source Table from File: 1.APK I: Total Manifesto Pack ... I: Decipher the File Resources ... I: Decode Values*/*XMLS ... I: Baksmaling Class.dex ... I: I: I : I: I: I: I: Copy Source and Libraries ... Re -checking $ androidmanifest.xml is obtained something much more usable < "Packe =" Brut .apktool.testapp ""
PlatformbuildversionCode = "21" "PlatformBuildversename" = "APTOTOOL"/> In addition to XML Phylas, such resources are available. Corrections, layouts, lines and much more are properly deciphered in their original form. The decoding parameter decoded the Aptool can be called from D or from decoding as shown below. $ aptool d foo.Jar // Defoding the file foo.Jar to foo.Jar.ou $ aptool deciphered foo.Jar // Debt Foo.Jar to foo.Auta $ aptool d Bar.apk // Debt Bar.apk im Bar -Mape $ aptool decipher Bar.apk // Decipher the Bar.Apk bar in the folder $ aptool d bar.apk -o baz // decipher the Bar.Apk to the base, the assembly of the building can be activated from B or from assembly. Or from assembly, as shown below $ aptool b fooo.Jar.out // Create a folder foo.Jar.out foo.Jar.ot/Dist/Foo.Jar $ Build Foo.Jar.out // Create a folder foo. jar.
Out Foo .jar.out/Dist/Foo.Jar File $ APTOOL B BAR // Bar Files on Bar/Dist/Bar.Apk $ Aptool B. // Create Current Directory ./Dist $ aptool b bar-o new_bar.apk // bar new_bar.apk $ aptool b bar.apk // brut.androlibexcation: brut.directory.pathesist .yml // to use the folder, and no APK/JAR file to run an unfair application. You have to cancel the program.
Android documentation can help with this. Freimvorki frameworks can be installed using an IF or installation frame -an additional two parameters -P, -Frame -Parath

-Forework files -Tag -Framework. with allows you to control more accurately which files are accessed and how they are accessedShop. $ 1 FrameWork-rés.apk i: FrameWork installed: 1.Apk // pkgid framework-re.Apk defines a number (from 0x01) $ clogool if com.htc.resources.apk i: FrameWork installed: /pkgid de com.htc.Resources is 0x02 $ impeller if com.htc.Rerces.apk -t htc i if framework- res .p -p -foo/bar i PLUCOL if the system -res.apk -baz -p foo/bar i: Framework installed: Foo/Bar/1 -Base. AK FRAMEWORK files, as you probably know, Android apps use the Android operating system code and resources itself. They are known as the Framework resources, and the Kakurolo resources are well founded to decipher the APK and proper assembly. Each version of Killsol has the latest version of AOSP at the time of release. So you can easily decipher and create the most APK.

However, manufacturers add their main files in addition to the usual AOSP files. You must first install the Framework files to use the Kakurol before this manufacturer. An example of saying, you want to decipher htccontacts.apk from the HTC device. If you try, you will get an error message. $ KRUKUKOL D HTCCONTACTACTS.APK I: Resource table is loaded. Instead, the use of a decoded value: NS = Android, Name = Drackable W: It is impossible to decipher the ATR value, instead of packaging sources: 2. You must install the appropriate system and system and get additional information on the project website. We need to get the HTC Framework resources before decoding this APK. We pull out the Com.htc.Sources.apk and set the $ impeachol from our device if com.htc.resources.apk i: FrameWork installed: 2.PAPK now try this decoding. $ KRUKUKOL D htccontacts.apk i: Loading the resource tableI: Download resource table from file: /home/brutall/apkool/framork/2.apk I: Copy goods and libs...as you can see. Apctool used Framework 1.APK and 2.PK files to properly decrypt this app. What most APKs look for in the /system/framework folder on your device is the Framework file. On some devices, you can end up with a data/systems partition and even cleverly hide /system/application or /system/privateapplication. They are usually called "resources", "resu" or "frameworks". Example HTC has a system named com.htc.Resources.apk, LG named LGE-RES.APK after discovering the system file. You can be used via ADB Pull/route/file or a file manager application. After the local file, pay attention to how Apkool will install you. The number with which the system is accessed during installation corresponds to the PKGID of the application.
These values should range from 1 to 30. Each APK installed as 127 is 0x7f, which is the internal PKGID. As mentioned above, Apkool comes with an internal system. This file is copied to $home/apkool/framework/1.APK during use. Note that Apkool has no knowledge of the version of the system there. Assuming it is updated, delete the file during Apkoloon updates as the system archives at different points depending on the operating system in question. Unix - $home/.local/share/apktool windows -%userprofile%\appadata\local\apktool mac -$/library/apkool If these directories are not available, they are shown in java.oio.tmpdir, this is predefined. Usually /tmpt. This is a temporary directory, so it would make sense to use the fracta-Poth parameter to select an alternate folder for the Framework files. Since these positions are sometimes hidden storage, managing these paintings becomes a challenge. With a simple support function (added in version 2.2.1), you can run APKTool Leer-Fracta sale to empty the frame. Please note that the system is not endorsed by ApkoolHowever, you can manage these files.
Frameworks are archived in the order of the names: - .apk. They are identified by PKGID and optionally a custom day. Systems do not normally have to be marked, but if you work with apps on different devices and have incompatible schemas, you need a way to simply switch between you. You can tag systems: $ apktool if com.htc.resources.apk - t hero i: framework is installed in: /home/brutall/apktool/2-hero.apk $ apktool if com.htc.resources.apk - T Wish I: Framework installed in: /home/brutall/apktool/framework/2-desire.apk Then: $ apktool D htccontacts.apk -t hero i: resource table is loaded ... i: resources are deciphered ... i : Loading the resource table from file: /home/brutall/apktool/framework/1.apk I: loading the resource table from file: /home/brutall/apkool/framework/2-hero.apk I: copying resources and libraries .. . $ APKTOOL D HTCCONTACTS.APK -T Wish I: Load resource table ...



I: Decode resources ... I: Loading the resource table from File:/Home/Brutall/APKOOL/1 .apk I: Load resource table from File: /home/brutall/apktool/framework/2-desire.apk I: Copy resources and libraries ... When creating an APK, no day has to be selected: APCTOOL basically uses the same day as with the Deco dieren. Smali Debugging Warning Smalidebugging was out of date in version 2.0.3 and was removed from version 2.1. Check out Smaliidea for the debugger. 9patch images Documentation for cryptic 9patch pictures are available here and there. (Read them first). However, this documentation is aimed at developers and does not contain any information for those who work with third -party applications that have already been compiled. Here you will find information about how you create it, but not about how you actually work. I will try to explain it here. One thing that is missing in the official documentation is that 9Patch images are available in two variants: source and compiled. Source - you know. You can find it available in the source code of the program or freely on the Internet. These are pictures ofA border around them. Collected: A mysterious form found in APK files.



There are no boundaries and 9Patch data is stored in a binary chunk called NPTC. It cannot be easily viewed or edited, but the Android operating system can read it. There are problems with the above two points. 9Patch images cannot be transferred between these two types without conversion.

If you try to unpack the 9ppatch images and use them in another source, you will get errors during assembly. Conversely, you cannot transfer 9-ATC source images directly to an APK file. The 9patch binary is not recognized by modern image processing tools. Thus, modifying the compiled image is likely to result in an NPTC fragment, corrupting the image on the device. The only solution to this problem is to easily convert between these two types. The encoder (which downloads the compilation source code) is built into AAPT and is automatically used during compilation. This means that we just need to create a decoder that will be around from version 1.3.0 and will automatically run on all 9ppatch paintings when decoding. So if you want to edit your 9ppatch images, don't do it directly.
Use Apktool to decrypt the application (including 9Patch images) and then edit the images. Now we will create the 9Patch Source images while building the application. FAQ and what about the switch shown in the original YouTube videos? Read number 199. In short - no. Can Apktool boot your device? Unfortunately, no.
There is an inconsistency with running AAPT binaries on the device. Also, older versions of Android have problems with Java.nio. Where can I download Apktool sources? From our GitHub or Bitbucket project. The resulting APK file is much smaller than the original! Is there something missing? There are several reasons that can cause this. Apktool creates an unallocated apoty. This means that the entire meta-inin directory is missing. New Aapt binary. Newer versions of Apktool contain newer AaptOptimize images in other ways. These points could contribute to a smaller APK file as usual since the resulting APK does not have the Meta-Inf director. OK? YES. Meta-inf contains APK signatures. The APK is no longer signed after the change. You can use -c / -copy -opinal to keep these signatures. However, using the -C option causes the original AndroidManifest.xml file to be used, so changes to them are lost in it. How do you say "Apk Magic"? For some reason there are APK files that are built with changed build tools. These APK files do not work for normal AOSP Android compilation, but are usually provided by a modified system with which these modified APK files can be read. Aptool cannot support these APK files, so they are "magic". Can I integrate apkool in my tool?
Can I change Apktool resources? Do I owe you a loan? In fact, the Apache license that Aptool uses answers all these questions. Yes, you can redistribute and/or change aptol without my permission. However, if you do, it would be nice to add to the loans of our colleagues (Brut.Tutti, Ibotpeaches and Jesusfrex), but it is not necessary. Where does Apkool store its framework files? Unix -$ home/.local/share/aptool mac -$ home/library/aptool Windows -$ home/appData/local/aptool -migrate instructions v2.6.1 -> v2.7.0 -dir -force v2.5.0-> v2 . 6.0 Apkool to V2.6.0 Empty Aptool Framework-for-Force V2.4.1-> V2.5.0 Update aptol to v2.5.0 aptool emplo-framework-force-force v2.3.0-> v2.3.1 aptool to v2.3.1 Empty Aptool Framework-for-Force v2.2.4-> v2.3.0 Update aptol to v2.3.0 empty Aptool Framework-for-Force V2. 2.0-> v2.2.1 Update aptool to v2.2.1 Empty aptool fremework-force v2.1.1-> v2.2.0 Issue the following commands to properly migrate apttool framework catalog migration without these commands, i.e. just removed. UNIX files -Mkdir -p ~/.local/share; MV ~/Aptool ~/.Local/Share Windows - transfer %user profiles%\ AptoolV2.0.1 -> v2.0.2 Aptool upgrade to v2.0.2 Delete system $home/apktool/framework/1.apk due to API update (Android Marshmallow) v1.5.x -> v2 .0.0 Java 1.7 required Apktool will updated As of version 22.0.0, AAPT is now included in Apktool binaries. The AAPT installation does not have to be considered as path $. (But functions like -A / -AAPT are still used and can replace internal AAPTs) The addition of AAPT replaces the need for separate AAPT download packages. Helper scripts are here, remove framework $home/apktool/framework/1. APK Eagle users will notice that resources are now decoded before sources. This is because we need to know the API version that the manifest uses to decode changes to the Smali / Baksmali 2.0 sources. This is a big change with 1.4.2. For more details read here SMALI updates -O / - output is now used to exit APK / directory -T / - framework files -Advance / -Advanced introduces settings and prepaid output information -M / -Mek -Original is the new APK parsing feature. This APK almost keeps the original format, but most likely the recovery will work as the changes required by the new AAPT after ecode[d] are ignored and new folders (original/unknown) appear in the decoded APK Original = Meta-Inf folder /AndroidManifest.xml, required to keep the APK signature to avoid termination. Used with -c / -copy -original [b] Uild Unknown = Files/folders that are not part of the standard AOSP compilation procedure. These files will be embedded in the recovered APK. Apktool.ml collects more information than the latest version of sdkinfo - it is used to push the SDK information to AndroidManifest.xml because in the latest AAPT version information needs to be passed through the Papertinfo parameters - it is used to to support the manifest function in android. 4.2. Automatically detect differences between resources andAnd launches the Rename-Manifest automatic package for the [b] uld version, which is used to update the version of the version in Androidmanifest.xml, since the new AAP in the source APK file is used to play [b] Uld during unknown files. For storing the name/location of files that are not a dump APK properly placed in the APK APK APK file, is used to support the function of the Android 5 library, which is automatically distributed and using common libraries-Hared-Lib by [b] uld Examples New Use 2.0 versus 1.5.x old (Apktool 1.5.x) new (Apktool 2.0.x) apktool, if framework-re.apk, mark apktool if Framework -es. APK -t Apktool d Framework -resaler.apk Conclusion Apktool d Framework.res.apk -o or Apktool B remove New.aptool B Out-o new.apk v1.4.x -> V1 . 5.1 Windows Mac, S or Linux updating AAV $ Home / Apktool / Framework / 1.APK OPTIONS OF THE FREMWER FIRMOVOR OPTIONS, which can be launched at any time. -Aizas, - the current version of the output. (Example: 1.5.2) -v, - verbal conclusion.
The first parameter should be -Q, -quiet output and silence. The first parameter should be - Advance, - Advanced Expanded use of the DIR video output is all options for compensating for the Framework catalog. -F, - forcibly remove the target catalog. -P, -frame-paath

Location for downloading files from Dir de List.
Information about the framework is added in version 2.5.0, these are all the parameters when reporting the Freimvorka catalog. -P, -frame-paath place where the main decoding files will be uploaded is all options for APK decoding. -API, -COUCH LEVEL SMALI API files to generate API files (default value-Targetsdkversion)).
If so, then priority is usedRemove them from the same APK version. The number and mix of different versions will change, which can cause problems with different reports. -F, -Force force delete target directory. If you are trying to decrypt in an existing folder, use -Forifest -v2.3.1, which is used to force decrypt AndroidManifest for parsing, regardless of the resource's decode flag. This will most likely prevent the analysis of the static description from being restored. -Kast-Break-Res-Advanced If you get an error like "incorrect configuration flags are detected.
This means that the structure of the APK is different from Paddle. This may be a newer version of Android or a non-standard case APK. If you do this, you will have should be decrypted, but then you'll have to repair the folders manually with -Er -M, -igch -original -used to parse so files are as close as possible but prevents recovery -No-Assets-V2.3.0 Does not allow decryption / unknown a copy of the resource files. -O, -fed

APK folder name in dates -of -Main class -IK v2.4.1 -Frame -Path

Folder location to store / scan frame files from -r should This is way the file resource.ARSC remains intact without it.Editing only Java (Smali) is the recommended action for hunting and rebuilding faster -s, -no -c, it prevents messing up DEX files.It has APK DEX files and they are simply transferred to build If you are only editing resources This is the recommended action to disassemble and reassemble u faster -t -frame -tag Use system files marked with Remuild. These are all CPA options. -A, -aapt loads AAPT from the specified file location without relying on the path. Fallback loading $Path if file not found. Unless the $path points to an existing custom Aapt. it will beWhich most likely won't work. -api, --Api -level -V2.4.0 Small API level of files to create (by default Minsdk Version) -C, - -Copy -Original -remove -v3.0.0 copy the original Android parse. XML and Meta-INF folder in the embedded APK. Planned depreciation. -d, -debug -v2.4.0 adds debugablet="true" to androidmanifest. -F, -Force-All overwrites existing files at creation time creating Resources.arsc file and Dex files. -N, -NET-SEC-CONF-V2.6.2 adds a general network security configuration file for APK output. -NC, -No-CRUNCH-V2.4.0 disable resource file crunching during build. -o, -output apk name and location for saving to -p,--frame-path space where frame files are loaded from --US-AAPT2-V2.3.2 Use the instead app of the Binär AAPT2 application