IL SISTEMA OPERATIVO ANDROID

Guida alla creazione e modifica di android
nei nostri device

di Giacomo Giacalone alias morfes

Blog: http://pointnext.blogspot.com/

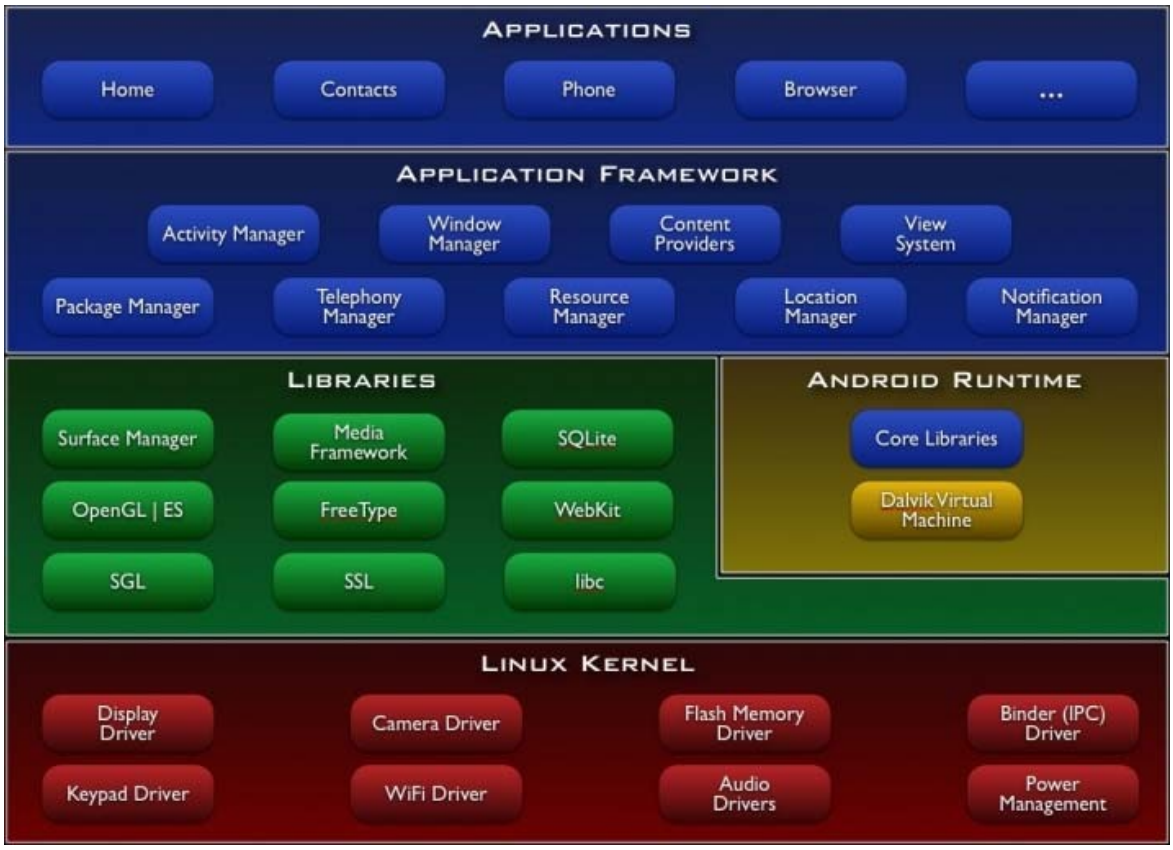
Donazione

INTRODUZIONE

Con questo manuale vi spiegherò il funzionamento del sistema operativo android, come compilarlo con i sorgenti forniti da google e come personalizzarlo nei nostri smartphone o table. Introdurrò come è fatto android, come modificarlo tramite alcuni scripts che troviamo in rete e con le varie cucine “kitchen” che ci aiutano per modificare le nostre rom moddate. Per rom moddate si intende modificare il system di android cioè il suo sistema personalizzando il framework.apk, rimuovendo e aggiungendo le app che troviamo nella cartella app e compilando i vari programmi come busybox che ci permettono di interagire con l'intero sistema come se fosse una distro unix/linux. Parlerò anche come fare dei eventuali porting con devices che hanno lo stesso hardware modificando semplicemente alcune librerie ed eseguibili e di come ricompilare il kernel per abilitare funzioni aggiuntive che possano migliorare il sistema come il supporto per la **swap**, i vari filesystem **ext**, **cifs**, **nfs** e **tun** per connessioni con openvpn. Questo manuale è una prima versione sintetica, successivamente potranno esserci altre versione più dettagliate.

COS'È ANDROID E COME È FATTO

Android è un sistema operativo sviluppato da Google per i dispositivi mobili, esso si basa da sistemi operativi open source linux anche se è stato modificato completamente, infatti il cuore di ogni sistema Android è un kernel Linux, versione 2.6. Direttamente nel kernel sono inseriti i driver "moduli" per il controllo dell'hardware dei dispositivi: driver per la tastiera, lo schermo, il touch screen, il Wi-Fi, il Bluetooth, il controllo dell'audio e così via. Sopra il kernel poggiano le librerie fondamentali, anche queste tutte mutuate dal mondo Open Source. Da citare sono senz'altro OpenGL, per la grafica, SQLite, per la gestione dei dati, e WebKit, per la visualizzazione delle pagine Web. L'architettura prevede poi una macchina virtuale ed una libreria fondamentale che, insieme, costituiscono la piattaforma di sviluppo per le applicazioni Android. Questa macchina virtuale si chiama Dalvik, ed è una Java Virtual Machine. In android vi sono inclusi applicativi destinati per l'utente finale che sono installati di base come browser, il player multimediale e molto altro. La sua architettura è rappresentato in questa figura:



Prima di iniziare con la personalizzazione di android, dobbiamo scaricare e installare sul proprio pc la **sdk android** gratuitamente da questo link: http://developer.android.com/sdk/. La sdk è un kit di sviluppo che serve sia per sviluppare app che per interagire con il nostro sistema android, esso contiene un emulatore, le librerie e i vari tools come adb, un tool molto importante che serve per lavorare su android con il nostro pc tramite una console di terminale. Ci servirà anche fastboot, un programma che si trova in rete e che permette di flashare le immagini del boot, della system e

recovery. Purtroppo la personalizzazione di android non è uguale per tutti i dispositivi, infatti vi sono device che hanno il bootloader sbloccato e altri no o che servono determinati programmi per caricare android. In un device che ha il bootloader sbloccato possiamo caricare il boot.img e la system.img android tramite fastboot o con un pacchetto zippato tramite una recovery modificata che permette di farlo. Per esempio uno smartphone Huawei per aggiornare android o per reinstallarlo dobbiamo flashare il suo firmware chiamato updata.app o update.app, per altri device invece si usano altri programmi che sono messi a disposizione dai vari produttori, vedi samsung kies per i device Samsung. Fortunatamente in rete ci sono le varie guide su come installare e aggiornare android in base ai vari dispositivi, la mia guida si baserà sui dispositivi Huawei, visto che io ho un Huawei Ideos U8150.

ADB E FASTBOOT

Adb "**Android Debug Bridge**" è uno strumento a riga di comando che permette di comunicare il device android con il nostro pc, infatti è un programma di client-server, dal quale grazie ad una shell possiamo vedere l'intero contenuto del filesystem del device. Per poter utilizzare adb in windows dobbiamo prima installare i driver del nostro device per farlo riconoscere quando lo colleghiamo al pc dalla porta usb, i driver di solito li troviamo direttamente nella rom android del device o li possiamo scaricare dal sito del produttore, invece per linux non serve nessun driver dobbiamo dare solo questi comandi da root:

```
giacomo@morfes:~/android_sdk/platform-tools$ sudo ./adb kill-server
giacomo@morfes:~/android_sdk/platform-tools$ sudo ./adb start-server
```

ecco l' help di adb:

```
./adb -h
Android Debug Bridge version 1.0.26


 -d                            - directs command to the only connected USB device
                                 returns an error if more than one USB device is present.
 -e                            - directs command to the only running emulator.
                                 returns an error if more than one emulator is running.
 -s <serial number>            - directs command to the USB device or emulator with
                                 the given serial number. Overrides ANDROID_SERIAL
                                 environment variable.
 -p <product name or path>     - simple product name like 'sooner', or
                                 a relative/absolute path to a product
                                 out directory like 'out/target/product/sooner'.
                                 If -p is not specified, the ANDROID_PRODUCT_OUT
                                 environment variable is used, which must
                                 be an absolute path.
 devices                       - list all connected devices
 connect <host>[:<port>]       - connect to a device via TCP/IP
                                 Port 5555 is used by default if no port number is
specified.
 disconnect [<host>[:<port>]]  - disconnect from a TCP/IP device.
```

```
                                  Port 5555 is used by default if no port number is
specified.
                                  Using this ocmmand with no additional arguments
                                  will disconnect from all connected TCP/IP devices.
device commands:
  adb push <local> <remote>    - copy file/dir to device
  adb pull <remote> [<local>]  - copy file/dir from device
  adb sync [ <directory> ]     - copy host->device only if changed
                                  (-l means list but don't copy)
                                  (see 'adb help all')
  adb shell                    - run remote shell interactively
  adb shell <command>          - run remote shell command
  adb emu <command>            - run emulator console command
  adb logcat [ <filter-spec> ] - View device log
  adb forward <local> <remote> - forward socket connections
                                  forward specs are one of:
                                    tcp:<port>
                                    localabstract:<unix domain socket name>
                                    localreserved:<unix domain socket name>
                                    localfilesystem:<unix domain socket name>
                                    dev:<character device name>
                                    jdwp:<process pid> (remote only)
  adb jdwp                     - list PIDs of processes hosting a JDWP transport
  adb install [-l] [-r] [-s] <file> - push this package file to the device and install it
                                  ('-l' means forward-lock the app)
                                  ('-r' means reinstall the app, keeping its data)
                                  ('-s' means install on SD card instead of internal
storage)
  adb uninstall [-k] <package> - remove this app package from the device
                                  ('-k' means keep the data and cache directories)
  adb bugreport                - return all information from the device
                                  that should be included in a bug report.
  adb help                     - show this help message
  adb version                  - show version num
DATAOPTS:
 (no option)                   - don't touch the data partition
  -w                           - wipe the data partition
  -d                           - flash the data partition
scripting:
  adb wait-for-device          - block until device is online
  adb start-server             - ensure that there is a server running
  adb kill-server              - kill the server if it is running
  adb get-state                - prints: offline | bootloader | device
  adb get-serialno             - prints: <serial-number>
  adb status-window            - continuously print device status for a specified device
```

```
  adb remount                   - remounts the /system partition on the device read-write
  adb reboot [bootloader|recovery] - reboots the device, optionally into the bootloader
or recovery program
  adb reboot-bootloader         - reboots the device into the bootloader
  adb root                      - restarts the adbd daemon with root permissions
  adb usb                       - restarts the adbd daemon listening on USB
  adb tcpip <port>              - restarts the adbd daemon listening on TCP on the
specified port
networking:
  adb ppp <tty> [parameters]    - Run PPP over USB.
 Note: you should not automatically start a PPP connection.
 <tty> refers to the tty for PPP stream. Eg. dev:/dev/omap_csmi_tty1
 [parameters] - Eg. defaultroute debug dump local notty usepeerdns
adb sync notes: adb sync [ <directory> ]
  <localdir> can be interpreted in several ways:
  - If <directory> is not specified, both /system and /data partitions will be updated.
  - If it is "system" or "data", only the corresponding partition
    is updated.


environmental variables:
  ADB_TRACE                     - Print debug information. A comma separated list of the
following values
                                1 or all, adb, sockets, packets, rwx, usb, sync, sysdeps,
transport, jdwp
  ANDROID_SERIAL                - The serial number to connect to. -s takes priority over
this if given.
  ANDROID_LOG_TAGS              - When used with the logcat option, only these debug tags
are printed.
```

come vedete dall'help con push possiamo copiare dei file dal nostro pc al device, invece con pull possiamo prelevare file o app dal device al pc, questa è un opzione molto importante che può servire nella modifica del sistema. Possiamo usare adb anche tramite wifi senza collegarlo al pc da usb, basta scaricare dal market l'applicazione adb wireless sul nostro android. Con adb shell abbiamo una shell remote dove possiamo esplorare il contenuto del nostro device.

FASTBOOT

Fastboot quando ho installato l'sdk di android per linux non era presente, quindi l'ho dovuto scaricare da internet, ma forse con le nuove versione di sdk ci dovrebbe essere, se non lo trovate, scaricatelo da internet. Fastboot è un programma utilizzato per aggiornare il filesystem della flash nei dispositivi Android via Usb, esso consente di flashare immagini di partizioni non firmati. Per alcuni devices non si possono flashare le partizioni perchè hanno il bootloader bloccato, quindi prima si deve vedere su internet in qualche blog o forum se si può sbloccare oppure dobbiamo flashare android con altri programmi vedi per esempio il programma odin e kies che serve per mettere versioni aggiornati di android in certi modelli smartphone Samsung. Per poter usare fastboot dovete mettere il vostro device in modalità **bootloader**, avviandolo tramite una serie di combinazione dei tasti volume e power, anche in questo caso varia dai diversi device, per avviarlo

in automatico potete installare dal market l'app **quick boot** ovviamente nel vostro android dovete avere i permessi di **root**, permessi che potete avere installando app come per esempio **z4root** o installando particolari programmi nel vostro pc. Dopo di che avviate fastboot su linux e flashate con questi comandi:

```
sudo ./fastboot flash boot boot.img
sudo ./fastboot flash system sytem.img
sudo ./fastboot reboot
```

come potete vedere questi comandi servono per flashare il vostro boot e la system della vostra rom modificata o compilata, nei prossimi capitoli vi spiegherò come flashare la vostra rom anche tramite la vostra recovery modificata con un pacchetto zippato che contiene il boot e la system, grazie all'aiuto di una kitchen come quella di dsixda. Qui sotto vi elenco l'help di fastboot:

```
./fastboot -h


usage: fastboot [ <option> ] <command>


commands:


  update <filename>                        reflash device from update.zip
  flashall                                 flash boot + recovery + system
  flash <partition> [ <filename> ]         write a file to a flash partition
  erase <partition>                        erase a flash partition
  getvar <variable>                        display a bootloader variable
  boot <kernel> [ <ramdisk> ]              download and boot kernel
  flash:raw boot <kernel> [ <ramdisk> ]    create bootimage and flash it
  devices                                  list all connected devices
  reboot                                   reboot device normally
  reboot-bootloader                        reboot device into bootloader


options:
  -w                                       erase userdata and cache
  -s <serial number>                       specify device serial number
  -p <product>                             specify product name
  -c <cmdline>                             override kernel commandline
  -i <vendor id>                           specify a custom USB vendor id
./fastboot -h
```

Se volete sapere come è partizionata la flash del vostro device dovete dare tramite adb o emulatore di terminale questo comando:

```
cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00500000 00020000 "boot"
mtd1: 00500000 00020000 "recovery"
mtd2: 00140000 00020000 "misc"
mtd3: 00060000 00020000 "splash"
mtd4: 0aa00000 00020000 "system"
mtd5: 05d00000 00020000 "cache"
mtd6: 0a6a0000 00020000 "userdata"
mtd7: 01400000 00020000 "cust"
```

questa tabella di partizione è quella del mio Ideos Huawei, con queste info potete estrarre il dump delle img tramite alcuni tools di linux come cat o dd sia il boot che la system, per esempio:

```
cat /dev/mtd/mtd0 > /sdcard/boot.img
cat /dev/mtd/mtd4 > /sdcard/system.img
```

KITCHEN DSIXDA

La cucina dsixda serve per automatizzare la procedura per modificare il boot e la system di una rom android, infatti grazie a questa cucina si possono creare diverse rom moddate, essa contiene una serie di scripts che si possono trovare anche in rete che permettono di scompattare e ricompattare il boot.img, estrarre il contenuto del system.img e quello dei firmware di alcuni produttori come quello dell'updata.app dei devices huawei. Per poter scaricare la kitchen e vedere le sue info visitate il forum di xda a questo indirizzo: http://forum.xda-developers.com/showthread.php?t=633246 . Io vi spiegherò alcune funzioni della kitchen, quelle che di solito vengono usate per creare una rom moddata, intanto vi dico che la cucina supporta molti firmware di vari produttori, molti devices htc, quelli huawei, lg, motorola ecc..., per un changelog completo leggete il post di dsixda. Parlerò di come usare la cucina in ambiente linux, per utilizzarla su windows seguite la guida di Lucky76 postata sul forum TuttoAndroid: http://www.tuttoandroid.net/forum/cucinare-custom-rom-android-con-kitchen-0-179-t9382.html . Prima di avviare la cucina ovviamente dovete avere installato la java jdk, il menu è in inglese ma molto intuitivo per coloro che non masticano bene la lingua, ecco l'elenco delle funzioni del programma:

```
  ================================================================
```

```
 Android Kitchen - by dsixda (xda-developers.com)
================================================================

Main Menu

 1. Set up working folder from ROM
 2. Add root permissions
 3. Add BusyBox
 4. Disable boot screen sounds
 5. Add wireless tethering
 6. Zipalign all *.apk files to optimize RAM usage
 7. Change wipe status of ROM
 8. Change name of ROM
 9. Check update-script for errors
10. Show working folder information

Advanced Options

 11 - Deodex files in your ROM
 12 - Add task killer tweak for speed (stock ROM only)
 13 - Add /data/app functionality
 14 - Add Nano text editor + sysro/sysrw
 15 - Add Bash (command shell)
 16 - Add Apps2SD
```

```
 17 - Add /etc/init.d scripts support (busybox run-parts)
 18 - Add custom boot animation functionality
 19 - Porting tools (experimental)
 20 - Tools for boot image (unpack/re-pack/etc.)
 21 - Unpack data.img
 22 - Sign APK or ZIP file(s)
 23 - Convert update-script or updater-script
 24 - Miscellaneous optins / Plugin scripts
99. Build ROM from working folder
00. About/Update kitchen
 0. Exit
```

L'opzione 1 serve per iniziare a lavorare con la rom, vi è un elenco dove dovete scegliere su quale rom dovete lavorare se avete messo dei firmware, rom zippate o immagini di boot e system, copiate dentro la cartella **original_update** della dsixda:

```
Ensure there is at least one ROM under the original_update folder.


The format for each ROM must be one of the following:


 - ZIP file for a custom ROM  (e.g. update.zip)
 - ZIP file containing *.img  (e.g. rom.zip from shipped ROM)
 - ZIP file containing shipped ROM in SYSTEM + BOOT folder format
 - APP file from Huawei       (e.g. UPDATE.APP from software update)
 - system.img + boot.img      (e.g. from shipped ROM or Nandroid)
 - Working folder made with this kitchen  (e.g. WORKING_old_rom)
Press Enter to continue
Available ROMs:


  (1) android-froyo-2.2.2.zip
  (2) updata8180.app
  (3) WORKING_android-ginger
  (4) WORKING_ideos2.2.2
  (5) WORKING_cyano8160


Enter selection number (default=1, cancel=0):
```

questo è un esempio della mia dsixda dove ho varie rom da modificare. Dopo aver selezionato la rom da modificare presa direttamente dal vostro device tramite un dump o dal firmware, iniziate ad aggiungere per prima cosa il root che è molto utile in una rom moddata per fare successivamente delle modifiche interne al suo filesystem, come per sempio creare scripts in */system/etc/init.d/*, aggiungere o eliminare app nella cartella */system/app*, poter installare applicazioni che richiedono il root, ecc... Infatti quando andiamo a modificare una rom la dsixda crea una cartella WORKING_ seguita da una serie di numeri, la kitchen ci chiederà se vogliamo modificare pure il nome della cartella dove andranno il boot.img e la cartella system e la meta-inf, in questa cartella ci saranno gli scripts che serviranno ad installare la rom zippata tramite recovery modificata. Un'altra cosa da abilitare sempre dal menu con la opzione 3 è la busybox, un programma per distro linux che contiene al suo interno vari programmi nativi per gestire

l'intero sistema android:

```
BusyBox v1.19.0.git.adrynalyne (2010-12-21 22:03:30 MST) multi-call binary.
Copyright (C) 1998-2009 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: busybox --list[-full]
   or: function [arguments]...
        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as.

Currently defined functions:
        [, [[, add-shell, addgroup, adduser, adjtimex, ar, arp, arping, ash,
        awk, basename, beep, blkid, blockdev, bootchartd, brctl, bunzip2,
        bzcat, bzip2, cal, cat, catv, chat, chattr, chgrp, chmod, chown,
        chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio,
        crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt,
        delgroup, deluser, depmod, devmem, df, dhcprelay, diff, dirname, dmesg,
        dnsd, dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, ed,
        egrep, eject, env, envdir, envuidgid, ether-wake, expand, expr,
        fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk,
        fgconsole, fgrep, find, findfs, flock, fold, free, freeramdisk, fsck,
        fsck.minix, fsync, ftpd, ftpget, ftpput, fuser, getopt, getty, grep,
        gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd,
        hush, hwclock, id, ifconfig, ifdown, ifenslave, ifplugd, ifup, inetd,
        init, insmod, install, ionice, iostat, ip, ipaddr, ipcalc, ipcrm, ipcs,
        iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5,
        klogd, last, length, less, linux32, linux64, linuxrc, ln, loadfont,
        loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls,
        lsattr, lsmod, lspci, lsusb, lzcat, lzma, lzop, lzopcat, makedevs,
        makemime, man, md5sum, mdev, mesg, microcom, mkdir, mkdosfs, mke2fs,
        mkfifo, mkfs.ext2, mkfs.minix, mkfs.vfat, mknod, mkpasswd, mkswap,
        mktemp, modinfo, modprobe, more, mount, mountpoint, mpstat, mt, mv,
        nameif, nbd-client, nc, netstat, nice, nmeter, nohup, nslookup, ntpd,
        od, openvt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress,
        pivot_root, pkill, pmap, popmaildir, poweroff, powertop, printenv,
        printf, ps, pscan, pstree, pwd, raidautorun, rdate, rdev, readahead,
        readlink, readprofile, realpath, reboot, reformime, remove-shell,
        renice, reset, resize, rev, rm, rmdir, rmmod, route, rpm, rpm2cpio,
        rtcwake, run-parts, runlevel, runsv, runsvdir, rx, script,
        scriptreplay, sed, sendmail, seq, setarch, setconsole, setfont,
```

```
setkeycodes, setlogcons, setsid, setuidgid, sh, sha1sum, sha256sum,
sha512sum, showkey, slattach, sleep, smemcap, softlimit, sort, split,
start-stop-daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd,
swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar,
tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, timeout, top,
touch, tr, traceroute, traceroute6, true, tty, ttysize, tunctl,
tune2fs, udhcpc, udhcpd, udpsvd, umount, uname, unexpand, uniq,
unix2dos, unlzma, unlzop, unxz, unzip, uptime, usleep, uudecode,
uuencode, vconfig, vi, vlock, volname, wall, watch, watchdog, wc, wget,
which, who, whoami, xargs, xz, xzcat, yes, zcat, zcip
```

questa è la versione di busybox installata nella mia rom android. Per vedere tutte le informazione della nostra rom prese anche dal file build.prop che si trova dentro la cartella system, dobbiamo selezionare l'opzione 10:

```
Working folder information


 Android OS version             : 2.2.2
 Device                         : HuaweiU8180
 Model                          : U8180
 ROM Name                       : Morfes Droid 0.2
 Rooted (Superuser.apk + su)    : YES
 Rooted (unsecured boot.img)    : YES
 BusyBox installed              : YES
 BusyBox run-parts support      : YES
 Apps2SD (Apps to EXT) enabled  : NO
 /data/app enabled              : YES
 Custom boot animation allowed  : NO
 Nano text editor installed     : NO
 Bash shell support             : NO
 /system/framework is deodexed  : YES
 /system/app is deodexed        : YES
 radio.img found                : NO
 ROM will wipe all data         : YES

Press Enter to continue
```

Per entrare nel menu opzioni avanzate selezionate lo 0, in questo menu potete deodexare tutte le app nella cartella system e i jar nella cartella framework, vi consiglio di farlo cosi potete modificare qualunque applicazione apk e archivi jar come quello services.jar e android.policy.jar che servono per modificare il colore del testo della barra di notifica e abilitare il menu power del framework android, invece modificando l'apk framework possiamo sostituire le varie icone e il layout del nostro android, personalizzando un proprio tema o mettendo quello della sense o gingerbread, io nella mia rom ho modificato il framework.apk abilitando il menu power e mettendo il tema sense, più avanti accennerò come modificare e personalizzare il proprio framework. Oltre al deodexare vi consiglio pure di aggiungere la funzione */data/app*, in questa cartella possiamo mettere tutte le applicazione “app” che non vogliamo mettere nella */system/app* per non occupare troppo la ram del

device. Se invece volete mettere degli scripts di avvio per attivare alcuni servizi dovete abilitare il supporto busybox run-parts con l'opzione 17, basta creare una cartella init.d in */etc/* e possiamo mettere i vari scripts che verranno avviati ad ogni avvio del device. Personalmente l'opzione più interessante è il **Tools for boot image (unpack/re-pack/etc.)** con questa funzione possiamo avere le informazione del nostro boot:

```
Working folder's boot.img information
-------------------------------------
 Kernel Base Address  : 0x00200000
 Page Size            : 4096 bytes


 Command Line: "mem=211M console=ttyMSM2,115200n8 androidboot.hardware=huawei
console=ttyUSBCONSOLE0 androidboot.console=ttyUSBCONSOLE0"
```

estrarre il kernel e il ramdisk dal boot.img nella cartella WORKING o in un'altra e convertire il boot in una NAND. Dopo aver fatto tutte le varie modifiche nella nosta rom, dobbiamo zipparla creando al suo interno lo script che ci servirà per flashare la rom, fortunatamente l'opzione 99 Build ROM from working folder si occupa di fare tutto questo automaticamente. Quando avviate il build rom vi consiglio di selezionare l'opzione 1 che vi guida alla creazione della rom zippata, vi dice se volete zipalignare i file apk, dite si con y, se volete usare lo script **amend** o quello nuovo **edify**, consiglio di usare lo script edify che sarebbe l' **updater-script**, perchè le ultime recovery clockworkmod supportano solo questo, infine vi chiederà di firmare la vostra rom, anche se di default è impostata a y dovete stare attenti perchè se non firmate la rom la recovery non potrà installarla nel vostro device. Quando la kitchen finisce di creare la rom, la copia nella cartella OUTPUT_ZIP, ora dovete mettere la rom zippata nella vostra sd e flasharla.

RECOVERY

Come avevo accennato la recovery serve per flashare rom modificate, ma non fa solo questo fa anche un intero backup del sistema android. In giro per la rete ci sono varie recovery modificate per vari devices, ma la più diffusa ed adattata per la maggior parte dei devices android è la **Clockworkmod** creata dal team CyanogenMod, essa però supporta solo lo script Edify. Esistono altre recovery come quella di **UltraJack** per il device Huawei Ideos che supporta sia il vecchio script Amend che quello nuovo Edify. Queste recovery oltre a flashare rom e fare backup completi, ti permettono pure di montare le varie partizioni della flash anche la propria sd, ma la funzione importante è che ti permette di formattare la sd in filesystem ext2-3-4 e di creare una partizione swap. Nella recovery c'è anche una opzione chiamata **wipe** che permette di cancellare tutto il contenuto del sistema prima di installare una nuova rom. La recovery viene installata tramite fastboot, mettendo il device in modalità bootloader, i comandi da dare su linux sono questi:

```
sudo ./fastboot flash recovery nomedellarecovery.img
sudo ./fastboot reboot
```

BOOT

Il boot.img di android è simile a quello di una distro linux, quando andiamo a scompattarlo con la cucina di dsixda troviamo al suo interno la **zImage**, l'immagine del kernel compilato e una cartella boot.img-ramdisk, all'interno di questa cartella troviamo degli scripts che servo a caricare i vari moduli, kernel, librerie ed eseguibili durante l'avvio di android.

```
./default.prop
./proc
./dev
./init.rc
./init
./sys
./init.goldfish.rc
./sbin
./sbin/adbd
./system
./data
```

Questo è un esempio del contenuto del ramdisk che può variare dai diversi dispositivi. Il binario e lo script più importante sono l'init e l'init.rc, essi permettono l'inizializzazione dei file del sistema e l'avvio di android. Sta a voi andare ad esplorare e leggere il contenuto degli scripts all'interno del ramdisk per ulteriori info.

KERNEL

Il kernel è il cuore del sistema android, esso gestisce tutto l'hardware del device, wifi, bluetooth, touchscreen, audio, ec... I kernel android di default non hanno abilitati molti supporti e moduli, come per esempio la swap, cifs, nfs, ecc..., quindi per avere un kernel che migliori le prestazioni del nostro android dovremo compilarlo noi. Molti produttori di devices rilasciano i sorgenti del kernel android con le relative patch applicate per funzionare con l'hardware dei loro dispositivi, quindi dovremmo scaricare tali sorgenti per poter compilare il kernel che funzioni con il nostro devices. La procedura che spiego per compilare il kernel può essere adattata per qualsiasi dispositivo android che abbia una architettura **Arm**, prima di tutto dovete avere una distro linux preferibilmente Debian o derivate come Ubuntu installata nel vostro pc, con Windows non potete ne ora ne mai compilare un kernel android, dopo di che dovete aggiungere in */etc/apt/sources.list* alla fine del file questi repository:

```
# -- Emdebian sources.list entries
#
# deb http://www.emdebian.org/debian/ unstable main
# deb http://www.emdebian.org/debian/ testing main
deb http://www.emdebian.org/debian/ lenny main
```

da terminale date queste serie di comandi per poter installare la toolchain di emdebian per la crosscompilazione di architetture arm:

```
sudo apt-get update
```

```
sudo apt-key adv --keyserver keyserver.ubuntu.com -recv-keys
```
(qui dovete mettere il numero della chiave pubblica che vi compare sul terminale dopo aver fatto l'update)

```
sudo apt-get install gcc-4.3-arm-linux-gnueabi linux-libc-dev-armel-cross
```

infine andate nella cartella principale del vostro kernel e compilate con questi comandi:

```
export ARCH=arm
```

```
export CROSS_COMPILE=arm-linux-gnueabi-
```

prima di dare il make menuconfig per avere il menu dove andare ad abilitare i vari moduli, dobbiamo mettere il file config dentro la cartella principale del kernel, per ricavare il config del nostro device, dobbiamo accedere alla shell di adb e con pull prendere il **config.gz** che si trova in */proc/*, lo decomprimiamo e lo mettiamo nella cartella del kernel rinominandolo in **.config**, date i comandi

```
make menuconfig
```

e andiamo a scegliere i vari moduli da abilitare. Per far partire la compilazione diamo

```
make
```

la compilazione ci impiegherà un po' in base alla velocità del processore, quando finisce avremo la **zImage** in */arch/arm/boot* e la copiamo nel nostro boot.img scompattato con la cucina di dsixda, ricompattiamolo e flashamolo con fastboot oppure possiamo creare un pacchetto update.zip che ci permette di flasharlo dalla recovery, ovviamente in questo zip andremo a mettere solo il boot e a modificare l'updater-script.

SYSTEM

La system di android è la parte, dopo quella del boot, più importa perchè in essa vi sono i binari, le librerie, il framework.apk con i relativi archivi jar e i file di configurazione del bluetooth, wifi, audio e altri, che servono per far funzionare l'intero sistema android. Di norma in base alla rom ed alla versione di android il contenuto del **system** dovrebbe essere cosi:

```
./app
./bin
./etc
./fonts
./framework
./lib
./media
./tts
./usr
./wifi
./xbin
./build.prop
```

Sono tutte cartelle, a parte il file build.prop, come potete intuire nella cartella app ci sono tutte le applicazioni del sistema, esempio phone.apk, contact.apk, ec..., nella cartella bin tutti gli eseguibili, nella cartella framework abbiamo il framework.apk e gli archivi jar, nella lib tutte le librerie di sistema, come audio, camera, bluetooth, ecc.., su xbin possiamo metterci la busybox o programmi che andremo a cross compilare nativamente con la ndk android oppure staticamente con una toolchain arm. Nella system non dovete fare molte modifiche, soltanto aggiungere o rimuovere qualche applicazione nella app senza però eliminare quelle che gestiscono il sistema, la parte che sicuramente andremmo a modificare sarà il framework.apk e gli archivi jar come services.jar e android.policy.jar.

FRAMEWORK

L'interfaccia grafica di android viene gestita dal framework.apk che troviamo in */system/framework* e dagli archivi jar. Vi sono varie procedure per poter modificare il framework.apk, sia tramite alcuni programmi sia semplicemente aprendo l'apk, infatti l'apk è l'estensione di un file di archivio come quello zip. Se lo apriamo e andiamo a sostituire l'immagine all'interno della cartella drawable-mdpi in */res/* (per i devices con risoluzione 340x640) con lo stesso nome, possiamo cambiare icone e immagini senza bisogno di decomprimere l'apk con tools come **APK Manager 4.9** o **Apk Tool**, ovviamente l'utilizzo di questi programmi per decomprimere il framework.apk e modificarlo comporta una maggiore difficoltà rispetto quando andiamo solo a sostituire un' immagine o un file xml. Per maggiori info su come utilizzare questi programmi, vi linko i post pubblicato da IceMatrix sul forum di Myppc: http://www.myppc.it/web/community/viewtopic.php?f=59&t=4600, http://www.myppc.it/web/community/viewtopic.php?f=59&t=5509, invece per aggiungere il power menu leggete la guida di bimbomix72 (http://www.myppc.it/web/community/viewtopic.php?f=125&t=6246) per il porting dell'Huawei U8180 e la guida di PaoloM70 (http://www.androidworld.it/forum/modding-e-firmware-lg-optimus-one-114/power-menu-paolom70-13126/) per rom generiche.

PORTING

Il porting di una rom consiste nel mettere una rom di un device in un altro che ha caratteristiche hardware uguali o simili. La kitchen di dsixda ha una funzione che permette di fare un porting tra due rom per devices simili, ma di solito non funziona, infatti come dice anche l'autore della kitchen è in via sperimentale. Io prendendo per esempio il mio porting che ho fatto per la rom U8180 per farla andare nel device Huawei U8150 Ideos, vi spiego come andare a sostituire manualmente le librerie, i binari e i vari moduli che servono per fare funzione wifi, bluetooth, sensore di prossimità, accelerometro, radio fm, ecc... Per la radio ho sostituito l'app con quella originale del nostro Ideos U8150 e sostituito le tre librerie **libaudio.so**, **libaudioflinger.so**, **libaudiopolicy.so** presi da una qualsiasi rom originale U8150. Per il bluetooth ho sostituito l'app e sostituito le tre librerie, **libbluedroid.so, libbluetooth.so e libbluetoothd.so** presi sempre da una rom U8150. Per il wifi ho sostituito in */system/wifi/* i file **firmware.bin, firmware_apsta.bin e nvram.txt**, in */system/etc/wifi/* il file **wpa_supplicant.conf** e ho scompattato il boot.img della rom U8180 tramite la kitchen di dsixda sostituendo il modulo wifi **dhd.ko** che si trova nella cartella *boot.img-ramdisk/wifi/* con quello di una rom originale U8150. Infine per la bussola, accelerometro e sensore di prossimità ho messo in */system/bin/* gli eseguibili **akmd2 e rild** presi ovviamente da una rom U8150 e tolti i binari akmd8975 e akmd8962 ed eliminate le seguenti righe dal file init.rc del boot.img:

```
# /*<DTS2010123100691 modified by yuxuesong on 2010-12-28 begin*/
service akm8975 /system/bin/akmd8975
    disabled
    oneshot

service akm8962 /system/bin/akmd8962
    disabled
    oneshot
# /* DTS2010123100691 modified by yuxuesong on 2010-12-28 end>*/
```

Prima di effettuare un porting, provate la rom senza effettuare nessuna modifica, per vedere quali problemi da e cosa non funziona (wifi, bluetooth, ecc...), dopo di che incominciate a sostituire le

varie librerie, eseguibili o app, sicuramente dovete fare molte prove e molti flash.

COMPILARE ANDROID

Vediamo come possiamo anche compilarci un sistema android dai sorgenti ufficiali di Google (http://source.android.com/source/downloading.html) e quelli del team Cyanogenmod (http://wiki.cyanogenmod.com/wiki/Building_from_source) pubblicati su github. Come ho detto per la compilazione del kernel android, anche per compilare l'intero sistema android ci serve una distro linux, vi consiglio una ubuntu o debian. Iniziamo a scaricarci i sorgenti aprendo una shell e dando questi comandi:

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH

$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

creiamo una cartella, per esempio possiamo chiamarla gingerbread, se vogliamo compilare questa versione di android

```
$ mkdir Gingerbread
$ cd Gingerbread
```

ora specifichiamo nel repo quale versione di android vogliamo scaricare, per l'ultima versione di gingerbread diamo questi comandi:

```
$ repo init -u https://android.googlesource.com/platform/manifest -b android-2.3.7_r1
$ repo sync
```

Prima di avviare la compilazione dobbiamo installare la JDK di Java e alcuni programmi:

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"

$ sudo add-apt-repository "deb-src http://archive.canonical.com/ubuntu lucid partner"
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk


$ sudo apt-get install git-core gnupg flex bison gperf build-essential \
  zip curl zlib1g-dev libc6-dev lib32ncurses5-dev ia32-libs \
  x11proto-core-dev libx11-dev lib32readline5-dev lib32z-dev \
  libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown \
  libxml2-utils
```

Adesso possiamo avviare la compilazione, dobbiamo solo decidere se compilare android generico, per l'emulatore o compatibile per il nostro device. Se optiamo per la compilazione del nostro device, dobbiamo avere dei file di configurazione ed eventuali binari e librerie proprietari che ci servono per compilare. Se siamo fortunati questi file di configurazione li troviamo nei sorgenti CyanogenMod, infatti questo team ha sviluppato la propria rom per molti devices, per un elenco completo consultate il loro wiki (http://wiki.cyanogenmod.com/wiki/Building_from_source) o il repo su github (https://github.com/CyanogenMod). I sorgenti android hanno file di configurazione solo per i device come il Nexus, se noi abbiamo per esempio uno smartphone Huawei U8150 Ideos dobbiamo scaricare i file dal team CyanogemMod e tramite lo script **extract-files.sh** estrarre i binari e le librerie proprietarie da una rom originale U8150 che ci servono per la compilazione e che vengono salvati in *vendor/huawei/u8150/proprietary/*, mentre i file di configurazione vanno messi in *device/huawei/u8150*. Quando abbiamo tutto pronto possiamo iniziare la compilazione, se compiliamo con i sorgenti CyanogenMod non dovremo avere molti problemi durante la compilazione, invece se compiliamo con i sorgenti android avremo abbastanza errori che dovremo

fixare, operazione che può essere svolta solo da persone che sanno programmare. I comandi per compilare sono

```
$ . build/envsetup.sh
$ lunch
```

per vedere la lista dei devices da compilare

```
$ make
```

per fare partire la compilazione.

Non è semplice creare una rom da zero, anche se si riesce a portare a termine la compilazione, di sicuro ci saranno dei problemi o con il modulo telefonico, con il wifi, con l'audio, con la camera, con il bluetooth, con la radio FM, anche le rom Cyanogenmod non sono funzionanti al 100% per alcuni devices.

CROSS COMPILAZIONE DI PROGRAMMI

Come ho già detto android è sviluppato basandosi su sistemi operativi linux, anche se è abbastanza diverso rispetto a linux, possiamo comunque compilare programmi che girano su linux, come nano, file, bash, ssh, nmap, ecc... Possiamo adottare due tipi di compilazione statica o dinamica, per una compilazione statica possiamo utilizzare una qualsiasi toolchain per architettura **arm** come quella di emdebian che abbiamo scaricato per compilare il kernel, invece per cross compilare dinamicamente dobbiamo scaricare e installare la ndk di android. Per esempio se vogliamo cross compilare nmap staticamente con la toolchain emdebian dobbiamo dare questi comandi:

```
$ export CC=arm-linux-gnueabi-gcc-4.3 CPP=arm-linux-gnueabi-cpp-4.3 LD=arm-linux-gnueabi-
ld
$ export CFLAGS=--static


$ export LDFLAGS=--static
$ ./configure --build=arm-linux-gnueabic --host=arm-linux-gnueabi --with-libpcap=included
--with-liblua=included -with-pcap=linux -with-static-linked-ext
```

Invece se vogliamo cross compilare dinamicamente non possiamo farlo con una semplice toolchain arm, perchè le librerie sono diverse da quelle di android, per risolvere questo problema dobbiamo installare la ndk di android. La ndk è una toolchain sviluppata per android che contiene le sue librerie native. Scarichiamo la ndk versione 6 da questo link: http://developer.android.com/sdk/ndk/index.html, è disponibile per linux, mac e windows, ovviamente la mia guida si riferisce per quella di linux e installiamola con questi comandi:

```
$ NDK/build/tools/make-standalone-toolchain.sh --platform=android-8 -install-
dir=/home/giacomo/toolchain/
$ export PATH=/home/giacomo/toolchain-ndk/bin:$PATH


export CC=arm-linux-androideabi-gcc CPP=arm-linux-androideabi-cpp CXX=arm-linux-
androideabi-g++ LD=arm-linux-androideabi-ld
```

dove $NDK e' la 'pwd' cioè il percorso della cartella e supponendo di installarla in /home/giacomo/toolchain, giacomo è la mia cartella user, compiliamo con questi comandi:

```
$ ./configure –host=arm-linux-androideabi
```

o

```
$ ./configure –host=arm-eabi
```

questi comandi variano in base alla configurazione del configure di ogni programma.

La mia guida per ora finisce qui, ho parlato sinteticamente di tutto quello che ho fatto e sperimentato con il sistema android, spero che questo manuale possa servire e dare maggiori chiarimenti per coloro che si avvicinano all'ambiente android, ovviamente aiuta molto coloro che già usano linux, per chi viene da windows questo manuele potrebbe confonderlo un po', proprio per questo l'ho redatto cosi chiaro e semplice. Tutto quello che ho riportato qui è frutto dei miei studi effettuati esclusivamente su internet consultando e leggendo vari siti, blog e forum, linkandoli alcuni anche nella guida.