Raspberry Pi

Part 16
-
Devices, Partitions and Filesystems on Your Pi

Version: 2017-08-10

Devices, Partitions and Filesystems

People are commonly confused by the difference between three distinct things:

- A random access block storage device such as an SD card (or HDD).
- A storage partition which is a section of a device; there may only be one which occupies pretty much the whole device.
- A filesystem which is something used to organize the data on a partition.

The significance of the second is that it is defined according to a particular scheme, and this information is stored somewhere on the device separate from the partition itself. Currently there are two such schemas used predominantly, MBR and GPT. What these primarily implement is a partition table which describes the layout of the partitions on a device.

The partition table may include a clue about what kind of filesystem can be expected to be found in a partition, and thus where most people get most confused is the difference between the second and the third. They are not synonymous. A partition may be indicated to exist in the partition table, containing a certain kind of filesystem, but this does not mean that the partition actually does contain such a filesystem. For that to be true, the partition must be formatted correctly.

Beware this formatting is distinct from the formatting of the device, which just creates the MBR or GPT style partition table. Some tools (such as fdisk) may create partitions and tag them as containing a particular kind of filesystem, but they do not create the actual filesystem -- so again, that a partition table lists a partition as being of a certain sort does not make it true!

The partition/filesystem formatting includes information about what and where things like directories and files are. Pi image files generally contain two partitions with two different kinds of filesystem:

A small (usually about 50 to 60 MB) partition with a FAT32 filesystem. This is the "boot partition" which the hardware loads firmware from, then the firmware loads a bootloader, then the bootloader loads an operating system kernel. Or something along those lines. The Pi hardware requires this partition, and it requires the SD card use an MBR style partition table.

A second, much larger partition containing an ext4 filesystem. This is not required and since it is the OS kernel that accesses it, it could be of any sort. Ext4 is the "native" filesystem used by the linux kernel, although again, it could be many other things. There is a much wider variety of filesystems than there are partitioning schemes.

Image files can represent either devices or partitions. The image files used to format Raspberry Pi SD cards are usually device images, complete with MBR and multiple partitions. When these are copied block for block onto a physical medium, presto, it is as if someone formatted the device and a set of partitions containing data.

Partitions and filesystems have defined sizes that can be changed, but they still must be defined. Filesystems may contain any amount of data up to their defined size limit.

A 5 GB filesystem requires a partition >= 5GB. If it only contains 100 MB of data, this means 4.9 GB of the filesystem is free, and the partition mostly empty. However, that space must actually exist, or else things start to become complicated in a way that most users will not find entertaining.

The image files distributed for use on the Pi are minimal in size for a couple of obvious reasons:

So they will fit on as small a card as possible.
So that people do not waste bandwidth transferring, e.g., 4.9 GB of nothing.

That's why you need to expand first the partition and then the filesystem to fill whatever card you are using. Recent editions of Raspbian may do this automatically.

Commands to check devices, partitions and filesystems on Raspbian

Let's take a look at some commands that can be used to check up the partitions on your system. The commands would check what partitions there are on each disk and other details like the total size, used up space and file system etc.

First you should install `hwinfo`

```
sudo apt-get install -y hwinfo
```

The hwinfo is a general purpose hardware information tool and can be used to print out the disk and partition list. The output however does not print details about each partition.

```
$ sudo hwinfo --block --short
disk:
  /dev/ram11            Disk
  /dev/ram2             Disk
  /dev/ram0             Disk
  /dev/ram9             Disk
  /dev/ram7             Disk
  /dev/ram14            Disk
  /dev/ram5             Disk
  /dev/ram12            Disk
  /dev/ram3             Disk
  /dev/ram10            Disk
  /dev/ram1             Disk
  /dev/ram8             Disk
  /dev/ram15            Disk
  /dev/ram6             Disk
  /dev/mmcblk0          Disk
  /dev/ram13            Disk
  /dev/ram4             Disk
partition:
  /dev/mmcblk0p1        Partition
  /dev/mmcblk0p2        Partition
```

Besides the many in memory disks (`/dev/ram??`), you see also the SD card as `/dev/mmcblk0`
You see also that there are 2 partitions `/dev/mmcblk0p1 and /dev/mmcblk0p2`.
Next we are going to check the partions using `fdisk`, the most commonly used command to check the partitions on a disk. The `fdisk` command can display the partitions and details like file system type.

```
$ sudo fdisk -l

Disk /dev/mmcblk0: 29.6 GiB, 31724666880 bytes, 61962240 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
Disklabel type: dos
Disk identifier: 0x872806b7

Device         Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk0p1        8192   137215   129024   63M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      137216 31116287 30979072 14.8G 83 Linux
```

`parted` is another command line utility to list out partitions. It allows you also to modify them if needed. Here is an example that lists out the partition details.

```
$ sudo parted -l
Model: SD USDU1 (sd/mmc)
Disk /dev/mmcblk0: 31.7GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      4194kB  70.3MB  66.1MB  primary  fat16        lba
 2      70.3MB  15.9GB  15.9GB  primary  ext4
```

`df` is not a partitioning utility, but prints out details about only mounted file systems. The list generated by `df` even includes file systems that are not real disk partitions.

Here is a simple example

```
$ df -h
Filesystem                                Size  Used Avail Use% Mounted on
/dev/root                                  15G  2.4G   12G  17% /
devtmpfs                                  426M  4.0K  426M   1% /dev
tmpfs                                     430M     0  430M   0% /dev/shm
tmpfs                                     100M  5.8M   95M   6% /run
tmpfs                                     5.0M  4.0K  5.0M   1% /run/lock
tmpfs                                     430M     0  430M   0% /sys/fs/cgroup
tmpfs                                     100M  1.1M   99M   2% /var/log
tmpfs                                      10M     0   10M   0% /var/tmp
tmpfs                                     100M  1.5M   99M   2% /tmp
/dev/mmcblk0p1                             63M   22M   42M  35% /boot
```

Only the file systems that start with a `/dev` are actual devices or partitions.
Use grep to filter out real hard disk partitions/file systems.

```
$ df -h | grep ^/dev
/dev/root                                  15G  2.4G   12G  17% /
/dev/mmcblk0p1                             63M   22M   42M  35% /boot
```

To display only real disk partitions along with partition type, use df like this

```
$ df -h --output=source,fstype,size,used,avail,pcent,target -x tmpfs -x devtmpfs
Filesystem                              Type  Size  Used Avail Use% Mounted on
/dev/root                               ext4   15G  2.4G   12G  17% /
/dev/mmcblk0p1                          vfat   63M   22M   42M  35% /boot
```

Note that `df` shows only the mounted file systems or partitions and not all.

`Lsblk` lists out all the storage blocks, which includes disk partitions and optical drives. Details include the total size of the partition/block and the mount point if any. It does not report the used/free disk space on the partitions.

```
$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0    0 29.6G  0 disk
├─mmcblk0p2 179:2    0 14.8G  0 part /
└─mmcblk0p1 179:1    0   63M  0 part /boot
```

Note that ff there is no MOUNTPOINT, then it means that the file system is not yet mounted. For cd/dvd this means that there is no disk.
`lsblk` is capbale of displaying more information about each device like the label and model.

```
sudo lsblk -o name,mountpoint,label,size
NAME        MOUNTPOINT LABEL  SIZE
mmcblk0                      29.6G
├─mmcblk0p2 /                14.8G
└─mmcblk0p1 /boot      boot    63M
```

You can specify plenty of columns in whatever order you like. Available columns are:

```
      NAME  device name
     KNAME  internal kernel device name
   MAJ:MIN  major:minor device number
    FSTYPE  filesystem type
MOUNTPOINT  where the device is mounted
     LABEL  filesystem LABEL
      UUID  filesystem UUID
        RO  read-only device
        RM  removable device
```

```
     MODEL  device identifier
      SIZE  size of the device
     STATE  state of the device
     OWNER  user name
     GROUP  group name
      MODE  device node permissions
ALIGNMENT  alignment offset
   MIN-IO  minimum I/O size
   OPT-IO  optimal I/O size
  PHY-SEC  physical sector size
  LOG-SEC  logical sector size
     ROTA  rotational device
    SCHED  I/O scheduler name
  RQ-SIZE  request queue size
     TYPE  device type
 DISC-ALN  discard alignment offset
DISC-GRAN  discard granularity
 DISC-MAX  discard max bytes
DISC-ZERO  discard zeroes data
```

And final blkid prints the block device (partitions and storage media) attributes like uuid and file system type. It does not report the space on the partitions.

```
$ sudo blkid
/dev/mmcblk0: PTUUID="872806b7" PTTYPE="dos"
/dev/mmcblk0p1: SEC_TYPE="msdos" LABEL="boot" UUID="B176-EFEE" TYPE="vfat" PARTUUID="872806b7-
01"
/dev/mmcblk0p2: UUID="0aed834e-8c8f-412d-a276-a265dc676112" TYPE="ext4" PARTUUID="872806b7-02"
```

From device to partition to filesystem to directory

Let's insert a memory USB stick (= device) of 8GB into the Pi.

First thing is to find the device name using `hwinfo`

```
$ sudo hwinfo --block --short
disk:
  /dev/ram11           Disk
  /dev/ram2            Disk
  /dev/ram0            Disk
  /dev/ram9            Disk
  /dev/ram7            Disk
  /dev/ram14           Disk
  /dev/ram5            Disk
  /dev/ram12           Disk
  /dev/ram3            Disk
  /dev/ram10           Disk
  /dev/ram1            Disk
  /dev/ram8            Disk
  /dev/ram15           Disk
  /dev/ram6            Disk
  /dev/mmcblk0         Disk
  /dev/ram13           Disk
  /dev/ram4            Disk
  /dev/sda             Generic STORAGE DEVICE <-- added device = USB memory stick
partition:
  /dev/mmcblk0p1       Partition
  /dev/mmcblk0p2       Partition
  /dev/sda1            Partition
  /dev/sda2            Partition
```

So we see that the memory USB stick is disk recognised as `/dev/sda` and it has already 2 partitions on it `/dev/sda1` and `/dev/sda2`

Next is to (re)partition the device. Should the device have been used before, you see existing partitions and you might want to erase (aka zero-ing) the device first using the following command:

```
sudo dd if=/dev/zero of=/dev/sda bs=4k && sync
```

This will take some long time. It will pretend to stuck. Just be patient. And the bigger the memory stick, the longer you should wait. The slower the USB stick, the longer you should wait. FYI: it took over 10 minutes to format 8GB.

```
$ sudo dd if=/dev/zero of=/dev/sda bs=4k && sync
dd: error writing '/dev/sda': No space left on device
3889537+0 records in
3889536+0 records out
15931539456 bytes (16 GB) copied, 1493.04 s, 10.7 MB/s
```

Make a new partition table on the device:

```
sudo fdisk /dev/sda
```

`fdisk` is interactive and will notice that the "device does not contain a recognized partition table". The first thing you have to do is create one. An MBR formatted device uses a DOS partition table, and the `fdisk` command to create that is `o`.

```
Command (m for help): o
Created a new DOS disklabel with disk identifier 0xbb8861c3.
```

You can see the empty table with the command `p`.

```
Disk /dev/sda: 14.9 GiB, 15931539456 bytes, 31116288 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
Disklabel type: dos
Disk identifier: 0xbb8861c3
```

We now need to create a new partition with `n`.

```
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-31116287, default 2048): 8192
```

Note I did not use the default value. 8192 is what's used in the actual Raspbian images and seems to be a common practice with SD cards.
I make only 1 partition on this device taking up the whole device. So I used the default value.

```
Last sector, +sectors or +size{K,M,G,T,P} (8192-31116287, default 31116287):

Created a new partition 1 of type 'Linux' and of size 14.9 GiB.
```

By default the type of the partition is "Linux" = EXT3 . We can to change that.

```
Command (m for help): t

Selected partition 1
Hex code (type L to list all codes): c

Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.
```

That's all we need here, but to hedge against mistakes, `fdisk` doesn't really do anything until you press `w`. At that point it will write out the new table and exit.

Check out what has been created using `hwinfo` (a reboot might be required first)

```
hwinfo --block --short
disk:
  /dev/ram11           Disk
  /dev/ram2            Disk
  /dev/ram0            Disk
  /dev/ram9            Disk
  /dev/ram7            Disk
  /dev/ram14           Disk
  /dev/ram5            Disk
  /dev/ram12           Disk
  /dev/ram3            Disk
  /dev/ram10           Disk
  /dev/ram1            Disk
  /dev/sda             Generic STORAGE DEVICE
  /dev/ram8            Disk
  /dev/ram15           Disk
  /dev/ram6            Disk
  /dev/mmcblk0         Disk
  /dev/ram13           Disk
  /dev/ram4            Disk
partition:
  /dev/sda1            Partition
  /dev/mmcblk0p1       Partition
  /dev/mmcblk0p2       Partition
```

The next step is to put a filesystem on the partition by formatting the partition.
Raspbian allows you to format any supported filesystem format using the `mkfs` tool.
As we marked our partition as FAT32, we format the partition using FAT32.
To format a drive to FAT32:

```
sudo mkfs.vfat /dev/sda1 -n disklabel
```

In the example you will notice an option `-n` followed by `disklabel`. This is an optional volume label to name your drive.

Finally we have to mount the filesystem. The filesystem is given a directory where you are able to access and modify its content. These directories are known as mount points and can be given any name that works for you but they should be placed in /mnt. I will call my mount point 'usbdrive'. First we need to create a mount point.

```
sudo mkdir /mnt/usbdrive
```

To mount the filesystem to your mount point 'usbdrive'.

```
sudo mount /dev/sda1 /mnt/usbdrive
```

Access to the USB stick is just like reading from and writing to the directory `/mnt/usbdrive`

Notes:

```
Hex code (type L to list all codes): L

 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden C:  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi eb  BeOS fs
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         ee  GPT
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ef  EFI (FAT-12/16/
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        f0  Linux/PA-RISC b
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f1  SpeedStor
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f4  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f2  DOS secondary
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      fb  VMware VMFS
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fc  VMware VMKCORE
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fd  Linux raid auto
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fe  LANstep
1c  Hidden W95 FAT3 75  PC/IX           be  Solaris boot    ff  BBT
1e  Hidden W95 FAT1 80  Old Minix
```

Unmount before you remove

It is always advisable that you unmount a USB drive before unplugging it from its power source. This forces all queued data to be written to the drive before it loses power.

```
sudo umount /dev/sda1
```

You may need to use the `-f` force option if the drive will not dismount.

```
sudo umount -f /dev/sda1
```

If you use the shutdown `-P -h 0` command to power down your Pi you do not need to use unmount.

Another way to safely remove a USB drive is to eject first

```
sudo eject /dev/sda1
```

USB Power Problems

A major limitation for running a USB drive on a Raspberry Pi are the power requirements. The Universal Serial Bus specification states that to adhere to the standard, up to 0.5A can be drawn from a single port.

My WD Elements 1TB USB hard drive requires 0.9A to work which is 0.4A above the USB2 specification. This is not an isolated example, many modern desktop PCs and laptops supply a greater amperage than the standard 0.5A to their USB ports to support devices such as portable USB hard drives. Unfortunately the Pi cannot power many external USD devices such as hard drives as its USB ports are restricted to the standard amperage.

To get around this problem you need a powered USB hub. You attach the hub device to the Pi's USB port, insert the hub's power supply into a walled power socket and plug the USB hard drive into one of the hub's USB ports.

I would also recommend against powering the Raspberry Pi off the same USB Hub as a USB drive.

Creating a bootable SD card for your Raspberry Pi

Here I describe how to create a device image with 2 filesystem partitions in it, but those partitions will be empty. You then need to copy the contents of each partition onto it (cfr Backup & Restore Your Pi). This is the most flexible method.

There are two partitions in the Raspbian image, the first small boot partition and the second larger root filesystem. There is also a partition table, aka. the master boot record (MBR), since this is a device image. The MBR is critical and occupies the first 512 bytes in the image.

Insert the SD card into a USB SD card holder and insert this into a free USB port of your Pi. I used a 8GB SD card.

First thing is to find the disk/device name using `hwinfo`

```
$ sudo hwinfo --block --short
disk:
  /dev/ram11           Disk
  /dev/ram2            Disk
  /dev/ram0            Disk
  /dev/ram9            Disk
  /dev/ram7            Disk
  /dev/ram14           Disk
  /dev/ram5            Disk
  /dev/ram12           Disk
  /dev/ram3            Disk
  /dev/ram10           Disk
  /dev/ram1            Disk
  /dev/ram8            Disk
  /dev/ram15           Disk
  /dev/ram6            Disk
  /dev/mmcblk0         Disk
  /dev/ram13           Disk
  /dev/ram4            Disk
  /dev/sda             Generic STORAGE DEVICE <-- added device = USB memory stick
partition:
  /dev/mmcblk0p1       Partition
  /dev/mmcblk0p2       Partition
  /dev/sda1            Partition
```

So we see that the memory USB stick is disk /dev/sda.

Next is to partition the device. Should the device have been used before, you might see existing partitions and you might want to erase (aka zero-ing) the device first using the following command:

```
sudo dd if=/dev/zero of=/dev/sda bs=4k && sync
```

This will take some long time. It will pretend to stuck. Just be patient. And the bigger the memory stick, the longer you should wait. The slower the USB stick, the longer you should wait. FYI: it took over 10 minutes to format 8GB.

```
$ sudo dd if=/dev/zero of=/dev/sda bs=4k && sync
dd: error writing '/dev/sda': No space left on device

1984257+0 records in
1984256+0 records out
8127512576 bytes (8.1 GB) copied, 1236.37 s, 6.6 MB/s
```

Make a new partition table in the device:

```
sudo fdisk /dev/sda
```

`fdisk` is interactive and will note that the "device does not contain a recognized partition table". The first thing you have to do is create one. An MBR formatted device uses a DOS partition table, and the `fdisk` command to create that is `o`.

```
Command (m for help): o
Created a new DOS disklabel with disk identifier 0x4fb6d463.
```

You can see the empty table with p. We now need to create the two new partitions with `n`.

```
Command (m for help): n

Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-31116287, default 2048): 8192
Last sector, +sectors or +size{K,M,G,T,P} (8192-31116287, default 31116287): 137216

Created a new partition 1 of type 'Linux' and of size 63 MiB.
```

Note I did not use the default value. 8192 is what's used in the actual Raspbian images and seems to be a common practice with SD cards. In the Raspbian image this partition is about 63 MB. Or in number of sectors: 66.060288 (= 63 * 1025 * 1024) / 512 = 129024 and so ends at sector 137216 when starting at 8192 (= 129024 + 8192).

By default the type of the partition is "Linux". We need to change that.

```
Command (m for help): t

Selected partition 1
Hex code (type L to list all codes): c

Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.
```

Let's do a quick check using the `p` command

```
Command (m for help): p
Disk /dev/sda: 14.9 GiB, 15931539456 bytes, 31116288 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
Disklabel type: dos
Disk identifier: 0x4fb6d463

Device     Boot Start    End Sectors Size Id Type
/dev/sda1        8192 137216  129025  63M  c W95 FAT32 (LBA)
```

Make this partition bootable

```
Command (m for help): a
Partition number (1, default 1): 1

The bootable flag on partition 1 is enabled now.

Command (m for help): p
Device     Boot  Start      End  Sectors  Size Id Type
/dev/sda1  *      8192   137216   129025   63M  c W95 FAT32 (LBA)
```

Now the second partition.

```
Command (m for help): n

Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (2-4, default 2): 2
First sector (2048-31116287, default 2048): 137217
Last sector, +sectors or +size{K,M,G,T,P} (137217-31116287, default
31116287):

Created a new partition 2 of type 'Linux' and of size 14.8 GiB.
```

Notice again I didn't use the default because we left empty space at the beginning, `fdisk` still wants to use it. Instead I used the sector number after the last sector of the first partition, which you can find in the partition table (it's the `End`).

For the last sector I chose the highest possible value (`default 31116287`) to fill the entire image file.

Once that's created the table looks like this:

```
Device     Boot  Start      End  Sectors  Size Id Type
/dev/sda1  *      8192   137216   129025   63M  c W95 FAT32 (LBA)
/dev/sda2       137217 31116287 30979071 14.8G 83 Linux
```

That's all we need here, but to hedge against mistakes, fdisk doesn't really do anything until you press `w`. At that point it will write out the new table and exit. A reboot might be required. To check:

```
sudo fdisk -l
...
Disk /dev/sda: 14.9 GiB, 15931539456 bytes, 31116288 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
Disklabel type: dos
Disk identifier: 0x4fb6d463

Device     Boot  Start      End  Sectors  Size Id Type
/dev/sda1  *      8192   137216   129025   63M  c W95 FAT32 (LBA)
/dev/sda2       137217 31116287 30979071 14.8G 83 Linux
```

Those offset `-o` values are the `Start` numbers from the partition table, `8192` and `28672`, times the unit size (`512`). Now we have device partitions that can be formatted:

```
sudo mkfs.vfat -F32 /dev/sda1
mkfs.fat 3.0.27 (2014-11-12)

sudo mkfs.ext4 /dev/sda2
mke2fs 1.43.3 (04-Sep-2016)
Creating filesystem with 3872383 4k blocks and 969136 inodes
Filesystem UUID: 9caf2207-5fe8-4f3e-abd5-8406ca999ec9
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

The second command will produce more output but there should not be any errors reported. We can now check the filesystems. First create a couple of directories and mount the image partitions there:

```
mkdir /mnt/usb1
mkdir /mnt/usb2
sudo mount /dev/sda1 /mnt/usb1
sudo mount /dev/sda2 /mnt/usb2
```

This should not report any errors. `one` will be empty, and `two` should have a `lost+found` directory in it.

```
ls /mnt/usb1
ls /mnt/usb2
lost+found
```

You can now copy whatever you want into these mount points, e.g., you could duplicate a Pi using rsync. (Cfr: Backup & Restore Your Pi).

Some quick commands to 'copy'

```
sudo rsync -axHv --delete-during /boot/ /mnt/usb1/
sudo rsync -axHv --delete-during /      /mnt/usb2/
```

Notice that there is a leading and trailing slash

When you're done, umount the partitions

```
sudo umount /dev/sda1
sudo umount /dev/sda2
```

and you remove USB stick with SD card in it. Remove SD card and put it into another Pi to boot from it.

Resize filesystem using parted & resize2fs

You can't shrink or expand a device but you can shrink or expand an individual partition and filesystem on it with `parted` and `resize2fs`.

Use these steps to expand your partition and file system on a Raspberry Pi. As example, I'll use the standard SD card on a Pi.

- backup your system in case of a misstake!
- use *"fdisk /dev/mmcblk0"* to view your partitions.
- use *"parted"* to delete the partition and then recreate it but with a larger size. (don't worry, the data will remain)
- reboot to activate the partition changes.
- use *"resize2fs /dev/mmclk0p2"* to enlarge the root file system.
- use *e2fsck -f /dev/mmcblk0p2* to perform a file system check.
- use "df -h" to check results.

Before you extend your root partition and filesystem you should know how big your rootfs is and how much space is available:

Determine the storage devices and active partitions on it:

```
Ls -l /dev/mm*
brw-rw---- 1 root disk 179, 0 Jun  3 13:22 /dev/mmcblk0
brw-rw---- 1 root disk 179, 1 Jun  3 13:21 /dev/mmcblk0p1
brw-rw---- 1 root disk 179, 2 Jun  3 13:21 /dev/mmcblk0p2

df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       1.6G  1.5G   53M  97% /
/dev/mmcblk0p1   50M   18M   33M  35% /boot
```

Check the partition table:

```
fdisk /dev/mmcblk0
...
Command (m for help): p

Disk /dev/mmcblk0: 16.0 GB, 16012804096 bytes, 31275008 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: 0x000622ba

        Device Boot      Start         End      Blocks   Id  System
/dev/mmcblk0p1   *        2048      104447       51200    c  W95 FAT32 (LBA)
/dev/mmcblk0p2          104448     3494304     1694928+  83  Linux

Command (m for help): q
```

So the SD card has 31275008 (16GB) sectors and the last one in use is 3494304 (1.6GB). Print the partition table with *"parted"*:

```
parted /dev/mmcblk0
GNU Parted 3.1
Using /dev/mmcblk0
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) unit chs
(parted) print
Model: SD  (sd/mmc)
Disk /dev/mmcblk0: 1946,198,43
```

```
Sector size (logical/physical): 512B/512B
BIOS cylinder,head,sector geometry: 1946,255,63.  Each cylinder is 8225kB.
Partition Table: msdos
Disk Flags:

Number  Start     End         Type     File system  Flags
 1      0,32,32   6,127,56    primary  fat16        boot, lba
 2      6,127,57  217,130,9   primary  ext4
(parted)
```

So the disk ends at 1946,198,43 cylinder,head,sector and the current root partition ends at 217,130,9.

Note: *"fdisk"* displays the partition info in 512 bytes blocks and *"parted"* displays the cylinder,head,sector geometry. Each cylinder is 82.25kB.

Now remove the second partition and recreate it larger.

Note: If you have a third swap or other partition that you don't need any longer, you can remove that one first and use the freed disk space to extend.

Removing the partition will only change the partition table and not the data. Creating a new partition will write a new start and end point in the partition table.

Be careful: If you make a misstake, you lose you root partition data:
(Ignore the warning.)

```
(parted) rm 2
Error: Partition(s) 2 on /dev/mmcblk0 have been written, but we have been
unable to inform the kernel of the change, probably because it/they are in
use.  As a result, the old partition(s) will
remain in use.  You should reboot now before making further changes.
Ignore/Cancel? i
(parted)
```

And check whether the partition was removed:

```
(parted) print
Model: SD  (sd/mmc)
Disk /dev/mmcblk0: 1946,198,43
Sector size (logical/physical): 512B/512B
BIOS cylinder,head,sector geometry: 1946,255,63.  Each cylinder is 8225kB.
Partition Table: msdos
Disk Flags:

Number  Start    End       Type     File system  Flags
 1      0,32,32  6,127,56  primary  fat16        boot, lba

(parted)
```

Now the second partition is removed. Do not reboot your system before you have created the new partition! Other wise you lose your root file system.

The new partition must start at the same position where the old root partition did start and it ends where you like. It must have at least the same size as current partition and it may not exceed the end of the disk (in my case 1946,198,43).
(Ignore the warning.)

```
(parted) mkpart primary 6,127,57  1946,198,43
Error: Partition(s) 2 on /dev/mmcblk0 have been written, but we have been
unable to inform the kernel of the change, probably because it/they are in
use.  As a result, the old partition(s) will
```

```
remain in use.  You should reboot now before making further changes.
Ignore/Cancel? i
(parted)
```

And check whether the partition was created:

```
(parted) print
Model: SD  (sd/mmc)
Disk /dev/mmcblk0: 1946,198,43
Sector size (logical/physical): 512B/512B
BIOS cylinder,head,sector geometry: 1946,255,63.  Each cylinder is 8225kB.
Partition Table: msdos
Disk Flags:

Number  Start     End          Type     File system  Flags
 1      0,32,32   6,127,56     primary  fat16        boot, lba
 2      6,127,57  1946,198,43  primary  ext4

(parted) quit
Information: You may need to update /etc/fstab.
```

Be carefull: The kernel is not aware yet of the new partition size. You must reboot your system before you do any thing else.

```
reboot
```

Check the new partition size after the reboot:

```
fdisk /dev/mmcblk0
Welcome to fdisk (util-linux 2.22.1).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): p

Disk /dev/mmcblk0: 16.0 GB, 16012804096 bytes, 31275008 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: 0x000622ba

        Device Boot      Start         End      Blocks   Id  System
/dev/mmcblk0p1   *        2048      104447       51200    c  W95 FAT32 (LBA)
/dev/mmcblk0p2          104448    31275007    15585280   83  Linux

Command (m for help): quit
```

Now the partition is larger, but the root file system has still the old size. Re-size the root filesystem:

```
resize2fs /dev/mmcblk0p2
resize2fs 1.42.3 (14-May-2012)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p2 is now 3896320 blocks long.
```

The root file system is now extended. Before rebooting, we want to make sure a file system check for errors is executed at (every) reboot. In order to get this done, issue the command:

```
sudo tune2fs -c 1 /dev/mmcblk0p2
tune2fs 1.43.3 (04-Sep-2016)
Setting maximal mount count to 1
```

Now reboot

```
sudo reboot
```

Once rebooted, do a final check of the file systems size and the available space:

```
df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        15G  1.5G   13G  11% /
/dev/mmcblk0p1   50M   18M   33M  35% /boot
```

It has lots of free space available and it is ready to use.
And do this verification as well:

```
sudo fdisk /dev/mmcblk0

Command (m for help): p
Disk /dev/mmcblk0: 7.4 GiB, 7946108928 bytes, 15519744 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
Disklabel type: dos
Disk identifier: 0x6f92008e

Device         Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk0p1 *      8192   131071   122880   60M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      131072  2658303  2527232  1.2G 83 Linux
```

Connect NAS or Windows share automatically at start-up

As good as the Pi is, the capacity of an SD card isn't going to go far. So connecting to trusty NAS drive (e.g. shared folder) can be a neccessity and automatically mount it when the Pi starts. Samba is needed to connect to a NAS.

1. **Install/Update Samba**
   On Raspian, Samba is already installed (Note: smbfs has been depreciated in wheezy.)
   Note: on Raspbmc, Samba is already installed but `cifs-utils` need to b installed in order to mount the NAS drive.

```
sudo apt-get install cifs-utils
```

2. **Make a share & directory on your NAS**
   Create a SMB share on your NAS and create a directory in it, if you need one

```
//NAS/nasShare/nasDir
```

3. **Make a directory to mount your NAS to**

```
cd /mnt
mkdir NAS
cd NAS
mkdir Share
```

4. **Edit fstab file**
   Edit the fstab file to mount the NAS drive at startup:

```
sudo nano /etc/fstab
```

   Add the following line to the bottom of the file:

```
//NAS/nasShare/nasDir /mnt/NAS/Share cifs
username=your_username,password=your_password,workgroup=your_workgroup,us
ers,auto,user_xattr 0 0
```

   and saved the changes.

WARNING : this will mean your username & password is stored in plain text viewable to all on the device, if this is going to be a problem you can use a credentials file, see http://anothersysadmin.wordpress.com/2007/12/17/howto-mount-samba-shares-in-fstab-using-a-credential-file/

5. **Test**
   Test the change by mounting the NAS drive using the command:

```
sudo mount -a
```

   Then navigate to the mount directory and retreive a directory listing from the NAS:

```
cd /mnt/NAS/Share
ls
```

   Create a file on your NAS in the directory of the share. Then do again a `ls` on your Pi and the newly created file must be listed.

6. **Reboot**
   To be double sure, reboot and make sure your NAS is connected:

```
sudo shutdown -r now
cd /mnt/NAS/Share
ls
```

Appendix

File Systems Compatibility

EXT has native support in Raspbian and the Raspberry Pi. It is turned on by default on the Raspberry Pi.

HFS+ has restricted support in Raspbian. It can read HFS+ formatted partitions but can only write to them if journaling is disabled. To enable Raspbian HFS+ support:

```
sudo apt-get install hfsutils hfsprogs hfsutils
```

FAT is probably the most supported file system but it is also the most limited. Raspbian supports FAT, VFAT and FAT32. To enable Raspbian FAT32 support:

```
sudo apt-get install dosfstools
```

NTFS has read-only support in Raspbian. To enable Raspbian NTFS support:

```
sudo apt-get install ntfs-3g
```

Raspbian allows you to format any supported fie system format using the `mkfs` tool.

To format a partition to EXT3 (Raspbian):

```
sudo mkfs.ext3 /dev/sda1 -L disklabel
```

To format a partition to EXT4 (Raspbian):

```
sudo mkfs.ext4 /dev/sda1 -L disklabel
```

To format a partition to HFS+ (Mac OS X):

```
sudo mkfs.hfsplus /dev/sda1 -v disklabel
```

To format a partition to FAT32 (DOS and legacy Windows):

```
sudo mkfs.vfat /dev/sda1 -n disklabel
```

To format a partition to NTFS (Windows):

```
sudo mkfs.ntfs /dev/sda1 -f -v -I -L disklabel
```

I have applied a few options here that I will explain.

- `-f` *Fast Format*. Due to the poor performance of 3g.ntfs on the Pi I highly recommend using the less CPU intensive fast format mode.
- -v *Verbose*. By default the NTFS status output is limited so this lets you know what is happening.
- -I *Disable Windows Indexing*. This improves the write performance of the drive but it will mean Windows Search queries used on this drive will take longer.
- -L *Disk Label.* This is an optional volume labels to name your drive.