**[Android File Hierarchy : System Structure Architecture Layout](https://www.cnblogs.com/shangdawei/p/4513604.html)**

Most of the Android user are using their Android phone just for calls, SMS, browsing and basic apps,

But form the development prospective, we should know about  Android internal structure.

Android uses several partitions (like boot, system, recovery,data etc) to organize files and folders on the device just like Windows OS.

Each of these partitions has it’s own functionality, But most of us don’t know  the significance of each partition and its contents.

In this article, we will take you on a tour of Android partitions.

So lets start the android file system tutorial.

There are mainly 6 partitions in Android phones, tablets and other Android devices.

Below is the list of partition for Android File System.

Note that there might be some other partitions available, it differs from Model to Model.

But logically below 6 partitions can be found in any Android devices.

* **/boot**
* **/system**
* **/recovery**
* **/data**
* **/cache**
* **/misc**

****

Also Below are the for SD Card Fie System Partitions.

* **/sdcard**
* **/sd-ext**

Please Note: Only /sdcard partition can be found in all Android devices and the rest are present only in select devices.

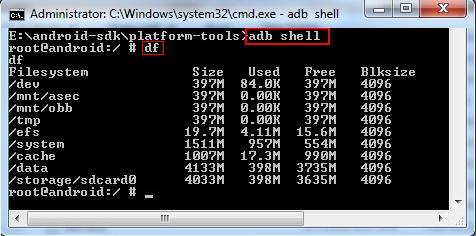
**Know Your Android Device Partition Size using adb Command**

You can know which partitions are available along with the partition size for all partition in your android device.

Go through the below image and run the adb command as shown in that image.

For more adb commands, you can read my atricle [**Useful adb Commands for Android Development.**](http://techblogon.com/adb-commands-for-android-development/)

Also for more details for android architecture, you can read my article [**Android Architecture**](http://techblogon.com/android-architecture/).



Note:

boot and recovery partition is not displayed in the above image.

So after adb shell, you need to run mount command. like

cat proc/mounts

**/boot**

This is the boot partition of your Android device, as the name suggests.

It includes the android kernel and the ramdisk.

The device will not boot without this partition.

Wiping this partition from recovery should only be done if absolutely required and once done,  
the device must NOT be rebooted before installing a new one,  
which can be done by installing a ROM that includes a /boot partition.

**/system**

As the name suggests, this partition contains the entire Android OS, other than the kernel and the ramdisk.

This includes the Android GUI and all the system applications that come pre-installed on the device.

Wiping this partition will remove Android from the device without rendering it unbootable,  
and you will still be able to put the phone into recovery or bootloader mode to install a new ROM.

**/recovery**

This is specially designed for backup.

The recovery partition can be considered as an alternative boot partition,  
that lets the device boot into a recovery console for performing advanced recovery and maintenance operations on it.

**/data**

Again as the name suggest, it is called user data partition.

This partition contains the user’s data like your contacts, sms, settings and all android applications that you have installed.

While you perform factory reset on your device, this partition will be wiped out,  
Then your device will be in the state, when you used for the first time or the way it was after the last official or custom ROM installation.

**/cache**

I hope you have some idea about cache, as you are expert on internet browsing.

This is the partition where Android stores frequently accessed data and app components.

Wiping the cache doesn’t effect your personal data but simply gets rid of the existing data there,  
which gets automatically rebuilt as you continue using the device.

**/misc**

This partition contains miscellaneous system settings in form of on/off switches.

These settings may include CID (Carrier or Region ID), USB configuration and certain hardware settings etc.  
This is an important partition and if it is corrupt or missing, several of the device’s features will will not function normally.

**/sdcard**

This is not a partition on the internal memory of the device but rather the SD card.

In terms of usage, this is your storage space to use as you see fit, to store your media, documents, ROMs etc. on it.

Wiping it is perfectly safe as long as you backup all the data you require from it, to your computer first.

Though several user-installed apps save their data and settings on the SD card and wiping this partition will make you lose all that data.

On devices with both an internal and an external SD card – devices like the Samsung Galaxy S and several tablets –  
the /sdcard partition is always used to refer to the internal SD card.

For the external SD card – if present – an alternative partition is used, which differs from device to device.  
In case of Samsung Galaxy S series devices, it is /sdcard/sd while in many other devices, it is /sdcard2.

Unlike /sdcard, no system or app data whatsoever is stored automatically on this external SD card  
and everything present on it has been added there by the user.

You can safely wipe it after backing up any data from it that you need to save.

**/sd-ext**

This is not a standard Android partition, but has become popular in the custom ROM scene.

It is basically an additional partition on your SD card that acts as the /data partition  
when used with certain ROMs that have special features called APP2SD+ or data2ext enabled.

It is especially useful on devices with little internal memory allotted to the /data partition.

Thus, users who want to install more programs than the internal memory allows can make this partition  
and use it with a custom ROM that supports this feature, to get additional storage for installing their apps.

Wiping this partition is essentially the same as wiping the /data partition – you lose your contacts, SMS, market apps and settings.

Now when you install a new binary, you  can know what you’re going to loose, make sure to backup your data before flash new binary in your android device.

I hope this small tutorial will help you at it’s best. If you have nay query or comments, please share with us, we will get back to you asap.

**Understanding the Android File Hierarchy**

A search of the Internet reveals very little about the Android file hierarchy.

For Windows users, it is another world.

For Linux users, it is a variation of the Linux file hierarchy.

In Linux / Android / Unix (for simplicity, referred to a just Linux), the file hierarchy is a single tree,

with the top of the tree being "/" - the *root* of the tree.

Under "/" are files and directories.

The Linux file hierarchy lacks the concept of drives, as in Windows.

Instead, file systems are mounted on a directory to create a single integrated tree.

For media based file systems. the file system represents a partition of some media.

It makes no difference whether the file system exists on the local device, or on a remote device.

Everything is integrated into a single file hierarchy that begins with root.

The term path refers to the path to a file or directory.

The absolute path to any file or directory begins with "/" - root.

For example, your file application may show the the path to the internal flash drive as */mnt/sdcard*, or simply "*sdcard*."

If your Android device includes an external SD Card, you will also see*/mnt/extSdCard*.

The actual names may vary, but the concept remains the same.

For example, CyanogenMod mounts the external flash drive appears on*/mnt/sdcard*, and the internal flash as*/mnt/emmc*.

The important point to remember is that each of these directories represents a *mount point* for a file system.

As a normal user, an Android file only displays the file systems to which you have access.

These file systems are just a part of the entire file hierarchy.

To see the complete Android file hierarchy, you need "root" access.

In this case, "root" refers to a special user account that has system administration privileges.

When you "root" a Android device, you are gaining access to the root account.

There is a reason for not automatically granting root access - one tiny mistake can go a long way.

However, for those experienced in Linux commands and Linux system administration, it opens the hood to the underlying structure of Android.

**Android File Systems**

As mentioned in [Android Kernel versus the Linux Kernel](http://www.all-things-android.com/content/android-kernel-versus-linux-kernel), Android uses the Linux kernel.

All file, and directory, operations from an application flow through a kernel abstract layer called the Virtual File System (VFS).

Each file system, and there are a lot of them, are implementations of VFS.

Each file system has a separate kernel module that registers the operations that it supports with VFS.

By separating the implementation from the abstraction, adding a new file system becomes a matter of writing another kernel module.

These modules are either part of the kernel, or are dynamically loaded on demand.

The Android kernel comes with a subset of the vast collection of file systems that range  
from the Journal File System (JFS) for AIX (IBM's flavor of Unix) to the Amiga file system.

All the fancy footwork is hidden from the user, as the kernel handles all the work when it mounts a file system.

The kernel configuration file determines what file system modules are compiled, and whether they are built into the kernel, or dynamically loaded.

Consequently, the Android kernel only contains those file system modules that are relevant to its operation.

In fact, I don't know of a single Linux distribution that compiles every file system module, as some of the file systems are hardware architecture dependent.

While the supported file systems vary on different Android devices, the common flash memory file systems are as follows:

* **exFAT** - The extended File Allocation Table is a Microsoft proprietary file system for flash memory.  
  Due to the licensing requirements, it is not part of the standard Linux kernel.  
  However, some manufactures provide Android support for the file system.
* **F2FS** - Samsung introduced The Flash-Friendly File System as an open source Linux file system in 2012.
* **JFFS2** - The Journal Flash File System version 2 is the default flash file system for the AOSP (Android Open Source Project) kernels,  
  since Ice Cream Sandwich. JFFS2 is a replacement to the original JFFS.
* **YAFFS2** - Yet Another Flash File System version 2 was the default AOSP flash file system for kernel version 2.6.32.  
  YAFFS2 is not supported in the newer kernel versions, and does not appear in the source tree for the latest kernel versions from kernel.org.  
  However, individual mobile device vendors may continue to support YAFFS2.

Besides flash memory file systems, Android devices typically support the following media-based file systems:

* **EXT2** / **EXT3** / **EXT4** - The EXTended file system is the standard Linux file system, with EXT4 being the current version.  
  Since 2010, EXT4 is often used in place of YAFFS2 or JFFS2 as the file system for internal flash memory on Android devices.
* **MSDOS** - The MSDOS driver supports the FAT12, FAT16 and FAT32 file systems.
* **VFAT** - The VFAT is not actually a file system, but an extension to the FAT12, FAT16, and FAT32 file systems.  
  Thus, you will always see the VFAT kernel module in conjunction with the MSDOS module.  
  External SD Cards are commonly formatted using VFAT.

The above file systems are media-based file systems.  
VFS also supports pseudo file systems, which are not media based.  
The Linux kernel supports a number of pseudo file systems, the ones that are important to Android devices are:

* **cgroup** - The cgroup (control group) pseudo file system provides a means to access, and define, various kernel parameters.  
  While cgroup is the pseudo file system, there are a number of different process control groups.  
  If your Adroid device supports process control groups, you will find a list of the groups in the file   
  */proc/cgroups*. Android uses cgroups for acct (user accounting), and cpuctl (CPU control).
* **rootfs** - This file system serves as the mount point for the root file system ("/").
* **procfs** - The procfs file system is normally mounted on the */proc* directory, and reflects a number of kernel data structures.  
  The  operations on these files actually read live kernel data.  
  The number directories reflect the process IDs (actually, the thread group leader process ID) for each running task.  
  The */proc/filesystems* file generates a list of the currently registered file systems.  
  File systems followed by NODEV are  pseudo file systems, as there is no related device.  
  The */proc/sys* directory contains kernel parameters, some of which are tunable.
* **sysfs** - The device model for the kernel is an object-oriented structure that reflects the devices known  
  by the kernel through the sysfs file system, which is normally mounted on the */sys* directory.  
  When a kernel discovers a new device, it builds an object in the*/sys/devices* directory.  
  The kernel uses a network socket to communicate the new device information to the udevd daemon,  
  which builds an entry in the*/dev* directory.  
  The */sys/fs* directory contains the kernel object structures for media based file systems.  
  The */sys/module* directory contains objects for each loaded kernel module.
* **tmpfs** - The tmpfs file system is often mounted on the*/dev* directory.  
  Since it is a pseudo file system, any data in the */dev*directory is lost when the device is rebooted.

While there seems like a lot of file systems, they are just the tip of the iceberg.

For those without root access, these file systems and directories are hidden from your view, as you do not have the permission to access them.

My Samsung Galaxy SIII provides a*USB debugging* option under *Developer options* in the*Settings* menu.

When enabled this option allows one to connect to the Samsung Galaxy SIII using adb (Android Debug Bridge).

While it only provides user level privileges, it does provide access to most directories and most of the Linux commands.

This option may be available on other Samsung Android devices.

You can also try the [Terminal Emulator](https://play.google.com/store/apps/details?id=jackpal.androidterm) app. It works on my Samsung Galaxy SIII, as well as my B&N Nook Color running CyanogenMod 10 (Jelly Bean).

**Android File Hierarchy**

As mentioned above, the Android file hierarchy is a modified version of the traditional Linux file hierarchy.

There are slight variations in the structure across the different version of Linux, and from different manufacturers.

However, the variations are slight.

The following gives a brief summary of the top levels of the directory structure for the AOSP release of Jelly Bean:

* **acct** - This directory is the mount point for the acct cgroup (control group), which provides for user accounting.
* **cache** - The mount point for the*/dev/block/mtdblock2* partition (the partition name may vary).  
  The size of the cache is limited to the size of this partition.
* **d** - A symbolic link to */sys/kernel/debug*.
* **data** - The mount point for the */dev/block/mtdblock1* partition.
* **default.prop** - This file defines various default properties.
* **dev** - The mount point for a tmpfs file system, which defines the devices available to applications.  
  The */dev/cpuctl* directory is the mount point for cpuctl control group, using the cgroup pseudo file system.
* **etc** - A symbolic link to */system/etc*.
* **init** - A binary program that processes the *init.rc* file.  
  The *init.rc* file imports the other *init.\*.rc* files.  
  When booting Android, the kernel executes the init program at the end of its boot process.  
  It is worth reading the *init.rc* file, as it tells the story of the configuration for the Android device.  
  Since Android does not support*/etc/sysctl.conf*, the updates to the */proc/sys/kernel* parameters are part of the*init.rc* file.  
  Unless you have a good understanding of the internal workings of the Linux kernel, you should not modify these parameters.  
  The same is true of the */dev/cpuctl* parameters.
* **mnt** - Besides the mounts for the internal and external SD cards, these directory serves as a mount point for other file systems.  
  The */mnt/asec* directory is a mount point for a tmpfs file system, and is part of Android security.  
  The */mnt/obb* directory is a mount point for the tmpfs file system, and stores expansion files for applications whose files exceed 50MB.  
  The */mnt/secure* directory is another component of Android security. You may also see mount points for one, or more, USB devices.
* **proc** - Mount point for the procfs file system, which provides access to kernel data structures.  
  Programs, such as ps, lsof, and vmstat, use the */proc* as the source for their information.
* **root** - The home directory for the root account.
* **sbin** - While much smaller than the */sbin* directory in a standard Linux distributions, it does contain the binaries for several important daemons.
* **sdcard** - A symbolic link to */mnt/sdcard*.
* **sys** - The mount point for the sysfs pseudo file system, which is a reflection of the kernel's device object structure.  
  There is a lot of information in this directory, but it does require understanding of the kernel device model.  
  In a nutshell, the directories represent kernel objects, and files are attributes of those objects.
* **system** - This directory is the mount point for the */dev/block/mtdblock0*.  
  Under this directory are the directories that you normally see under the root directory of a standard Linux distribution.  
  These directories include *bin*, *etc*, *lib*, *usr*, and *xbin*.
* **ueventd.goldfish.rc** **ueventd.rc** - These files define the configuration rules for the */dev* directory.
* **vendor** - A symbolic link to */system/vendor*.

The above is just a thumbnail sketch of the Android file structure.

To understand the workings of Android beneath the desktop, you need a fundamental knowledge of Linux, and the Linux command-line.

For kernel parameters, you need a working knowledge of the Linux kernel. In future articles, I shall explore the Android file hierarchy in more detail.