

Standalone partitioning explained

Bharat edited this page on Aug 14 · 4 revisions

Standalone partitioning (and booting) explained

Standalone partitioning

A standalone .img file is a complete byte-for-byte image of an *entire* SD card. It includes the [MBR](#) (which stores the [partition table](#)) as well as all the raw partitions. This is the lowest-level possible way of [writing an SD card](#), but it's the only way to access multiple partitions on an SD card using Windows (via the Win32DiskImager software).

Writing an image like `2014-06-20-wheezy-raspbian.img` to an SD card typically creates two partitions:

1. A small FAT-format **boot** partition, which contains the Raspberry Pi [firmware](#), a compiled [Linux kernel](#) and some config files. (This is the only partition that Windows is able to 'see'.)
2. A large [ext4](#)-format **root** partition, which contains all the files directly used by Linux, including all the applications and your user's home directory.

Because the partition table (and raw partitions) are copied directly from the .img file to the SD card, the root partition will be the exact size that the .img specifies. Therefore if your SD card is much larger than the .img file, you'll have a lot of unused space after the end of the root partition. To make use of this unused space you can use a tool like [raspi-config](#) to automatically resize the root partition as large as possible.

Partition naming

On the Raspberry Pi, the whole SD card is referred to (by Linux) as `/dev/mmcblk0`, the first (boot) partition is referred to as `/dev/mmcblk0p1` and the second (root) partition is referred to as `/dev/mmcblk0p2`. (Similarly, if accessing the SD card on a Linux PC using a USB SD card-reader, the whole card *might* be `/dev/sdb`, the first partition would then be `/dev/sdb1` and the second partition would be `/dev/sdb2`. If you have multiple drives connected to your PC, these names might change to `/dev/sdc`, `/dev/sdc1` & `/dev/sdc2`.)

Standalone booting

Unlike a traditional PC, a Raspberry Pi doesn't use any of the [bootstrap code](#) stored in the MBR. When the Raspberry Pi is powered on, it instead:

1. Loads and runs the `bootcode.bin` file from the first partition (`/dev/mmcblk0p1`) on the SD card, provided the partition has a FAT filesystem.

▼ Pages 3

Home

[NOOBS partitioning explained](#)

[Standalone partitioning explained](#)

Clone this wiki locally

<https://github.com/raspbe> 

 Clone in Desktop

2. `bootcode.bin` in turn then loads and runs `start.elf` (and `fixup.dat`) also from the first partition.
3. `start.elf` then reads `config.txt` and sets up any GPU configuration requested.
4. `start.elf` then reads `cmdline.txt` and loads and runs `kernel.img` (the Linux kernel) or `kernel7.img` (for Raspberry Pi 2 (B)), passing it the entire command-line that it read from `cmdline.txt`. (On most standalone Raspberry Pi images the root filesystem is on the second partition, so there'll be a `root=/dev/mmcblk0p2` option somewhere in `cmdline.txt`.)
5. The loaded Linux kernel then parses the command-line that it was given to determine where its root filesystem is stored (`/dev/mmcblk0p2` in most cases).
6. The Linux kernel then mounts the `/dev/mmcblk0p2` partition as the root (`/`) filesystem, and continues booting the rest of the system from there.
7. In most cases the `/etc/fstab` file (File System TABLE) on the root filesystem will have a line asking for the `/dev/mmcblk0p1` partition to be mounted at `/boot`. This allows you to modify files on the first partition (`/dev/mmcblk0p1`) from within Linux. For example, you could edit `/boot/config.txt` and these modifications to `config.txt` would then get read by `start.elf` when you next rebooted the Pi.

