GitHub This repository Search Explore Features Enterprise Pricing Sign up Sign in

raspberrypi / noobs Watch 133 Star 767 Fork 168

Code Issues 60 Pull requests 21 Wiki Pulse Graphs

NOOBS partitioning explained

Ankou edited this page on Jan 9 · 5 revisions

Pages 3

Home

NOOBS partitioning explained

Standalone partitioning explained

Clone this wiki locally

https://github.com/raspbe

Clone in Desktop

NOOBS partitioning (and booting) explained

The multiple partitions that NOOBS divides your SD card into (at *least* 5) can be quite overwhelming and confusing. This page will try and explain how it all works, and illustrate how NOOBS differs from the 'traditional' standalone images.

Non-NOOBS partitioning

Before you can understand how NOOBS partitioning works, you need to understand how standalone partitioning works on the Raspberry Pi, so go and read that page if you haven't already.

More partition naming theory

For historical reasons (only 4 'slots' in the partition table), hard-drives and SD cards can only have a maximum of 4 **primary** partitions. To work around that limitation, one of those primary partitions can optionally be an **extended** partition. An extended partition can contain an unlimited number of **logical** partitions inside it. Under Linux, the primary partitions are always numbered 1-4 (i.e. seen as `/dev/mmcblk0p1 - /dev/mmcblk0p4` on the Pi), and any logical partitions are always numbered 5 and above (i.e. seen as `/dev/mmcblk0p5` and above on the Pi).

NOOBS partitioning

When NOOBS is first copied to a FAT-format SD card, there's just a single partition taking up all the space on the card, and this is where the files from the NOOBS zipfile get written to. In tabular form it looks like:

Primary partition	Logical partition	Type	Label	Contents
1		FAT	New Volume	NOOBS boot files & initramfs, OS recovery images

The *only* difference between NOOBS and NOOBS lite is that NOOBS lite doesn't include any OS recovery images.

NOOBS bootup (low-level)

When the Raspberry Pi is powered on with a NOOBS card inserted, it:

1. Loads and runs `bootcode.bin` from the FAT-format `/dev/mmcblk0p1`, exactly as it does for standalone images. (This behaviour is built into the BCM2835's internal firmware on all Pis, and so can't be changed.)
2. `bootcode.bin` then spots that `start.elf` is missing, so it loads and runs `recovery.elf` instead.
3. Running `recovery.elf` then switches the firmware into "NOOBS mode" - it uses `recovery.img` instead of `kernel.img`, `recovery.cmdline` instead of `cmdline.txt`, and it sets the root filesystem to `recovery.rfs`.
4. `recovery.elf` then reads `recovery.cmdline` and loads and runs `recovery.img` (the Linux kernel), passing it the entire command-line that it read from `recovery.cmdline` and telling it to load `recovery.rfs` as the root filesystem (an initramfs containing various scripts and the NOOBS GUI application).
5. What happens next depends on which 'mode' NOOBS is operating in...

NOOBS bootup (setup mode)

If `runinstaller` is present in the kernel command-line, then this must be the first time NOOBS has been booted, so it enters 'setup mode'. It then:

1. Automatically shrinks the first (and only) partition `/dev/mmcblk0p1`, making it just large enough to hold whatever files it contains, and labels it as 'RECOVERY'. For NOOBS lite this partition will have a size of approximately XMB; for NOOBS this partition will have a size of approximately XGB.
2. Creates a new large empty extended partition `/dev/mmcblk0p2`, using up the vast majority of the remaining card space.
3. Creates a new small (32MB) ext4-format partition `/dev/mmcblk0p3` at the end of the card, and labels it as 'SETTINGS'. This is used to store files telling NOOBS which OSes are installed (and what partitions they're installed on), which OS should be loaded by default, which language/keyboard NOOBS should use, etc.
4. Removes `runinstaller` from `recovery.cmdline` to prevent this process from being triggered again.

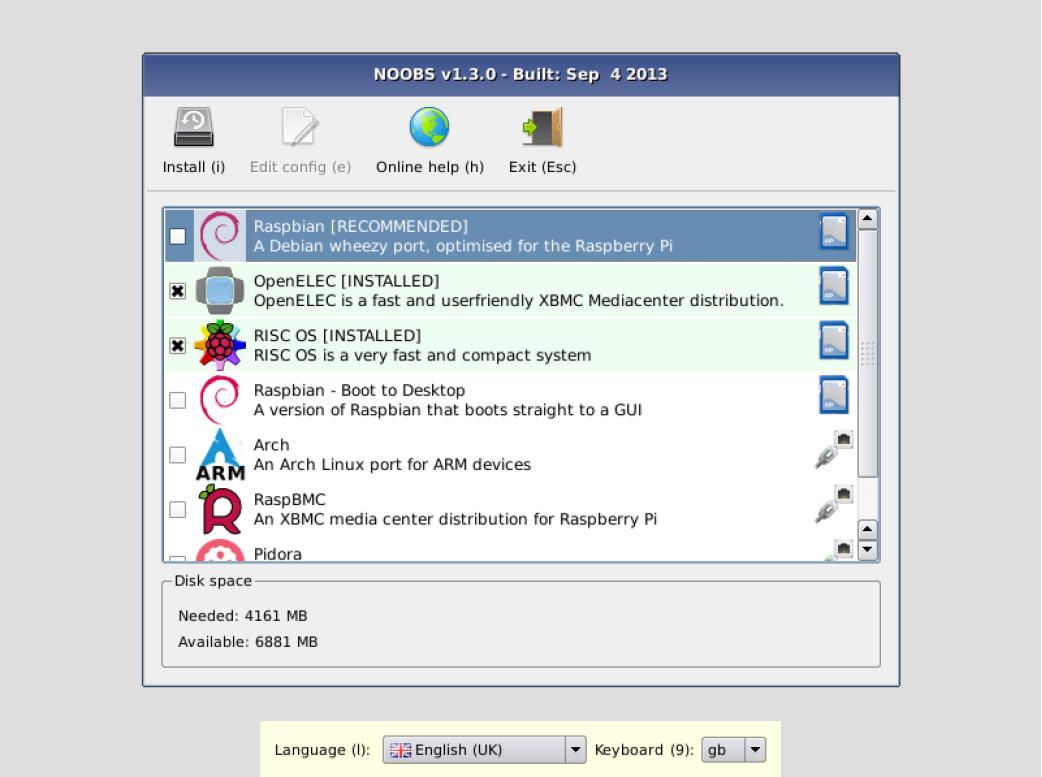
The settings are stored on a small auxiliary partition rather than the same `/dev/mmcblk0p1` partition as everything else. This is because of the NOOBS 'prime directive' - "NOOBS never writes to the first FAT partition. FAT partition the first, NOOBS no writee...". By never writing anything to the first partition (after the 'setup mode' has finished), this ensures that the first partition can never become corrupted; and so NOOBS 'recovery mode' will **always** be accessible (to allow OSes to be re-installed), no matter what happens to the rest of the SD card.

This then changes the partitions to:

Primary partition	Logical partition	Type	Label	Contents
1		FAT	RECOVERY	NOOBS boot files & initramfs, OS recovery images
2		extended		Any logical partitions
3		ext4	SETTINGS	NOOBS settings

NOOBS bootup (recovery mode)

If NOOBS detects that no Operating Systems have been installed yet, or if the user is pressing the Shift key (or any of the other trigger actions are in effect), NOOBS enters 'recovery mode'. This displays the OS-installation menu, allowing the user to choose which OS(es) to install. Refer to the normal documentation for more details a about this menu.



As you may have guessed, the "Available space" displayed here is the size of the extended `/dev/mmcblk0p2` partition, which is where all the OSes get installed to.

NOOBS datafiles

In contrast to the standalone images described earlier (which contain raw partitions), NOOBS instead uses (compressed) tarballs of the partition contents, along with a bunch of settings files. NOOBS is responsible for actually creating the partitions on the SD card itself, which means the partitions are always created at the "correct" size in the first place, there's no need to resize them later. And unlike the low-level raw partitions, the tarballs don't store unused disk blocks.

NOOBS OS installation

For the first example, let's assume that the user is installing just Raspbian. The partitions.json (which can be viewed online here) then specifies which partitions should be created, how big they should be, and which filesystems they should use. In this example it would create a 60MB FAT partition (`/dev/mmcblk0p5`), format it, and extract the contents of `boot.tar.xz` to it. As the root partition has `want_maximised: true` it would then create an ext4 partition (`/dev/mmcblk0p6`) filling up the *entirety* of the rest of the extended partition, format it, and extract the contents of `root.tar.xz` to it. This gives us the full partition layout shown in the table earlier. It then runs the `partition_setup.sh` script which mounts these new partitions, and edits files (typically just `cmdline.txt` on the boot partition and `/etc/fstab` on the root partition) to tell Raspbian which partitions it got installed to. This allows Raspbian to adjust itself to being stored on `/dev/mmcblk0p5` and `/dev/mmcblk0p6` instead of `/dev/mmcblk0p1` and `/dev/mmcblk0p2` . And finally it updates the settings partition with details of the OS we just installed.

Primary partition	Logical partition	Type	Label	Contents
1		FAT	RECOVERY	NOOBS boot files & initramfs, OS recovery images
2		extended		Any logical partitions
	5	FAT	boot	Raspbian boot files
	6	ext4	root	Raspbian root filesystem
3		ext4	SETTINGS	NOOBS settings

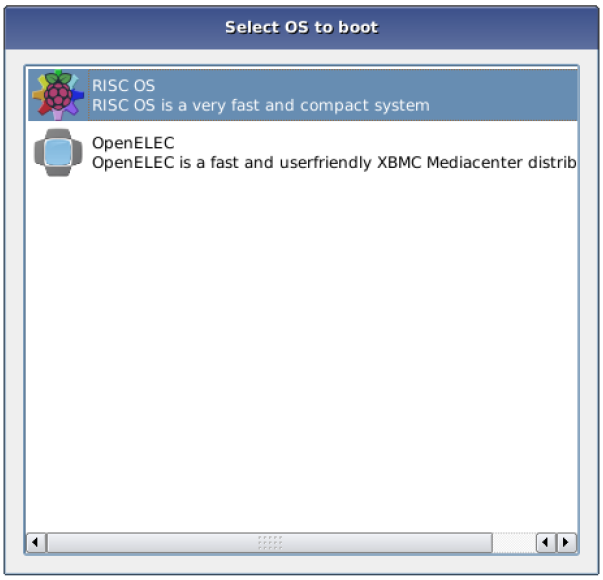
If instead we were installing Raspbian *and* ArchLinux then we might end up with ArchLinux's boot partition as `/dev/mmcblk0p5`, ArchLinux's root partition as `/dev/mmcblk0p6`, Raspbian's boot partition as `/dev/mmcblk0p7` and Raspbian's root partition as `/dev/mmcblk0p8`. As both Raspbian's and ArchLinux's `partitions.json` file specify one of their partitions as `want_maximised: true` then we'd end up with two small boot partitions and two large-as-possible root partitions. NOOBS never 'wastes' any space on an SD card.

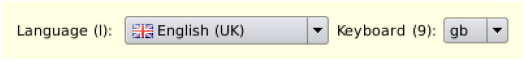
Primary partition	Logical partition	Type	Label	Contents
1		FAT	RECOVERY	NOOBS boot files & initramfs, OS recovery images
2		extended		Any logical partitions
	5	FAT	boot	ArchLinux boot files
	6	ext4	root	ArchLinux root filesystem
	7	FAT	boot1	Raspbian boot files
	8	ext4	root1	Raspbian root filesystem
3		ext4	SETTINGS	NOOBS settings

NOOBS bootup ('boot mode')

If the user isn't pressing the Shift key, and (using the information stored on the settings partition) NOOBS detects that only one bootable Operating System has been installed, it automatically boots that Operating System. It does this by reading the settings partition to determine the boot partition for that OS, and then instructs the firmware to "soft-reboot" using the OS's boot partition. This then 'reboots' the firmware and loads start.elf from the specified partition (typically /dev/mmcblk0p5 if only one OS is installed) and then proceeds the same as the standalone boot described at the very top of this page - start.elf loads kernel.img and reads cmdline.txt, and then kernel.img uses the command-line that was passed to it to determine which partition the root filesystem is stored on (typically /dev/mmcblk0p6 if only one OS is installed), and loads the rest of the system from there.

If instead multiple Operating Systems have been installed, NOOBS then displays the OS-boot menu, allowing the user to choose which OS(es) to boot.





Once the user has selected an option (or if the menu times out and defaults to the last-booted option) then the boot proceeds as described immediately above, with NOOBS using the information on the settings partition to determine which partition to "soft-reboot" as the boot partition.

If using the autoboot.txt feature described here then bootcode.bin immediately "soft-reboots" to the specified partition at power-on, and skips loading NOOBS entirely.

Booting RISC OS within NOOBS

The one small caveat to the above is that RISC OS doesn't understand partition tables, and so it has to be installed to a specific partition at a specific offset. This is what the riscos-boot.bin file is for, and why the RISC OS 'root' partition is still stored as a raw partition and not as a tarball. However NOOBS handles all these details for you automatically, and it's still possible to install other OSes alongside RISC OS.

© 2015 GitHub, Inc. Terms Privacy Security Contact Help

Status API Training Shop Blog About Pricing