Linux and Android HOWTO

Table of Contents

Linux and Android HOWTO

Guido Gonzato, Ph.D. `guido.gonzato at gmail.com`

March 2016, version 1.0.0

This HOWTO provides information on how to manage your Android device using a GNU/Linux system. It's aimed at reasonably competent Linux users who want more control over their Android device.

---

1. Introduction

I started this document at the end of 2014, over 15 years after my previous HOWTOs. In the meantime, GNU/Linux on the desktop has basically failed, but a GNU-less Linux variant has gained a large market share in the mobile industry. It's Android: a partially open platform, based on the Linux kernel and a mix of open and closed programs (``apps''). As a matter of fact, Android is by far the most widespread Linux distribution, even though most of its users don't even know they're using Linux!

(A pocket-size Unix system that one can get for less than 100 bucks. Amazing! If they'd told me about it in the early '90s, I would have dismissed it as cheap science fiction.)

I purchased an Android phone and a mid-spec Android tablet, and I wondered if I could do something interesting with them --- apart from making phone calls and browsing the net, that is. Android is meant to be very easy to use, and no trickery is required for normal use. The thing is, I'm not a normal user: I'm a GNU/Linux sysadmin, and just scratching the surface is not enough for me. So I started experimenting and gathering information.

1.1 Purpose of this guide

This guide is meant to be a quick reference for GNU/Linux users who want to use their Android device in less-than-trivial ways, and want their Linux and Android boxes to talk to each other. Most vendors provide drivers and ancillary programs for Microsoft Windows or Apple OS X only, but a GNU/Linux system is perfectly capable of interacting with Android devices. With a bit of hacking, as usual.

In the following, I will share a few tricks I have collected:

- using the Android Debug Bridge;
- understanding and using the Android file system;
- installing programs from other markets;
- copying and syncing files and directories;
- backing up stuff;
- using a terminal emulator;
- compiling native command-line programs;
- remote control to/from Android;
- and more.

I work on a GNU/Linux Mint box, but I'll try and be distribution-agnostic. I'll concentrate on Free and Open Source Software (FOSS) whenever possible, and I'll take standard, unrooted Android devices into account. By the way: ``rooting'' means tweaking your Andbox to gain root permissions, as you would do in Linux with `sudo`. ``Unrooted'' means ``not tweaked''.

Since Android is very fragmented (hey, it's Linux after all!) and several versions are available, I'll just provide information that is applicable to what I own: Android 4.0.4, 4.1.1, and 5.0.1 on ARM architecture. All examples and code in this HOWTO were actually tested on my devices; hopefully, they should work on your device too. If you want me to cover more Android versions, I'll be glad to receive new equipment; or just tips. Equipment is preferred :-)

Currently, the majority of Android devices are ARM based; others are based on x86 or MIPS CPUs, in both 32 and 64 bit flavours. This is not a significant difference: most applications are written in Java with no native code, so they are CPU-agnostic. Instructions in this HOWTO should work for these Android versions, too. I guess that relevant differences concern security policies.

Should you have trouble with your Android device, I suggest that you refer to Android forums. Among the many available, I find the following especially helpful:

- http://forum.xda-developers.com/
- http://forums.androidcentral.com/

Besides: if you find any errors in this guide, please report them to me.

Thoughout this HOWTO, all instances of ``Linux'' actually mean ``GNU/Linux''. The GNU part is very important, and I'm very grateful to GNU for its fantastic programs. ``Andbox'' will stand for ``Android device''.

1.2 Requirements

I shall assume that you are a reasonably competent Linux user: you must be able to open a terminal, issue commands, become root, edit files, compile and install software. No spoon-feeding here.

As far as Android expertise is concerned, only the very basics are required. You are expected to be able to perform common tasks such as installing software, enabling USB debugging, using Bluetooth, and so on. Nothing special, really: in general, you'll have to be able to find out where options are in your device. I would be glad to provide information, but unfortunately no identical menus or screens can be found across different devices of different brands and different Android releases. You will have to figure it out yourself.

Finally: rooting your device might be desirable, but it's not necessary as far as this HOWTO is concerned. Root permissions are normally forbidden in Android, unless an enlightened vendor decides otherwise. I will mention a great program that needs root access (Webkey), but the rest of recommended software will not need it. By the way: in the following, I'll use the appropriate term ``program'' or ``application'', not the marketing term ``app''. (Yep, I'm an old and grumpy guy.)

2. The basics

To begin our journey, you will have to install some essential programs. Please note that Linux-side programs might be available as native packages for your distribution, i.e. as `.rpm` or `.deb` archives.

The very first thing you should do is enable USB debugging; this feature is found under ``Settings'', ``Developer options''. If your device lacks this entry (shame on the vendor!), you can enable developer options by tapping 7 times --- I'm not kidding! --- on the ``Build version'' menu.

2.1 Linux ADB (Android Debug Bridge)

ADB is a command line tool, installable on your Linux box. It lets you communicate with an Andbox, usually connected via USB, in order to perform a wide range of operations.

If you're lucky, your distribution may include a package called `android-tools-adb`, which contains the `adb` command. If not, get the Android SDK Tools for Linux from:

http://developer.android.com/sdk/

and find out how to install it on your Linux box.

I suggest that you open a terminal and run `adb` with no options to get an information screen. You should become familiar at least with the following options:

- `adb devices`: list connected devices
- `adb push`: copy file/dir from Linux to device
- `adb pull`: copy file/dir from device to Linux
- `adb shell`: open command-line shell on the device
- `adb install`: install a `.apk` on the device
- `adb backup`: perform device backup
- `adb restore`: perform device restore

Before using the `adb` commands, you must make your Andbox visible to the Linux box. Enable USB debugging on your device and connect it via USB cable to your Linux host; the Andbox will probably make a sound and/or flash the screen and/or ask for your permission. Issue this command:

```
Linux:~$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
245a16e41fe71a95427cc4e65d36cc9f        device
```

If you don't see any output after ``List of devices attached'', or if a row of `?' is displayed, some steps and some patience are required to make your device visible. First of all, find out your device's Vendor ID and Product ID:

```
Linux:~$ lsusb
...
Bus 001 Device 004: ID 1e68:0072 TrekStor GmbH & Co. KG
...
```

In this example, `1e68` is my tablet's Vendor ID, while `0072` is the Product ID.

I assume that your Linux distribution uses `udev`. As root, edit the file:

```
/etc/udev/rules.d/50-android.rules
```

(create a new file if it's missing) and add this line:

```
SUBSYSTEM=="usb", SYSFS{idVendor}=="1e68", MODE="0666"
```

Change your Vendor ID and username as necessary, save the file and restart udev:

```
Linux:~$ sudo udevadm control --reload
```

Some devices may also need this additional step:

```
Linux:~$ mkdir $HOME/.android
echo "0X1E68" >> $HOME/.android/adb_usb.ini
```

that is, `0X` (zero-ex) followed by your Vendor ID, all uppercase. Now unplug the USB cable, plug it in again, and run the commands:

```
Linux:~$ adb kill-server
Linux:~$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
245a16e41fe71a95427cc4e65d36cc9f        device
```

Now your Andbox should be visibile. If it's still not recognised, try all the above as root. If it still doesn't work, there might be a problem with your Andbox; you should consult your device's vendor.

If you have more than one device, you might have to restart the `adb` service to make the new Andbox visible when you plug it:

```
Linux:~$ adb devices
List of devices attached

Linux:~$ # no luck. Unplug the device
Linux:~$ adb kill-server
Linux:~$ # plug the device
Linux:~$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
047011c34260a417        device
```

2.2 The Android file system

Now we can talk to the Andbox via `adb`, so let's have a look at the Android file system hierarchy. I assume that you're familiar with the standard Linux directory hieararchy; Android's is fairly similar, with a couple of important differences.

Connect your Andbox and run the `adb shell` command, which launches Android's internal shell. Please be aware that Android's native shell is much less sophisticated than `bash`; available commands are not made by GNU either, so you should expect minor differences.

```
Linux:~$ adb shell
shell@android:/ $ ls -l
drwxr-xr-x root     root              2015-10-19 09:48 acct
drwxrwx--- system   cache             2015-10-19 10:52 cache
dr-x------ root     root              2015-10-19 09:48 config
lrwxrwxrwx root     root              2015-10-19 09:48 d -> /sys/kernel/debug
drwxrwx--x system   system            2015-10-06 07:53 data
-rw-r--r-- root     root          116 1970-01-01 01:00 default.prop
drwxr-xr-x root     root              2015-10-19 09:48 dev
```

```
lrwxrwxrwx root     root              2015-10-19 09:48 etc -> /system/etc
-rwxr-x--- root     root        98692 1970-01-01 01:00 init
-rwxr-x--- root     root         7272 1970-01-01 01:00 init.antares.rc
-rwxr-x--- root     root         2344 1970-01-01 01:00 init.goldfish.rc
-rwxr-x--- root     root         2820 1970-01-01 01:00 init.nv_dev_board.usb.rc
-rwxr-x--- root     root        17549 1970-01-01 01:00 init.rc
drwxrwxr-x root     system            2015-10-19 09:48 mnt
drwx------ root     root              1970-01-01 01:00 modules
dr-xr-xr-x root     root              1970-01-01 01:00 proc
drwx------ root     root              2012-08-30 12:39 root
drwxr-x--- root     root              1970-01-01 01:00 sbin
lrwxrwxrwx root     root              2015-10-19 09:48 sdcard -> /mnt/sdcard
drwxr-xr-x root     root              2015-10-19 09:48 sys
drwxr-xr-x root     root              2014-01-15 13:23 system
-rw-r--r-- root     root         1308 1970-01-01 01:00 ueventd.antares.rc
-rw-r--r-- root     root          272 1970-01-01 01:00 ueventd.goldfish.rc
-rw-r--r-- root     root         3825 1970-01-01 01:00 ueventd.rc
lrwxrwxrwx root     root              2015-10-19 09:48 vendor -> /system/vendor
shell@android:/ $ _
```

Minor differences are possible; for example, in the above screenshot I omitted a directory that is apparently tied to a specific vendor. Note that all directories except three belong to `root:root`; as a consequence, access to their contents will be limited on unrooted devices.

Now run the `mount` command, to see what devices correspond to what directories (line-broken for readability):

```
shell@android:/ $ mount
rootfs / rootfs ro,relatime 0 0
tmpfs /dev tmpfs rw,nosuid,relatime,mode=755 0 0
devpts /dev/pts devpts rw,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,relatime 0 0
tmpfs /mnt tmpfs rw,relatime,mode=775,gid=1000 0 0
debugfs /sys/kernel/debug debugfs rw,relatime 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
tmpfs /mnt/asec tmpfs rw,relatime,mode=755,gid=1000 0 0
tmpfs /mnt/obb tmpfs rw,relatime,mode=755,gid=1000 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
/dev/block/platform/sdhci-tegra.3/by-num/p3 /system ext4 \
  ro,relatime,user_xattr,acl,barrier=1,data=ordered 0 0
/dev/block/platform/sdhci-tegra.3/by-num/p7 /data ext4 \
  rw,nosuid,nodev,noatime,errors=panic,user_xattr,acl,barrier=1,\
  journal_async_commit,nodelalloc,data=ordered 0 0
/dev/block/platform/sdhci-tegra.3/by-num/p4 /cache ext4 \
  rw,nosuid,nodev,noatime,errors=panic,user_xattr,acl,barrier=1,\
  journal_async_commit,nodelalloc,data=ordered 0 0
/dev/block/vold/179:8 /mnt/sdcard vfat \
  rw,dirsync,nosuid,nodev,noexec,relatime,gid=1015,fmask=0002,\
  dmask=0002,allow_utime=0020,codepage=cp437,iocharset=iso8859-1,\
  shortname=mixed,utf8,errors=remount-ro 0 0
/dev/block/vold/179:8 /mnt/secure/asec vfat \
  rw,dirsync,nosuid,nodev,noexec,relatime,gid=1015,fmask=0002,\
  dmask=0002,allow_utime=0020,codepage=cp437,iocharset=iso8859-1,\
  shortname=mixed,utf8,errors=remount-ro 0 0
tmpfs /mnt/sdcard/.android_secure tmpfs ro,relatime,size=0k,mode=000 0 0
/dev/block/dm-0 /mnt/asec/com.collabora.libreoffice-2 vfat \
  ro,dirsync,nosuid,nodev,relatime,uid=1000,fmask=0222,dmask=0222,\
  codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
shell@android:/ $ _
```

Note the similarities and the differences with Linux filesystems; it looks a lot like standard Unix, anyway.

What you can actually do with the file system layout is *much* less than you would with Linux. On standard, unrooted devices, you can only access the internal storage, usually mounted on `/mnt/sdcard`, and the external storage (if available), usually mounted on `/mnt/external_sd`. The actual names of these directories may vary; e.g. `/storage/sdcard0` or something else. You'll probably have to figure it out for each device.

User data, programs included, are stored in `/data`. Using the Android shell you can `cd` to that directory, but you can't list its contents if you lack root permissions. Programs are usually stored in `/data/app`, but others (typically, paid programs and programs moved to the external storage) under `/mnt/asec`.

A very important directory under `/data` is accessible and writeable: `/data/local/tmp`. We'll make use of this directory later.

Note that internal and external storage is mounted as VFAT, the beloved file system that lacks file permissions and many more features. We will see in the following how to circumvent VFAT limitations.

2.3 Main directories in the internal storage

Assuming that `/mnt/sdcard` is the directory containing the internal storage, you should become familiar with the following sub-directories. Remember that `/mnt/sdcard` is mounted as VFAT; directory names are therefore case-insensitive.

- `Android/data/` contains directories with programs' user data. Each directory is named after the application, using a naming convention that will be explained in Section Installing/Uninstalling Programs from Linux. For example, Firefox settings are stored in `org.mozilla.firefox/`.
- `Download/` may contain files downloaded by browsers, unless they're configured otherwise.
- `DCIM/` contains one or more directories, each of which contains photographs stored as `.jpg` files. In my phone, `DCIM/` contains two directories, `Camera` and `OpenCamera`. Each directory contains pictures taken with a specific camera program: the default (Camera) and an additional program I installed (OpenCamera).
- `LOST.DIR/` may contain files that were recovered at boot time, when Android performs a file system check.
- `Notifications/` contains sound files for notifications (SMS, etc.). You can add your favourite sound files there.
- `Ringtones/`, similarly, contains sound files used as ring tones. You can add your favourite sound files there.
- `System/` may contain system data; in my phone, contacts backups are saved in `System/PIM/`.
- `Bluetooth/` contains file transfered via Bluetooth.

In addition, some applications make their data directory under `/mnt/sdcard` directly; for example, `/mnt/sdcard/Foo123`.

3. Applications

3.1 Alternative markets

In case you don't know: Android applications are distributed as `.apk` (Android application package) files, which are simple zip-compressed archives. Apk is the standard Android package format, and it serves the same purpose as `.rpm` or `.deb` archives in Linux distributions. As you probably know, programs are not downloaded as `.apk`s from Google Play; they're pushed to your device.

If you don't want to use the Google Play repository, or if you can't, there are alternative markets that provide Android FOSS:

- F-Droid,

  https://f-droid.org/

  lets you download programs `.apk`s directly, or via its client called `FDroid.apk`. I strongly suggest that you install the latter, that (from the F-Droid page) ``makes it easy to browse, install, and keep track of updates on your device."
- AOpenSource.com,

  http://www.aopensource.com

  is another repository dedicated to FOSS. In many cases, it simply redirects to Google Play; in other cases, to the application's home page.

In fact, many Android programs have their own web page, from which you can download the `.apk` and, in some cases, the program sources, documentation and so on.

3.2 Useful tools

Advanced Android users will want to install a file manager and a decent keyboard. Several FOSS programs are available, and my suggestions are:

- Hacker's Keyboard:

  http://code.google.com/p/hackerskeyboard/

  is a ``real" keyboard that provides arrow keys, Esc, Alt, Ctrl, and so on; you really can't do without it if you plan to use a terminal emulator. Released under the Apache License 2.0.
- Ghost Commander:

  https://sites.google.com/site/ghostcommander1

  is a very powerful, easy to use and complete dual-pane file manager, released under the GPL3. In addition to the usual features, it also provides plugins for accessing files via SMB, SFTP, GoogleDrive, DropBox, and BOX.

There are many other free file managers, but I feel that Ghost Commander is the most complete. Of course, feel free to install another if you prefer.

In addition to the tools above, you may want to turn your Andbox into something similar to a real Linux machine. You really want a terminal emulator and BusyBox, http://www.busybox.net/. The latter is a single executable that provides the functionality of several commands that you normally expect to find in any self-respecting Unix box, but that are missing in Android. `cp` and `tar` are the first that spring to mind.

In Section Terminal Emulators and Shells some terminal emulators will be briefly described; they also include BusyBox. The terminal emulator will let you run countless command-line programs that you can port to Android yourself!

3.3 Installing/uninstalling programs from Linux

You may find it convenient to download and store `.apks` in your Linux machine, possibly to install them on several Andboxes.

Let's suppose you downloaded an application, `foo123.apk`, and want to install it in your Andboxes. The simplest solution is to use the `adb install` command:

```
Linux:~$ ls *apk
884K foo123.apk
Linux:~$ adb install foo123.apk
3658 KB/s (898144 bytes in 0.239s)
        pkg: /data/local/tmp/foo123.apk
Success
Linux:~$ _
```

You might be surprised to find out that an application's real name *does not correspond* to the name of its `apk`! For example, the real name of `foo123` could be something similar to `com.android.foo123`. This naming method is based on Java package conventions, which are described at this page:

http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html

You must know a program's real name if you want to uninstall it using `adb`. To find out a program's real name, you can use Ghost Commander. Select ``Home'', ``Applications'', and carefully read the list of installed `.apks`; under each entry, the program's real name is printed in smaller characters.

Now you have all the necessary information to uninstall `foo123` using `adb uninstall`:

```
Linux:~$ adb uninstall foo123
Failure
Linux:~$ adb uninstall com.android.foo123
Success
Linux:~$ _
```

3.4 Installing APKs via another Andbox

An Andbox can easily share its installed programs with another device, provided that the Android versions are compatible. All you need is Ghost Commander. In the next paragraph, you can see how to send a file to Alice using Bob's Andbox.

From the ``Home'' position, select ``Applications'' to get a list of installed `.apks` ; click on the one you want and copy it to a standard folder (say, `/mnt/sdcard`.) From this location, long press on the program `.apk` and select ``Send To...''. Now select Bluetooth, which is probably the easiest way to send the APK to Alice; or choose another method from Section Copying Files below. If your device refuses to send a `.apk` file via Bluetooth, just rename or zip it before sending it, then unpack it or rename it back on the new device.

From within the Ghost Commander, tap on the `.apk` file to install the program. When done, you can delete the `.apk`.

4. Copying files

Exchanging files between Linux and Android should be trivial, right? Well, it's not. There are several ways to copy files from/to your Andbox, using a cable or Wi-Fi.

4.1 Copying via USB cable (from Linux box)

When you connect your device via USB cable, Linux may see it either as an external USB drive or as an *MTP device*, i.e. a multimedia player. In the first case, `udev` mounts the device somewhere, like `/media/guido/DEVICE`. Exchanging files is now trivial:

```
Linux~$ rsync -av --delete -i ~/Documents/MyStuff/ /media/guido/DEVICE/MyStuff/
```

If the Andbox is connected via MTP, more trickery might be required. Some Linux versions will mount the device using `gfsd-fuse`; for instance, my phone's internal storage is accessible at this directory:

`/run/user/1000/gvfs/mtp:host=%5Busb%3A001%2C111%5D/Internal Storage`

You may want to make a directory like the above more accessible creating a symbolic link, but unfortunately this directory is only temporary. In fact, the next time you plug your Andbox, the string `mtp:host` is bound to change.

```
Linux:~$ ln -s /run/user/1000/gvfs/mtp\:host\=%5Busb%3A001%2C111%5D/Internal\ Storage/ Pho
Linux:~$ cd Phone
Linux:~/Phone$ ls -l
totale 481K
32K drwx------ 1 guido guido 32K lug 29 13:09 Alarms/
32K drwx------ 1 guido guido 32K mar  2  2012 Android/
  0 drwx------ 1 guido guido   0 ago 26 11:59 Backups/
32K drwx------ 1 guido guido 32K mag 20  2014 baidu/
  0 drwx------ 1 guido guido   0 ott  6 19:57 bluetooth/
32K drwx------ 1 guido guido 32K mar 23  2015 CallRecordings/
512 -rw------- 1 guido guido 145 lug 29 12:46 customized-capability.xml
32K drwx------ 1 guido guido 32K lug 29 13:10 DCIM/
  0 drwx------ 1 guido guido   0 ott  9 12:40 Download/
  0 drwx------ 1 guido guido   0 dic 31  1999 LOST.DIR/
32K drwx------ 1 guido guido 32K lug 29 14:33 MIUI/
  0 drwx------ 1 guido guido   0 lug 31 10:00 Music/
  0 drwx------ 1 guido guido   0 ago  3 11:03 Notifications/
  0 drwx------ 1 guido guido   0 set 22 14:54 Pictures/
32K drwx------ 1 guido guido 32K mar 12  2014 Ringtones/
32K drwx------ 1 guido guido 32K gen  7  2013 svox/
32K drwx------ 1 guido guido 32K gen  9  2013 System/
  0 drwx------ 1 guido guido   0 ott 15 14:40 tmp/
Linux:~/Phone$ _
```

You will immediately notice that accessing the Andbox file system is quite slow.

In other cases, nothing happens. To access files on the device, you'll have to use programs that deal with MTP:

- Libmtp, http://libmtp.sourceforge.net/

  Provides MTP Tools, i.e. command line utilities to manage files, albums, playlists etc. on the device.
- Gmtp, http://gmtp.sourceforge.net/

  A graphical program to perform the same actions as MTP Tools.

In my personal experience, copying files with MTP has proven to be unreliable and error-prone. Copying files is also possible via ADB commands, which appear to be quite reliable. The following two commands copy (push) a file from Linux to the Andbox, and the other way around (pull):

```
Linux:~$ adb push file.txt /mnt/sdcard/directory/
Linux:~$ adb pull /mnt/sdcard/directory/file.txt
```

In the latter example, please note you must not add a dot at the end of the command: it's not the same as the Linux command `cp /mnt/sdcard/file.txt .`

4.2 Syncing directories (from Linux box)

If you want to keep a directory synchronised between the Linux box and the Andbox, in theory the command `adb sync` should suffice. In practice, I never managed to make it work.

Fortunately, there's a nice tool called `adb-sync`. It's written in Python and it's released under the Apache License at this address:

https://github.com/google/adb-sync

get it by cloning the GIT repository, then copy the command to a directory included in your `$PATH`:

```
Linux:~$ git clone https://github.com/google/adb-sync
Linux:~$ mv adb-sync/adb-sync ~/bin/
Linux:~$ _
```

`adb-sync` works in a similar manner as standard `rsync`:

```
Linux:~$ adb-sync --delete SYNC-ME/ /mnt/sdcard/SYNC-ME/
Sync: local SYNC-ME, remote /mnt/sdcard/SYNC-ME/
Scanning and diffing...
Warning: could not parse 'd---rwxr-x    2 system   sdcard_r \
32768 Oct 19 10:54 /mnt/sdcard/SYNC-ME/'.
Push: /mnt/sdcard/SYNC-ME/
Push: /mnt/sdcard/SYNC-ME//file1.txt
Push: /mnt/sdcard/SYNC-ME//file2.txt
Push: /mnt/sdcard/SYNC-ME//file3.txt
Total: 0 KB/s (0 bytes in 0.640s)
Linux:~$ _
```

4.3 Sorting files: `fatsort` (from Linux box)

Let's suppose you copied a bunch of music files to your device. You run your music player and open the directory containing the files. Surprise: they're not sorted, and are displayed in (apparently) random order!

It's the way the VFAT filesystem works, on Andboxes and MP3 players too. Enter `fatsort`, a great tool available here:

http://fatsort.sourceforge.net/

It's a command line program for Linux that sorts VFAT file systems on connected devices.

Plug the USB cable and mount the device. As root, run the `fdisk` tool:

```
Linux:~# fdisk -l
...
Disk /dev/sdd: 15.9 GB, 15925772288 bytes
1 heads, 32 sectors/track, 972032 cylinders, total 31105024 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1              32    30777343    15388656    c  W95 FAT32 (LBA)
```

In the above example, `/dev/sdd1` is the Android VFAT file system. To sort the files therein, check the consistency of the file system and run `fatsort`:

```
Linux:~# fsck.vfat -a /dev/sdb1
dosfsck 3.0.16, 01 Mar 2013, FAT32, LFN
0x41: Dirty bit is set. Fs was not properly unmounted and some data may be corrupt.
 Automaticaly removing dirty bit.
Performing changes.
/dev/sdb1: 8060 files, 856099/1921702 clusters
Linux:~# fatsort -f /dev/sdb1
FATSort Utility 0.9.15 by Boris Leidner <fatsort(at)formenos.de>

File system: FAT32.

Sorting directory /
Sorting directory /CTR/
Sorting directory /Foto/
Sorting directory /Guido/
...
Linux:~# _
```

In theory, `fatsort` should be run on unmounted file systems, but the above works and is reasonably safe.

4.4 Copying files from an Andbox FTP server

This is the most traditional way to upload and download files to/from a server. Among the several available, I picked up Swiftp FTP Server:

http://ppareit.github.io/swiftp/

also available on F-Droid.

This application is released under the GPL and runs on the non-standard port 2121, so it can run on unrooted Andboxes. (21 is a privileged port.) In the Login settings, set user and password for uploads and downloads, or anonymous login for download only. In the latter case, username and password are ``ftp'', ``guest''. Take note of your Andbox's IP adress and, from the Linux box, start an ftp session. The default remote directory is `/mnt/sdcard`:

```
Linux:~$ ftp 157.27.180.18 2121
Connected to 157.27.180.18.
220 SwiFTP 2.10.2 ready
Name (157.27.180.18:guido): ftp
331 Send password
Password:
230 Access granted
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT OK
150 Opening ASCII mode data connection for file list
drwxr-xr-x 1 owner group           40 Oct 23 08:23 .android_secure
drwxr-xr-x 1 owner group        32768 Sep 16 16:09 .MySecurityData
drwxr-xr-x 1 owner group        32768 Jul 29 13:09 Alarms
drwxr-xr-x 1 owner group        32768 Mar 02  2012 Android
...
drwxr-xr-x 1 owner group        32768 Oct 19 10:55 tmp
-rw-r--r-- 1 owner group          145 Jul 29 12:46 customized-capability.xml
226 Data transmission OK
ftp> put foo.dat
local: foo.dat remote: foo.dat
200 PORT OK
150 Data socket ready
226 Transmission complete
ftp> bye
ftp> 221 Goodbye
Linux:~$ _
```

4.5 Copying files from an Andbox web server

Another way of getting files from your Andbox is making it a Web server. Among the several free http servers available, I suggest that you install ServDroid that is quite straightforward to use:

https://github.com/joanpuigsanz/servdroid/wiki.

Configuring ServDroid is very simple. Copy the files you want to make available to

```
/mnt/sdcard/servdroid/var/www/
```

then start ServDroid and point a browser to port 8080 of your Andbox's IP.

5. Backup

This section may look pointless, as there are many backup & restore programs (I regularly use one). But there are alternatives that may come in handy.

5.1 Backing up applications

Installed applications are stored in `/data/apps/`, which is inaccessible on unrooted devices. You can, however, backup your `.apk` files using Ghost Commander or the `adb backup` command. I prefer the former, which is way faster and more convenient (IMHO).

Start Ghost Commander and select a directory on the internal storage; for example, `/mnt/sdcard/Tmp`. Move to the other pane, select ``Home", then ``Applications". Select the applications you want to backup, the select ``Copy". Please note that all applications will be listed, including system applications.

To copy the selected `.apk` to your Linux box, all you have to do is:

```
Linux~/backups/apk$ adb pull /mnt/sdcard/Tmp/
```

To restore, say, `foo.apk`, use this command:

```
Linux~/backups/apk$ adb push foo.apk /mnt/sdcard/Tmp/
```

Then open Ghost Commander, move to `/mnt/sdcard/Tmp/`, and tap on `foo.apk` to install it.

5.2 Backing up applications data

As explained in Section Main Directories in the Internal Storage, applications data is stored in directories under `/mnt/sdcard/Android/data`. You already have all the information you need to back up these directories!

6. Terminal emulator and shells

If you're a geek, you'll want to use a command line interface on your Andbox. Properly configured, a terminal emulator will give you almost the same interface as the Linux terminal. This is especially useful if your Andbox has a physical keyboard.

If you're not a geek, you should install a terminal emulator nonetheless: it will be useful anyway, since you'll be able to use some important command-line commands like Busybox. More about this later.

One of the best FOSS options is Android Terminal Emulator, available here:

https://github.com/jackpal/Android-Terminal-Emulator/wiki

It provides a fully functional terminal emulator, which is invaluable for at least two reasons. First and foremost, you will be able to run most command-line programs available on Linux; in many cases, you can even compile them yourself. Secondly, the terminal makes the standard Android commands found in `/system/bin` and `/system/xbin` available to you. But *beware*: these are not the same as the standard Linux commands! They're only a subset of the standard GNU commands (even `cp` is missing: use `cat file > newfile` instead), and they usually don't behave exactly the same.

If you need a more complete command-line experience, you will also want to install a command-line environment like KBOX, ZShaolin, Terminal IDE, or Termux:

- KBOX3 (KBOX2 is an older version):

  http://kevinboone.net/kbox3.html

KBOX is a compact and fully functional Linux-like command-line environment. It also provides additional software, such as `gcc`, `tmux`, the `dropbear` SSH server, and much more.
- Zshaolin:

  http://www.dyne.org/software/zshaolin/

  Zshaolin is a `zsh`-based terminal that provides lots (I mean *lots*) of additional free software. The whole thing is FOSS, but is not cost-free.
- Terminal IDE:

  http://www.spartacusrex.com/terminalide.htm

  It's a very complete Linux-like command-line environment. It's focused on development, providing Vim, `gcc`, `mc`, `ssh`, and loads of software. Once installed, it gives you the option ``Install System'' to install additional components, including the `bash` shell.

All of them are great programs: advanced users really can't do without a terminal.

Which one to install depends on how much sdcard space you can afford. If you have a few hundreds of MBytes to devote to it, I would suggest that you install Terminal IDE: it provides a very Linux-like experience. If you're short on storage, I suggest that you install KBOX: it's a very good compromise.

Users of Android 5 and later versions may want to install an alternative terminal emulator called Termux:

https://termux.com/

It's very nice and self-contained, i.e. it does not need KBOX or another command-line environment. Additional software can be installed simply using apt (Debian users, does that ring a bell?):

```
$ gcc
The program 'gcc' is not installed. Install it by executing:
  apt install gcc
$ ...
```

Termux is released under GPLv3, but some add-ons are available for a small fee.

6.1 Installing KBOX

Whether to install KBOX2 or KBOX3 depends on your Andbox' Android version. I installed KBOX2 on two of my Andboxes, which run Android 4.*, and KBOX3 on my new phone. In the following, I'll show how to install KBOX2; installing KBOX3 is very similar. Try KBOX3 first; if you get a segmentation fault running the KBOX3 installer, then try with KBOX2.

Download the base installer, called `kbox2-base-installer` or `kbox3-install-base`. Open Terminal Emulator and issue the following commands:

```
foo@android:/ $ cd
foo@android:/.../app_HOME $ cat /sdcard/Download/kbox2-base-installer > kbox2-base-install
foo@android:/.../app_HOME $ chmod 755 kbox2-base-installer
foo@android:/.../app_HOME $ ./kbox2-base-installer
UnZipSFX 6.00 of 20 April 2009, by Info-ZIP (http://www.info-zip.org).
  inflating: setup
  creating: installer-image/
 inflating: installer-image/install.sh
 inflating: installer-image/base-image.tar
 inflating: installer-image/busybox
 inflating: installer-image/.install.sh.swp
foo@android:/.../app_HOME $ ./setup
Running installer for KBOX2
Running installation script using busybox
CWD is /data/data/jackpal.androidterm/app_HOME
KBOX root directory is /data/data/jackpal.androidterm/app_HOME/kbox2
KBOX shell is /data/data/jackpal.androidterm/app_HOME/kbox2/bin/kbox_shell
```

If installation succeeds, you can replace the standard Android shell with KBOX's. Open the Terminal Emulator preferences, select ``Command line'', and change:

`/system/bin/sh -`

to:

`/data/data/jackpal.androidterm/app_HOME/kbox2/bin/kbox_shell`

Be very careful to write the above correctly! If you make a mistake, Terminal Emulator will not start and you will have to remove it and reinstall it.

Exit Terminal Emulator and reopen it. The command prompt should have changed to `kbox$`:

```
kbox$ pwd
/home/kbox
kbox$ _
```

Now you're working in a fake UNIX environment; many commands are provided by Busybox. If you want to access the Android file system, you'll find it under the `/android_root` directory.

You may want to install additional software from KBOX home page. Depending on your KBOX version, either link:

http://kevinboone.net/kbox2_downloads.html

http://kevinboone.net/kbox3_downloads.html

Additional programs are provided as `.deb` packages. This is the very same format used by Debian and many other Linux distributions, but obviously the packages provided for KBOX only work on KBOX. Download the package you want and install it as in the following example. Let's install coreutils, an improved implementation that replaces the default provided by KBOX's BusyBox:

```
kbox$ dpkg -i /android_root/mnt/sdcard/Download/coreutils_8.22_kbox.deb
Unpacking coreutils (from /android_root/mnt/sdcard/Download/coreutils_8.22_kbox.deb)...
Setting up coreutils (8.22)...
kbox$ ls --color /android_root
...
```

Printing via FTP

If you install the KBOX command-line ftp client, you *could* be able to print PDF files from your Andbox. I don't mean Google Cloud Print; and I repeat you *could*, because it actually depends on your printer's capabilities.

Some network printers include an FTP server, and you can print PostScript or PDF files sending them directly to the printer via FTP. First of all, find out whether your printer supports FTP:

```
Linux:~$ nmap MY.OFFICE.PRINTER

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-22 11:08 CEST
Nmap scan report for MY.OFFICE.PRINTER (10.15.130.105)
Host is up (0.00023s latency).
rDNS record for 10.15.130.105: csf-p1-v7913500307.xxx.yyy.zzz
Not shown: 990 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
23/tcp   open  telnet
80/tcp   open  http
139/tcp  open  netbios-ssn
514/tcp  open  shell
515/tcp  open  printer
631/tcp  open  ipp
7443/tcp open  oracleas-https
8080/tcp open  http-proxy
9100/tcp open  jetdirect

Nmap done: 1 IP address (1 host up) scanned in 12.27 seconds
Linux:~$ _
```

As you can see, the above printer (it's an Aficio MP 4001) has an open FTP port, in addition to the standard JetDirect port 9100 and many others (that should be closed, but still...)

From KBOX, run the ftp client and copy a file to the printer as in the following session. The user name is ``ftp'', the password is ``guest''. These are common FTP defaults.

```
kbox$ ftp MY.OFFICE.PRINTER
Connected to csf-p1-v7913500307.MY.OFFICE.DOMAIN
220 NRG MP 4001 FTP server (7.34) ready.
Name (MY.OFFICE.PRINTER:guido): ftp
331 Password required for ftp.
Password:
230 User ftp logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put my_file.pdf
local: my_file.pdf remote: my_file.pdf
200 PORT command successful.
150 Opening BINARY mode data connection for 'my_file.pdf'.
226 Transfer complete.
24057 bytes sent in 0.00 secs (405054.5 kB/s)
ftp> bye
221 Goodbye.
kbox$ _
```

When the file is transfered to the printer, it will be printed.

I successfuly tested this method on several HP and Aficio network printers. Obviously, you mileage may vary; for instance, you may need to provide a real user name and password instead of ``ftp", ``guest". Ask your local sysadmin (and buy him/her a beer while you're at it).

Ghost Commander provides access to ftp servers natively, so it can be used instead of the command-line ftp client.

6.2 Remote shell via SSH

In addition to local shells, you can open a shell on your device remotely, via `ssh`; you will need a `ssh` server on the Andbox. Such server is SSHelper (free software, GPL'ed):

http://arachnoid.com/android/SSHelper/

Since it runs on unrooted devices, it uses the port 2222 instead of the standard port 22; i.e., to connect to it from the Linux terminal you will do:

```
Linux:~$ ssh -p 2222 157.27.188.78
The authenticity of host '[xx.xx.xx.xx]:2222 ([xx.xx.xx.xx]:2222)' can't be established.
ECDSA key fingerprint is 20:3f:f1:c1:3b:ce:fb:61:3e:a3:bb:0d:a4:15:54:c7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[xx.xx.xx.xx]:2222' (ECDSA) to the list of known hosts.
SSHelper Version 7.8 Copyright 2014, P. Lutus
Default password is "admin" (recommend: change it)
guido@157.27.188.78's password:
Android:2.6.39.4-g9560a99
app_105@android:/data/data/com.arachnoid.sshelper/home $ _
```

The connection uses the standard Android shell, `/system/bin/sh`. Nothing prevents you from starting, say, KBOX. All you have to do is type the full path of the KBOX shell:

```
...home $ /data/data/jackpal.androidterm/app_HOME/kbox2/bin/kbox_shell
kbox$ _
```

6.3 Compiling C programs, Linux side

Once you have a terminal emulator, you can add a lot of command-line based software to your Android box. If you're a geek, this is an invaluable extension that makes your Andbox closer to a ``real" computer; but even if you're a casual user, please read on.

You can compile C programs directly on your Andbox, and also on your Linux box. In the latter case, you use a so-called *cross compiler toolchain*: a compiler and associated binutils that produce code for Android. In many cases, but not only, for the ARM cpu.

Most Linux distributions provide one or more cross compiler toolchains that target ARM on Android. On my box, I installed the package `gcc-arm-linux-androideabi` and its dependencies. If your Linux distributions doesn't include it by default, get the relevant toolchain from the NDK home page:

http://developer.android.com/ndk/downloads/index.html

Another toolchain that targets Android on x86 CPUs is called `gcc-i686-linux-android`.

Choose the relevant architecture and download the binary package for Linux 32-bit or 64-bit. The directory

```
toolchains/arm-linux-androideabi-4.9/prebuilt/linux-x86/bin/
```

contains prebuilt binaries: include that directory in the `$PATH`. MIPS and x86 binaries are also available.

Supposing you're compiling for ARM, the compiler is called `arm-linux-androideabi-gcc`. You use it instead of `gcc` to compile Android binaries. In general, all binutils have a `arm-linux-androideabi-` prefix.

```
Linux:~$ arm-linux-androideabi-gcc -static -o hello hello.c
Linux:~$ arm-linux-androideabi-strip hello
Linux:~$ file hello
hello: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (SYSV), statically linked, strippe
Linux:~$ _
```

The `-static` option makes the executable self-contained. It may be necessary to avoid problems with shared libraries, but it also makes the executable larger.

Now you can copy the program to your Android box, but you can't copy it to any directory of your choice. In fact, if you copy a program to a directory in the internal memory, like `/mnt/sdcard`, you'll be unable to run the program as it lacks the 755 permissions. Remember, `/mnt/sdcard` is mounted as VFAT, so it cannot provide file permissions.

Fortunately, the Android file system has a directory that is writeable and supports file permissions: `/data/local/tmp`. Unlike the `/tmp` directory in Linux, files in this directory are not deleted at boot time.

Here is how to copy (all Android versions) and run (Android older than 5.*) your application on the Andbox:

```
Linux:~$ adb push hello /data/local/tmp
57 KB/s (2752 bytes in 0.046s)
geo:~$ adb shell
shell@android:/ $ cd /data/local/tmp
shell@android:/data/local/tmp $ chmod 755 hello
shell@android:/data/local/tmp $ ./hello
Hello, world!
32|shell@android:/data/local/tmp $ exit
Linux:~$ _
```

Beware: Android 5.* has better security policies and will not allow running programs from `/data/local/tmp`, or even accessing that directory. From the terminal, you'll have to copy the compiled programs to a directory where you have write and execute permissions. Beware: you will have to copy files in the blind, specifying their exact names, no wildcards allowed!

For instance, using KBOX:

```
kbox$ ls /android_root/data/local/tmp
ls: cannot open directory  /android_root/data/local/tmp: permission denied
kbox$ cp /android_root/data/local/tmp/hello .
kbox$ ls
hello
kbox$ chmod 755 hello
kbox$ ./hello
hello, world!
kbox$ _
```

A remarkable list of command-line programs can be easily compiled for Android. Among others, I ported the Bywater BASIC interpreter:

http://sourceforge.net/projects/bwbasic/

(minor tweaks were needed: I added `-DHAVE_ISNAN=1` to the DEFS in the `Makefile`, and also `#define uint64_t unsigned long` at the top of `bwbasic.h`)

The C version of the e3 text editor (version 2.7.1) can be compiled with no tweaks at all:

https://sites.google.com/site/e3editor/

same for the ABC music tools:

http://abcplus.sourceforge.net/

In general, any program that doesn't depend on fancy libraries is readily portable to Android. Explanations on compiling stuff in Android are available here:

http://kevinboone.net/android_native.html.

Compile the programs you need, copy them to `/data/local/tmp/`, and make sure this directory is included in the `$PATH` of your favourite terminal.

6.4 Compiling C programs, Android side

The simplest way to compile programs on an Android box is installing Terminal IDE, cited above, and its ``System'' that includes the `gcc` compiler. It's included, but as a compressed archive: it's up to the user to unpack it and install it. Let's see how to do it.

Start Terminal IDE, enter the terminal and insert these commands:

```
        cd system
        tar zxvf android-gcc-4.4.0.tar.gz
        cd
        vi .bashrc
```

Surely you can use `vi`, can't you? Move to the end of the file and add this line:

```
export PATH=$HOME/system/android-gcc-4.4.0/arm-eabi/bin:$PATH
```

save, exit `vi`, exit Terminal IDE then restart it. You have expanded the PATH to include the directory containing the `gcc` executable. Now you should be able to run it:

```
        gcc
        gcc: no input files
```

That's it! From now on, you can compile programs directly on your Android box. Don't expect to be able to compile every single, though.

6.5 CCTools

As the CCTools home page states, ``CCTools is native IDE for Android devices. It includes C/C++/Lua/GNU Makefile/Shell source code editor with syntax highlighting and complete android gcc toolchain for arm/mips/x86 devices."

CCTools is a very nice addition, and it also provides support for Fortran, SDL and much more. I suggest that you give it a try.

7. Remote control

An Andbox can be turned into a remote that works via WiFi, Bluetooth, or USB, to control a PC or other devices. Conversely, a PC can be used to remotely control your Andbox.

7.1 Web interface to android

There are several free-as-in-free-beer programs that provide a Web interface to Wi-Fi connected Andbox: AirDroid, 3CX DroidDesktop, RemoteDesktop, and many others. These programs let you access your device and manage files, SMS, multimedia, and more. Great programs, really: too bad they're not FOSS.

As of this writing, the only FOSS program that provides the same functionality (and much more) is Webkey, released under the GPL:

https://code.google.com/p/webkey-dev/downloads/list

It lets you fully control your device via WiFi or 3/4G using a browser. Unfortunately, it requires root: but it's such an incredibly great program, you may want to root your device just to use it.

Enable Wi-Fi, start Webkey and take note of the https address. Open a browser on the Linux box and go the https address to be greeted by Webkey's login page. Click on ``Registration", insert a username and password and click on ``Create new user". Webkey will ask for your permission to allow the user to control the phone. Using the newly created credentials, you can now log in.

Now, put the Andbox down and navigate through available options. Everything you can do on the Andbox can be done in Webkey's page: it provides a terminal, file manager, screenshot, program control, and *much* more!

7.2 Android as a remote

Many commercial and/or free-as-in-free-beer applications exist; just search the web to find them. FOSS applications worth mentioning are:

- Gfx Tablet (MIT License): it turns the Andbox into a Wi-Fi graphics tablet, useful for drawing in applications like GIMP. It needs a sudo'ed Linux-side server (networktablet).

  Home: http://rfc2822.github.io/GfxTablet/
- RemoteDroid: WiFi only, GPL'ed. It also provides a handy keyboard that includes T9.

  Home: https://code.google.com/p/remotedroid/

- PRemoteDroid: WiFi and Bluetooth (libbluetooth-dev required). Released under generic Open Source license.

  Home: https://code.google.com/p/premotedroid/
- Gmote: WiFi, it's meant to be a multimedia controller. Released under the Apache license.

  Home: https://code.google.com/p/gmote/
- DroidPad (GPL3): USB and WiFi. To make it work as a normal user, issue the command `sudo chmod 666 /dev/uinput`.

  Home: http://www.digitalsquid.co.uk/droidpad/.
- Anyremote (GPL2): Bluetooth, Wi-Fi or just TCP/IP connection.

  Home: http://anyremote.sourceforge.net/

They all work in a similar manner. First off, you install a ``server'' program on the Linux box, then use the Android application that talks to it. Let's see, for instance, how to install RemoteDroid.

Copy the server `RemoteDroidServer.jar` somewhere to your Linux box, then run it:

```
Linux:~$ java -jar RemoteDroidServer.jar
```

A window will pop up, showing some information and the IP address you can connect to. Activate WiFi on the Andbox (3G/4G/xG is not supported), launch RemoteDroid, and provide the IP address you were shown. There you are: slide your finger on the Android screen and see the arrow move on the Linux box.

7.3 X servers

If you're a geek, you know that you can run a program on a remote computer and display its graphical user interface on another computer. This useful feature is provided by the X server; most Linux distributions use X.Org.

At least two Android X server implementations are available:

- android-xserver: http://code.google.com/p/android-xserver/
- XSDL, http://sourceforge.net/projects/libsdl-android/files/apk/XServer-XSDL/

I encountered a few issues using `android-xserver`, while XSDL gave me no problems. XSDL is pretty self-explanatory: when you first start it, it asks for your permission to download and install additional fonts. Then, it gives you instructions on how to start a window manager and an application on the remote Linux (or Unix) machine.

Start XSDL, then start Terminal IDE and run these commands:

```
terminal++@157.27.100.1:~$ ssh user@remote.linux.machine
user@remote.linux.machine's password:
Linux:~$ export DISPLAY=157.27.0.1:0 # IP of Andbox
Linux:~$ marco & # window manager
Linux:~$ gimp
```

At this stage, you'll have the Gimp running on the remote machine and displaying its user interface on the Andbox. Now you can enjoy the awkward experience of using a graphical application without a physical mouse!

7.4 VNC client

Using a VNC client you can control a remote graphical session from your Andbox. Several VNC clients exist; a GPL'ed one is androidVNC,

https://code.google.com/p/android-vnc-viewer/

also available from F-Droid.

You will also need a VNC server on the Linux remote machine. There are many VNC servers available; in the following example I'll use Tight VNC, GPL'ed,

http://www.tightvnc.com

The first command you must provide on the Linux box is `tightvncpasswd`, which lets you set an access password for VNC sessions (user name is not required). Once you've set your password, start the server:

```
Linux:~$ tightvncserver

New 'X' desktop is linux-machine:1

Creating default startup script /home/guido/.vnc/xstartup
Starting applications specified in /home/guido/.vnc/xstartup
Log file is /home/guido/.vnc/geo:1.log
```

The server is started, most probably on TCP port 5901 (check the log file). A new file, `$HOME/.vnc/xstartup`, has been created: you're free to modify it to customize your VNC session.

Now, run androidVNC and provide the connection information. Colour depth will influence your connection speed; the fewer colours, the faster the connection. Again, enjoy the awkward world of mouse-less user interface!

8. Taking screenshots and screencasts

Some Android versions let the user take a screenshot natively, usually holding the Power and Volume Down buttons together. For all others, there's DroidAtScreen:

http://droid-at-screen.ribomation.com/

This application is invaluable if you want to display your Android screen on a computer and/or a projector.

On the Andbox, all you have to do is enable USB debug. DroidAtScreen is a Java application that runs on the Linux box, where it needs `adb`. Extract the .jar file and run it:

```
Linux:~$ java -jar droidAtScreen-1.0.2.jar &
```

Then configure DroidAtScreen telling it where to find the adb executable; type in its full path. You may have to click on Restart ADB and/or Reload Devices.

9. Sharing GPS via Bluetooth

One of the most useful features of many Andboxs is the integrated GPS circuitry. GPS-enabled Android phones can share the location with other devices, including Android and Linux machines. All you need is Bluetooth and some software.

9.1 Sharing from Android to Android

Let's suppose that you have a GPS-enabled Android phone and a GPS-less Android tablet. On the phone, you can install a program that turns it into an NMEA GPS---that is, a GPS receiver that can ``talk'' to other devices using the NMEA protocol. On the tablet, you will install a program that connects to the NMEA GPS and reads the location info.

Bluetooth GPS for Android,

http://sourceforge.net/projects/bluegps4droid/

is the program you need to connect to the NMEA GPS. In our example, you will install it on the tablet.

On the phone, you have more choice. I tested the following NMEA programs, all of which worked well:

- GPS over BT:

  https://play.google.com/store/apps/details?id=com.Saenko.GpsOverBt

  It's the most complete of the bunch: provides a lot of information on the visible satellites, GPS position and so on. Free to use, unknown license.
- Share GPS:

  http://sharedroid.jillybunch.com/

  It's pretty simple to use; free to use, unknown licence.
- GPS 2 Bluetooth:

  http://android.cajax.net/

  This program is extremely simple to use: it's basically an on/off button to enable the external GPS. Free to use, unknown licence.
- BlueNMEA:

  http://max.kellermann.name/projects/blue-nmea/

  It's a tiny, very basic program. It's free to use, sources available, unknown licence.

Once you have the software installed on the devices, you're ready to go. First of all, enable GPS on the phone and get the fix. Then, enable bluetooth on both devices and pair them. Then start the NMEA app of your choice on the phone.

To get the GPS location on the tablet, start Bluetooth GPS for Android. The location will become available in a few seconds, and will be shown in applications. My favourite is Osmand, http://www.osmand.net.

10. The end, for now

Congratulations! You have now some basic information on how to turn your Andbox into something even more useful, and how to interact with it using your trusted GNU/Linux machine.

At the end of this document, I can confess you the truth: while I'm a long-term GNU/Linux fan, I'm afraid I've just started to appreciate Android. I find it insanely fragmented; until recently, it was unacceptably sluggish and laggy; I also believe it still suffers from (at least) a couple of serious design flaws. Besides, I don't like tablets, all of them; I find way too limited, if compared to a real computer. That said, I can say that my 8-core, Android 5.0.1-based phone is quite nice.

I hope this guide provided some useful information. It's only the beginning: please stay tuned for updates.

10.1 Copyleft

Enjoy,

Guido `=8-)`