CHAPTER 9

SD Card Storage

The file system is central to the Unix system design from which Linux borrows. The mass storage requirements have traditionally been fulfilled through hard disk subsystems. However, as hosts become as small as credit cards, flash memory technology has replaced the bulky mechanical drive.

SD Card Media

The first Pi's used a standard-sized SD card. All newer models, however, now use the MicroSD card shown in Figure 9-1, along with the standard SD adapter.

Figure 9-1. *SD MicroSD adapter (left) and 8 GB MicroSD card at right*

W. Gay, *Advanced Raspberry Pi*, https://doi.org/10.1007/978-1-4842-3948-3_9

The 8 pins of the underside of the MicroSD are shown in Figure 9-2.

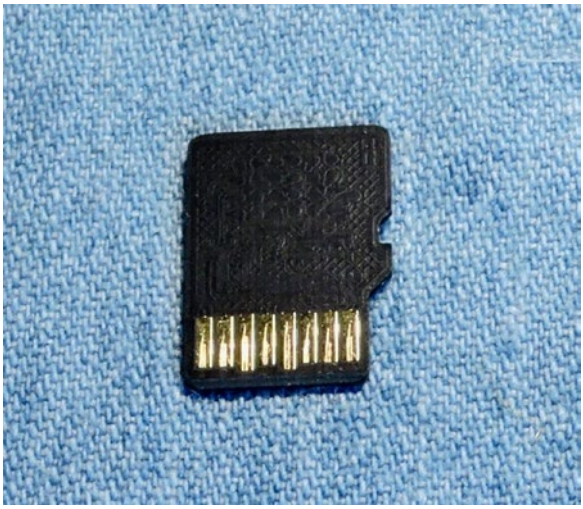

Figure 9-2. *The underside of the MicroSD card, with its 8 pins exposed*

SD Card Basics

The SD card includes an internal controller, also known as a Flash Storage Processor (FSP). In this configuration, the host merely provides a command and waits for the response. The FSP takes care of all erase, programming, and read operations necessary to complete the command. In this way, Flash card designs are permitted to increase in complexity as new performance and storage densities are implemented.

The SD card manages data with a sector size of 512 bytes. This was intentionally made the same as the IDE magnetic disk drive for compatibility with existing operating systems. Commands issued by the host include a sector address to allow read/writes of one or more sectors.

Note Operating systems may use a multiple of the 512-byte sector.

Commands and data are protected by a CRC (cyclic redundancy check) code generated by the FSP. The FSP also automatically performs a read after write to verify that the data was written correctly.[9] If the write was defective, the FSP automatically corrects it, replacing the physical sector with another if necessary.

The SD card soft error rate is much lower than a magnetic disk drive. In the rare case when errors are discovered, the last line of defense is a correcting ECC (error correction code), which allows for data recovery. These errors are corrected in the media to prevent future unrecoverable errors. All of this activity is transparent to the host.

Raspbian Block Size

The block size used by the operating system may be a multiple of the media's sector size. To determine the physical block size used under Raspbian, we first discover how the root file system is mounted (the following listing has been trimmed):

```
$ mount
/dev/mmcblk0p2 on / type ext4 (rw,noatime,data=ordered)
...
/dev/mmcblk0p1 on /boot type vfat (rw,relatime,fmask=0022,dma
sk=0022, \
      codepage=437,iocharset=ascii,shortname=mixed,
      errors=remount-ro)
```

From this we deduce that the device used for the root file system is `/dev/mmcblk0p2`. The naming convention used tells us the following:

Component	Name	Number	Type
Prefix	/dev/mmcblk		MMC block
Device number	0	0	
Partition number	p2	2	

From the earlier `mount` command output, notice that the `/boot` file system was mounted on `/dev/mmcblk0p1`. This indicates that the `/boot` file system is from partition 1 of the same SD card device.

Using the device information, we consult the `/sys` pseudo file system to find out the physical and logical sector sizes. Here we supply `mmcblk0` as the third-level pathname qualifier to query the device:

```
$ cat /sys/block/mmcblk0/queue/physical_block_size
  512
$ cat /sys/block/mmcblk0/queue/logical_block_size
  512
$
```

The result shown informs us that the Raspbian Linux used in this example uses a block (sector) size of 512 bytes, both physically and logically. This precisely matches the SD card's sector size.

Disk Cache

While we're examining mounted SD card file systems, let's also check the type of device node used:

```
$ ls -l /dev/mmcblk0p?
brw-rw---- 1 root disk 179, 1 Jun 19 07:42 /dev/mmcblk0p1
brw-rw---- 1 root disk 179, 2 Jun 19 07:42 /dev/mmcblk0p2
```

The example output displays a `b` at the beginning of the brw-rw---- field. This tells us that the disk device is a *block* device as opposed to a *character* device. (The associated character device would show a `c` instead.) Block devices are important for file systems because they provide a disk cache capability to vastly improve the file system performance. The output shows that both the `root` (partition 2) and the `/boot` (partition 1) file systems are mounted using block devices.

Capacities and Performance

SD cards allow a configurable data bus width within limits of the media. All SD cards start with one data bit line until the capabilities of the memory card are known. After the capabilities of the media are known, the data bus can be expanded under software control, as supported. Table 9-1 summarizes SD card capabilities.[10]

Table 9-1. *SD Card Capabilities*

Standard	Description	Greater Than	Up To	Data Bus
SDSC	Standard capacity	0	2 GB	1-bit
SDHC	High capacity	2 GB	32 GB	4-bit
SDXC	Extended capacity	32 GB	2 TB	4-bit

Transfer Modes

There are three basic data transfer modes used by SD cards:

- SPI Bus mode
- 1-bit SD mode
- 4-bit SD mode

SPI Bus Mode

The SPI Bus mode is used mainly by consumer electronics using small microcontrollers supporting the SPI bus. Examining Table 9-2 reveals that data is transmitted 1 bit at a time in this mode (pin 2 or 7).

Table 9-2. *MicroSD SPI Bus Mode*

Pin	Name	I/O	Logic	Description	SPI
1	NC				
2	/CS	I	PP	Card select (active low)	CS
3	DI	I	PP	Data in	MOSI
4	VDD	S	S	Power	
5	CLK	I	PP	Clock	SCLK
6	VSS	S	S	Ground	
7	DO	O	PP	Data out	MISO
8				Reserved	

The various SD card connections are used in different ways, as documented by the Table 9-2 mnemonics in the columns I/O and Logic. Table 9-3 is a legend for these and also applies to Table 9-4.

Table 9-3. *Legend for I/O and Logic*

Notation	Meaning	Notes
I	Input	Relative to card
O	Output	
I/O	Input or output	
PP	Push/pull logic	
OD	Open drain	
S	Power supply	
NC	Not connected	Or logic high

SD Mode

SD mode allows for varying data bus width for added I/O rates supported by SDHC and SDXC cards. Higher data clock rates also improve transfer rates. Table 9-4 lists the pin assignments.

Table 9-4. *Micro SD Mode Pins*

Pin	Name	I/O	Logic	Description
1	DAT2	I/O	PP	Data 2
2	CD/DAT3	I/O	PP	Card Detect/Data 3
3	CMD	I/O	PP/OD	Command/response
4	VDD	S	S	Power
5	CLK	I	PP	Clock
6	VSS	S	S	Ground
7	DAT0	I/O	PP	Data 0
8	DAT1	I/O	PP	Data 1

Wear Leveling

Unfortunately, Flash memory is subject to *wear* for each *write* operation performed (as each write requires erasing and programming a block of data). The design of Flash memory requires that a large block of memory be erased and rewritten, even if a single sector has changed value. For this reason, wear leveling is used as a technique to extend the life of the media. Wear leveling extends life by moving data to different physical blocks while retaining the same logical address.

Whether or not a given memory card supports wear leveling is an open question without supporting documentation. Some manufacturers may not implement wear leveling at all or use a lower level

of overprovisioning. Wear leveling is not specified in the SD card standard, so no manufacturer is compelled to follow SanDisk's lead.

Direct File System Mounts

There are times when it is convenient to make changes to a SD card file system with the Pi offline, using Linux. If you have an USB card adapter, this could also be done by a different Pi. Using the SD card slot or a SD card reader attached to your Linux box, you can mount the file systems directly.

But how do you know what to mount? There are at least two helpful commands you can apply:

- lsblk
- blkid

The `lsblk` command is great for showing you the block devices and the partition arrangements:

```
# lsblk
NAME     MAJ:MIN   RM     SIZE   RO   TYPE MOUNTPOINT
sda        8:0      0   149.1G    0   disk
├─sda1     8:1      0   147.3G    0   part /
├─sda2     8:2      0       1K    0   part
└─sda5     8:5      0     1.8G    0   part [SWAP]
sdb        8:16     1    14.5G    0   disk
├─sdb1     8:17     1    41.8M    0   part
└─sdb2     8:18     1    14.5G    0   part
sr0       11:0      1    1024M    0   rom
#
```

From that display, you can see the Linux root file system is mounted from /dev/sda1. Our SD card appears on /dev/sdb, with partitions sdb1 and sdb2. The blkid command gives us some more information, including the partition labels:

```
# blkid
/dev/sda1: UUID="51d355c1-2fe1-4f0e-aaae-01d526bb27b5"
TYPE="ext4" PARTUUID="61c63d91-01"
/dev/sda5: UUID="83a322e3-11fe-4a25-bd6c-b877ab0321f9"
TYPE="swap" PARTUUID="61c63d91-05"
/dev/sdb1: LABEL="boot" UUID="A75B-DC79" TYPE="vfat"
PARTUUID="2e37b5e0-01"
/dev/sdb2: LABEL="rootfs" UUID="485ec5bf-9c78-45a6-9314-
32be1d0dea38" TYPE="ext4" \
           PARTUUID="2e37b5e0-02"
```

This display shows that our Pi /boot partition is on /dev/sdb1, while its root partition is available on /dev/sdb2. These can be mounted directly. First make sure you have directory entries to mount them on (if they don't exist already):

```
# mkdir /mnt/1
# mkdir /mnt/2
```

Now they can be mounted:

```
# mount /dev/sdb1 /mnt/1
# mount /dev/sdb2 /mnt/2
```

Once mounted like this, you can list or change files at will in directories /mnt/1 or /mnt/2.

Read-Only Problem

What do you do when Linux complains that your SD card is read-only?

```
# mount /dev/sdb1 -o rw /mnt/1
mount: /dev/sdb1 is write-protected, mounting read-only
```

This problem can stem from at least three possible sources:

- The switch on the MicroSD adapter has slipped to the "Protect" (or locked) position.
- Linux has a software lock on the device.
- Or the MicroSD adapter is faulty (bad connection).

MicroSD Adapter Switch

The slide switch for the MicroSD adapter can be a real nuisance as it accidentally gets slid into the "locked" position. The solution is to pull it back out and fix the switch setting.

Software Protection

Another possibility is that Linux has a software lock on the device. The `-r1` option turns this feature on:

```
# hdparm -r1 /dev/sdb1

/dev/sdb1:
 setting readonly to 1 (on)
 readonly      =  1 (on)
# mount /dev/sdb1 /mnt/1
mount: /dev/sdb1 is write-protected, mounting read-only
#
```

The `hdparm` command using the `-r1` command can set a software lock for the device. Attempting to mount the file system with this lock enabled results in a read-only mount. The solution to this is to disable this protection using option `-r0`:

```
# hdparm -r0 /dev/sdb1
```

```
/dev/sdb1:
 setting readonly to 0 (off)
 readonly      =  0 (off)
# mount /dev/sdb1 /mnt/1
```

MicroSD Adapter Quality

When this read-only issue happened to me for the first time, I thought the issue was with the hardware in the Linux computer I was using. Researching this I discovered that some people had reported that their MicroSD adapter was the problem. After trying three different MicroSD adapters, I was eventually successful. The first two adapters failed with a bad connection making the device read-only.

If you got an adapter with your MicroSD card, that is probably the adapter you want to use. However, that may not guarantee success.

Image Files

If you lack a way to mount the SD card directly, then you can still manipulate the image file. This might be the downloaded Raspbian image or perhaps a friend was able to get it to you somehow. Perhaps they created the image from the SD card from their computer:

```
# dd if=/dev/sdb of=/tmp/sdcard.img bs=1024k
```

That `dd` command copies the input disk (`/dev/sdb`) to an output file (`/tmp/sdcard.img`), using a block size of 1 MB (1024k). The large block size is used for greater efficiency.

The problem with the image file is that it contains two partitions. If it were a single partition, then it could be mounted directly. The partitions require us to do a bit more work. Installing `kpartx` will make this task easier:

```
# apt-get install kpartx
```

Now when we have an image file to mount we can use it as follows:

```
# kpartx -v -a /tmp/sdcard.img
add map loop0p1 (254:0): 0 85611 linear /dev/loop0 8192
add map loop0p2 (254:1): 0 30351360 linear /dev/loop0 98304
```

Notice the names `loop0p1` and `loop0p2`? Use them to mount the file system partitions in the image file:

```
# mount /dev/mapper/loop0p1 /mnt/1
# mount /dev/mapper/loop0p2 /mnt/2
```

Now you will find your `/boot` files in `/mnt/1` and the Pi root partition mounted at `/mnt/2`. When you are done with making your changes, unmount the partitions:

```
# umount /mnt/1
# umount /mnt/2
```

After unmounting the file systems, you can copy the image file back to your SD card.

Summary

This chapter briefly introduced the SD card and its operation. Then two different ways of working with the SD card file system outside of the Raspberry Pi were examined—mounting the SD card on Linux directly and mounting an image file. Both have their uses, particularly in rescue operations.