Zugriff auf Android-Systeme mit ADB

ADB, die Android Debug Bridge, ist ein Standard-Werkzeug für die Anbindung und Vernetzung von Android-basierten Systemen mit einem Desktop/Entwicklungssystem über den bei allen Smartphones und Tablets vorhandenen USB-Port (lokal sicher) oder per Netzwerk (potenziell unsicher, da nicht passwortgeschützt).

ADB-Service unter Android (Server-Seite)

Standardmäßig ist ADB ausgeschaltet, kann aber z.B. durch Aktivieren des „Entwickler-Modus“ (7 mal auf die Build-Nummer in den Android-Einstellungen tippen) und Auswahl des „Debug-Modus“ in den daraufhin sichtbaren „Entwickler-Einstellungen“ freigeschaltet werden.

Beim Anschluss des Smartphones an einen Computer via USB-Kabel erscheint ein Dialog, der eine Bestätigung des ADB-Zugriffs unter Angabe der MAC-Adresse des Computers erfragt. Wird dieser Dialog bestätigt, stehen die ADB-Funktionen zur Verfügung.

ADB-Client unter Linux, Windows, Mac

Der ADB-Client steht für verschiedene Plattformen zur Verfügung und ist nicht Linux-spezifisch. Hier wird aus praktischen Gründen zunächst die Linux-Version behandelt, die anderen Plattformen verhalten sich analog. Unter Windows kann die Installation eines seriellen USB-Treibers erforderlich sein, bei Linux ist dieser im Kernel als „usbserial“ bereits integriert.

Das Paket `android-tools-adb` unter Debian and Ubuntu enthält das „`adb`“.Kommando. Die Installation des vollständigen Android SDK ist NICHT erforderlich.

```
sudo apt-get install [-t unstable]  android-tools-adb
```

Adb kann verwendet werden, um Apps und deren Daten zu archivieren, Apps zu installieren oder zu löschen, Dateien in beide Richtungen zu übertragen und administrative aufgaben unter Android zu erledigen, die es nicht als „App“ gibt.

Tipp: Selbst wen ein Smartphone in einer „Endlosschleife“ festhängt und nicht mehr startet, ist es oft über adb möglich, fehlerhafte Software zu entfernen oder Konfigurationen zu korrigieren, und somit das Smartphone wieder zum Starten zu überreden.

Beispiel:

```
adb root                    (Administrator werden, nur bei gerooteten
                             Smartphones)
adb shell setenforce 0      (Secure Linux Richtlinien in den „permissive“
                             Modus schalten, hiermit starten dann auch solche
                             Apps, die zuvor aufgrund von falsch gesetzten
                             Rechten abstürzten)
```

Mit „adb help“ im Terminalfenster wird eine kurze Hilfe ausgegeben, in denen aber nicht alle unterstützten Kommandos auftauchen.

Command	Description
`adb root`	Give root permissions for subsequent commands on the smartphone (requires a rooted phone and may ask for permission)
`adb shell`	Log in to a remote shell on the smartphone, feels almost like a shell on a desktop computer (cd, ls, …, type “exit” to leave)
`adb backup/restore ...`	Backup/Restore apps and app data in encrypted format.
`adb exec-out command`	(yet undocumented feature!): run command on Android and capture its unfiltered binary output (stdout)
`adb exec-in command`	(yet undocumented feature!): run command on Android and feed the unfiltered console input (stdin)
`adb help`	Short help (manpage is more verbose)

Die adb-Kommandos **exec-in** und **exec-out** findet man derzeit in keiner Dokumentation von adb, aber nach Lektüre des Quelltexts und durch Postings in Foren wird klar, dass es sich dabei um sehr nützliche Features handelt. Adb shell hat nämlich einen „Terminal-Filter“ für nicht-druckbare Zeichen vorgeschaltet, der bestimmte Sonderzeichen als Formatierungskommandos behandelt und durch entsprechende Codes des Terminals ersetzt. Dieses Verhalten ist für den Transfer von Binärdaten kontraproduktiv, sie werden dadurch verändert, z.B. zip- oder tar-archive als „Pipe“ mit Ausgabeumlenkung werden auf dem Zielsystem unbrauchbar ankommen. „exec-in“ und „exec-out“ erlauben zwar keine Interaktion mit dem gestarteten Programm, aber transferieren Daten unverändert.

Beispiel: Backup der Benutzerdaten per adb.

`adb root`
(works only for rooted phones)

```
adb exec-out "GZIP=-1 tar -zcpf - /data 2>/dev/null" > android-data.tar.gz
```

Explanation:

`GZIP=-1  tar -zcpf -`
creates (option -c) and streams a tar archive to standard output (option -f -), which is compressed (option -z, with GZIP level 1 for fastest compression), while also archiving the secure linux contexts and special permissions (option -p). The secure linux contexts, btw., are important for apps in order to get access their specific data files, since Android in “strict” SELinux mode will in fact deny access to files if these special modes are not set correctly, causing apps to crash. Of course, it’s a security feature, so tar on newer Android versions has the (also yet undocumented) -p option for storing additional permission information.

`/data`
is the partition containing settings and app data (the stuff you want to definitely keep, even if you decide to upgrade the `/system` partition with a new Android version).

`2>/dev/null`
just discards error messages. If the command terminates very quickly, you may want to remove that option to see what happened (maybe the -p option is not supported in older Android versions), but error messages and notices can pollute the tar archive data if inserted to the stream, so it’s recommended to send them to /dev/null if the command is otherwise OK.

`> android-data.tar.gz`
sends the captured tar archive stream from standard output to the file “android-data.tar.gz”. No intermediate data is stored on the Android side.

In order to check the archive for contents, run

`tar -ztvf android-data.tar.gz`
(no adb here, it’s local!)

which should show you the archive contents.

Restaurieren:

In order to restore parts of the archive back to the phone, the command is

```
adb exec-in “tar -zxpf - data/media/0/Music” < android-data.tar.gz
```

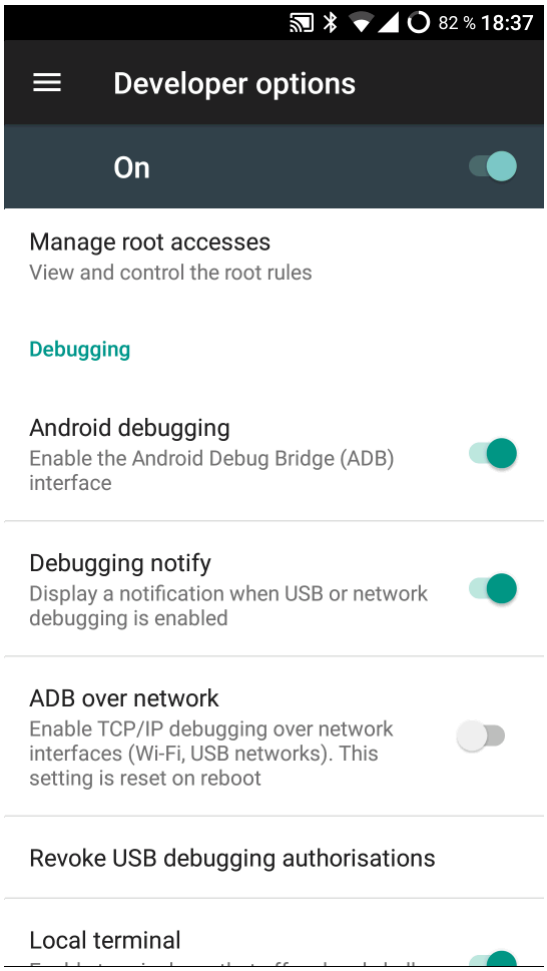
Explanation:

`tar -zxpvf -`
decompresses (option -z), extracts (option -x) and restores data from the archive streamed as input (option -f -), also restoring secure linux contexts (option -p).

`data/media/0/Music`
The only part of the archive you want to restore. Unless you are recovering from a crash or after a “factory reset”, you probably don’t want to restore the entore archive, since newer versions of settings and files would get overwritten. Default (without specifying a directory) is to extract the entire archive.

`< android-data.tar.gz`
Feed the stored archive as input stream to adb.

For transfering single files or directories from and to Android, “`adb pull filepath`” and “`adb push filepath`” are also useful, but not as flexible as tar.


82 % 18:37
Developer options
On
Manage root accesses
View and control the root rules
Debugging
Android debugging
Enable the Android Debug Bridge (ADB) interface
Debugging notify
Display a notification when USB or network debugging is enabled
ADB over network
Enable TCP/IP debugging over network interfaces (Wi-Fi, USB networks). This setting is reset on reboot
Revoke USB debugging authorisations
Local terminal