

# Move the Raspberry PI root file system to a USB drive - PragmaticLinux

*PragmaticLinux*

A Raspberry PI is a wonderful and powerful little computer. You can even run it as a server 24/7 all year round. There is just one snag: sooner or later the SD card gets corrupted during a write operation. Moving the root file system of your Raspberry PI to an external USB drive bypasses this problem. This article presents instructions on moving the root file system from the SD card to an external USB drive, such that you can boot your Raspberry PI with the root file system on the USB drive.

## Background

Google the phrase “Raspberry PI SD card corruption” and soon you’ll realize that this is a serious concern. Especially if you plan on running your Raspberry PI as a server 24/7. For some their Raspberry PI server runs without problems for over a year, others that are less lucky need to replace the SD-card several time a year.

When does this problem with the SD card occur? Well, only during write operations. So, if we can figure out a way to not write to the SD card, the problem is solved. One solution is by storing the Raspberry PI root file system on a USB drive, instead of the SD card. This article presents clear step-by-step instructions on moving the root file system from your Raspberry PI SD card to an external USB drive. Afterwards you can boot your Raspberry PI with the root file system located on the USB drive.

When should you move the Raspberry PI root file system to a USB drive? Ideally right after you installed the Raspbian operating system and booted your Raspberry PI for the first time with it.

## What you need

I assume that you already own a Raspberry PI, with a micro SD card and a suitable power supply. Furthermore I assume that you already installed [Raspbian](#) (the Raspberry PI operating system based on Debian) on the SD card. In case you haven’t yet installed the operating system, you can follow the instructions from a [previous article](#), covering the steps to setup your Raspberry PI as a headless server.

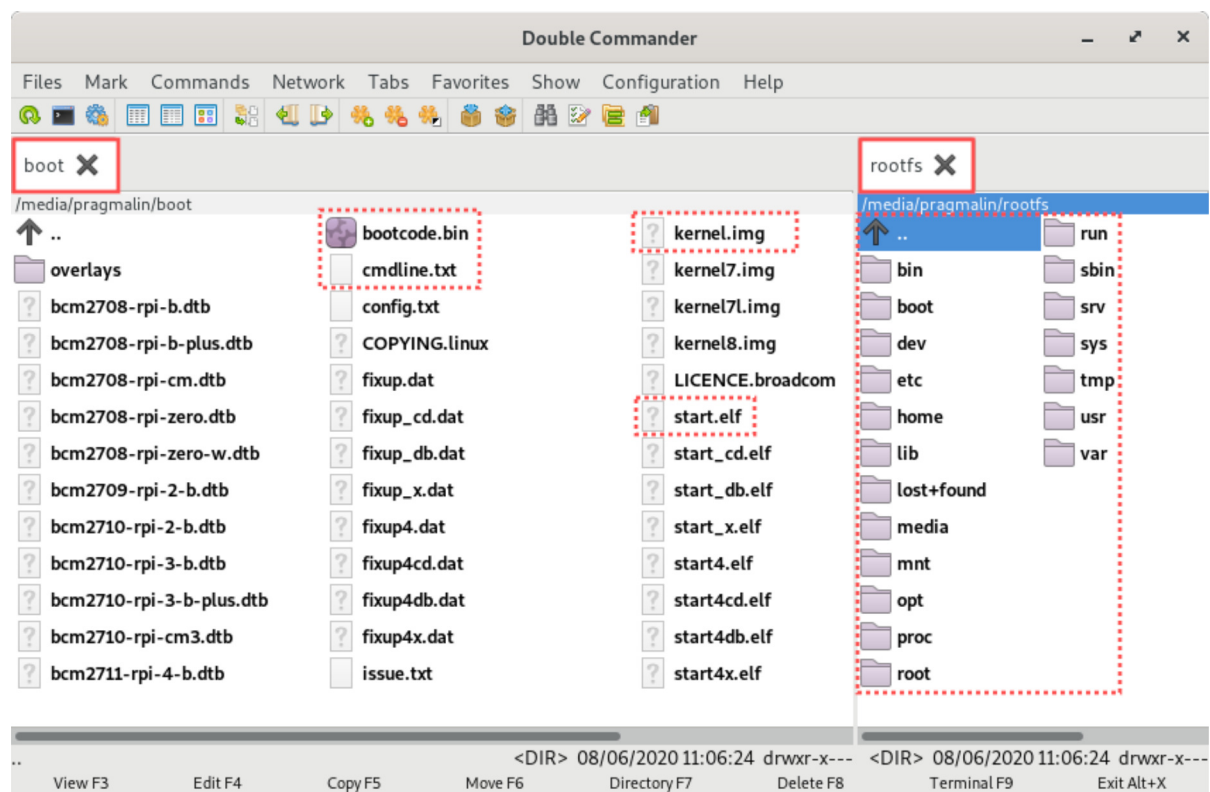
In addition you need the actual USB drive. Ideally a high quality 2.5” SATA SSD drive (such as the Samsung 860 PRO) in combination with a USB 3.0 to 2.5” SATA III adapter. If you first want to try out the steps in this article, before investing into these additional parts, a simple USB flash drive of 8GB or greater suffices for testing purposes.

In general, the procedure outlined in this article works on any Raspberry PI model. Note however that the Raspberry PI 4 features a USB 3.0. The USB 3.0 offers much higher read/write speeds compared to the USB 2.0 found on the older Raspberry PI models. In fact, if you store the root file system on a SATA SSD drive, which is connected to the your Raspberry PI 4 with a USB 3.0 to SATA adapter, it performs even better than with an SD card.

The actual move of the Raspberry PI’s root file system to the USB drive will be done with a Linux based PC. My laptop runs Debian 10, but you can use whatever Linux distribution you prefer for this. This does mean that we’ll connect both the USB drive and the micro SD card of the Raspberry PI to our PC. Therefore you might need to get an adapter for inserting the micro SD card to your PC.

## Overview of Raspbian partitions

To understand why moving the Raspberry PI root file system from the SD card to a USB drive prevents SD card corruption, you need a basic understanding of the Raspbian partitions. I inserted the Raspberry PI SD card into my PC. After this the partitions got automatically mounted in my case. If you do this, you’ll notice that the Raspberry PI operating system occupies two partitions. Below you can see a screenshot from the [Double Commander](#) twin panel file manager that shows both partitions:



## The rootfs partition

On the right side of the previous screenshot you can see the root file system partition. This is the one that gets mounted at / when the Raspberry PI boots up. It is also the one that we will move to the USB drive in this article. It contains all the files that you as a user can see when you logged into your Raspberry PI. Linux does the majority of its read and write operations to this partition. For example, it often writes to log files in /var/log. You as a user also perform write operations to this partition. For example to your /home directory. Due to the write operations to this partition, the chances of SD card corruption increase. Hence the idea to move this entire partition to a USB drive.

## The boot partition

The boot partition contains files needed just for booting up the Linux operating system. The Raspberry PI expects this boot partition on the SD card, otherwise it can't boot. Upon power-up, the CPU starts the actual bootloader, which is located in bootcode.bin. The bootloader loads and starts the firmware program in the start.elf file. This program in turn loads the actual Linux kernel from kernel.img into RAM. It passes the arguments found in cmdline.txt to the kernel and then gives control to the Linux kernel. Once the Linux kernel gets control, it continues with booting up your Raspberry PI operating system.

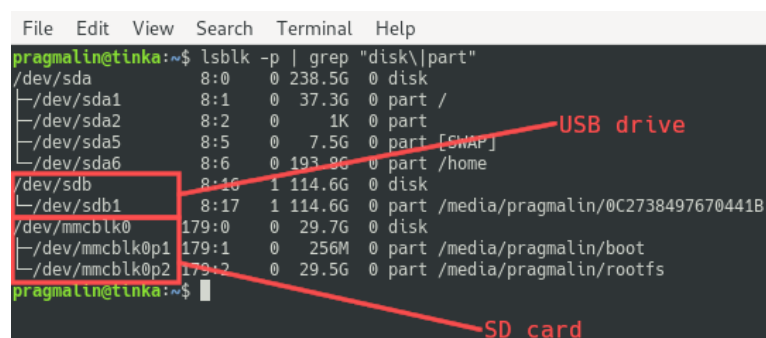
This is probably more information than you really wanted. The important take away here is that only read operations are performed on this partition. Consequently, we do not have to fear SD card corruption on the boot partition. The only time a write operation on this partition is performed, is when a new kernel with security fixes is installed during a system update. This happens only a few times a year and therefore the risk of SD card corruption is negligible.

## Insert the USB drive and SD card into your PC

To move the Raspberry PI root file system to a USB drive, we will use our own Linux PC. You shouldn't attempt to perform these steps directly on the Raspberry PI, because then the root file system is in use. Consequently, copying all files from the root file system might fail.

As a first step, make sure your Raspberry PI is powered down. Remove the SD card from the Raspberry PI and insert it into your PC. Afterwards connect the USB drive to your PC. With both devices connected to your PC, we need to figure out the assigned device names. Run the following command for listing all disks and partitions on your PC. The -p parameter make sure that the command output includes the full device name:

```
lsblk -p | grep "disk\|part"
```



After inspecting the command output we learn that `/dev/sda` is my PC's hard disk and the one we really shouldn't touch at all. Upon insertion, Linux assigned the `/dev/sdb` device name to the USB drive. The USB drive currently contains one partition (`/dev/sdb1`), which doesn't matter because we intend to reformat the USB drive in the next section. Linux assigned the `/dev/mmcblk0` device name to the SD card. As expected, the SD card contains two partitions: the boot partition (`/dev/mmcblk0p1`) and the root file system partition (`/dev/mmcblk0p2`).

For the remainder of the article, I will use these device names in the examples. However note that the device names, which your PC assigned, could be different. Make sure to substitute the device names accordingly in the subsequent commands.

Now that we know the device names, we just need to make sure the partitions are not mounted before continuing. The unmount the partitions, run these commands:

```
sudo umount /dev/sdb1
sudo umount /dev/mmcblk0p1
sudo umount /dev/mmcblk0p2
```

## Create and format an EXT4 partition on USB drive

Before we can copy the files from the SD card's root file system to the USB drive, we need to create one large EXT4 partition on the USB drive and then format this newly created partition. Note that this deletes all files that are currently stored on the USB drive. So make sure to make a backup of these files, in case they are important. Also triple check that you specify the correct device name in the following commands. Otherwise you might irreversibly alter the wrong disk.

Start out by creating a new partition table on the disk:

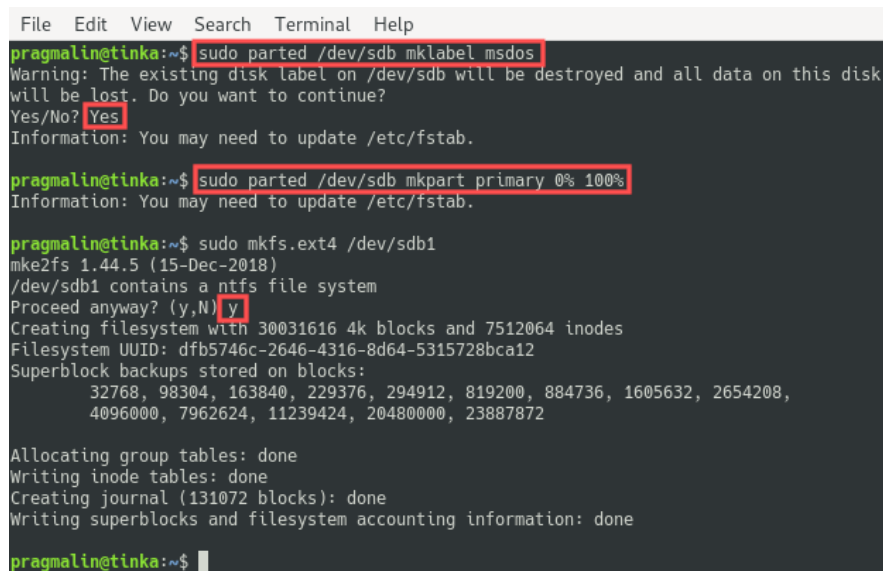
```
sudo parted /dev/sdb mklabel msdos
```

Next create one big partition. This creates `/dev/sdb1`:

```
sudo parted /dev/sdb mkpart primary 0% 100%
```

After the creation of the new `/dev/sdb1` partition, format it as EXT4:

```
sudo mkfs.ext4 /dev/sdb1
```



```
File Edit View Search Terminal Help
pragmalin@tinka:~$ sudo parted /dev/sdb mklabel msdos
Warning: The existing disk label on /dev/sdb will be destroyed and all data on this disk
will be lost. Do you want to continue?
Yes/No? Yes
Information: You may need to update /etc/fstab.

pragmalin@tinka:~$ sudo parted /dev/sdb mkpart primary 0% 100%
Information: You may need to update /etc/fstab.

pragmalin@tinka:~$ sudo mkfs.ext4 /dev/sdb1
mke2fs 1.44.5 (15-Dec-2018)
/dev/sdb1 contains a ntfs file system
Proceed anyway? (y,N) y
Creating filesystem with 30031616 4k blocks and 7512064 inodes
Filesystem UUID: dfb5746c-2646-4316-8d64-5315728bca12
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done
Writing inode tables: done
Creating journal (131072 blocks): done
Writing superblocks and filesystem accounting information: done

pragmalin@tinka:~$
```

## Mount the USB drive and SD card partitions

In the previous sections, we unmounted all partitions of the USB drive and the SD card. Furthermore we created one large EXT4 partition on the USB drive. In this section we create new mount points for all three partitions and manually mount the partitions.

Create the mount points with the following three commands:

```
sudo mkdir -p /mnt/usbdrive
sudo mkdir -p /mnt/sdbboot
sudo mkdir -p /mnt/sdrootfs
```

Mount the partitions by running the commands:

```
sudo mount /dev/sdb1 /mnt/usbdrive
sudo mount /dev/mmcblk0p1 /mnt/sdbboot
sudo mount /dev/mmcblk0p2 /mnt/sdrootfs
```

To verify that all worked as intended, run:

```
lsblk -p | grep "disk\|part"
```

```
File Edit View Search Terminal Help
pragmalin@tinka:~$ sudo mkdir -p /mnt/usbdrive
pragmalin@tinka:~$ sudo mkdir -p /mnt/sdbboot
pragmalin@tinka:~$ sudo mkdir -p /mnt/sdrootfs
pragmalin@tinka:~$ sudo mount /dev/sdb1 /mnt/usbdrive
pragmalin@tinka:~$ sudo mount /dev/mmcblk0p1 /mnt/sdbboot
pragmalin@tinka:~$ sudo mount /dev/mmcblk0p2 /mnt/sdrootfs
pragmalin@tinka:~$ lsblk -p | grep "disk\|part"
/dev/sda      8:0    0 238.5G  0 disk
├─/dev/sda1   8:1    0  37.3G  0 part /
├─/dev/sda2   8:2    0    1K  0 part
├─/dev/sda5   8:5    0   7.5G  0 part [SWAP]
├─/dev/sda6   8:6    0 193.8G  0 part /home
├─/dev/sdb    8:16   1 114.6G  0 disk
├─/dev/sdb1   8:17   1 114.6G  0 part /mnt/usbdrive
├─/dev/mmcblk0 179:0   0  29.7G  0 disk
├─/dev/mmcblk0p1 179:1   0  256M  0 part /mnt/sdbboot
└─/dev/mmcblk0p2 179:2   0  29.5G  0 part /mnt/sdrootfs
pragmalin@tinka:~$
```

## Copy the SD card root file system to the USB drive

Everything is in place to proceed with the actual root file system copying. The goal is to copy the entire root file system from the SD card to the USB drive. We can easily do this with the help of the `rsync` program. Run the following command. Note that it can take a few minutes for the copy operation to complete:

```
sudo rsync -axv /mnt/sdrootfs/* /mnt/usbdrive
```

```
File Edit View Search Terminal Help
pragmalin@tinka:~$ sudo rsync -axv /mnt/sdrootfs/* /mnt/usbdrive
sending incremental file list
bin/
bin/bash
bin/bunzip2
bin/busybox
...
...
var/mail/
var/opt/
var/spool/
var/spool/mail -> ../mail
var/spool/cron/
var/spool/cron/crontabs/
var/spool/rsyslog/
var/tmp/

sent 1,372,618,629 bytes  received 723,825 bytes  33,092,589.25 bytes/sec
total size is 1,369,767,527  speedup is 1.00
pragmalin@tinka:~$
```

## Remap the root partition

With the root file system copied to the USB drive, we need to inform the Raspberry PI operating system to actually use this one and not the one of the SD card. This needs to be done in two locations:

1. In the command line parameters that the bootloader passes to the Linux kernel.
2. In the file system table file (fstab) that informs the Linux kernel how to mount the file systems.

To make these changes, we first need to figure out the PARTUUID of the partition on the USB drive. So the partition that now holds the root file system. The abbreviation PARTUUID stands for universally unique partition identifier. The device name of this partition is `/dev/sdb1`. Run this command to determine the its PARTUUID:

```
sudo blkid | grep "/dev/sdb1"
```

```
File Edit View Search Terminal Help
pragmalin@tinka:~$ sudo blkid | grep "/dev/sdb1"
/dev/sdb1: UUID="dfb5746c-2646-4316-8d64-5315728bca12" TYPE="ext4" PARTUUID="a332dbd2-01"
pragmalin@tinka:~$
```

In my case it is `a332dbd2-01`, but it will be different for you. Therefore make sure to substitute your PARTUUID whenever I use `a332dbd2-01`.

## Update PARTUUID in kernel parameters

The parameters that the bootloader passes to the kernel are located in file `cmdline.txt` on SD card's boot partition. One of the parameters is the PARTUUID of the partition holding the root file system. Let's make a backup copy of this file first:

```
sudo cp /mnt/sdbboot/cmdline.txt /mnt/sdbboot/cmdline.org
```

As a next step, edit this file with the Nano terminal text editor. Refer to [this article](#) for a brief refresher on editing files with Nano. Change the PARTUUID to the one of the USB drive partition once you opened the file in Nano:

```
sudo nano /mnt/sdbboot/cmdline.txt
```

```
File Edit View Search Terminal Help
GNU nano 3.2 /mnt/sdbboot/cmdline.txt Modified
console=serial0,115200 console=tty1 root=PARTUUID=a332dbd2-01 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^M-U Undo ^M-A Mark Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^M-E Redo ^M-G Copy Text
```

## Update PARTUUID in file system table

Once the bootloader gave control to the Linux kernel, the kernel continues booting the operating system. During the boot process, the kernel reads the `fstab` configuration file to determine which partitions need to be integrated into the file system structure and how. This includes the root file system. So we need to change the PARTUUID of the root file system in here, such that the kernel uses the root file system on the USB drive partition.

The `fstab` file is normally located in `/etc/fstab`. Since we mounted the USB drive partition to `/mnt/usbdrive`, the file to edit is `/mnt/usbdrive/etc/fstab`. Let's make a backup copy of this file first:

```
sudo cp /mnt/usbdrive/etc/fstab /mnt/usbdrive/etc/fstab.org
```

As a next step, edit this file with the Nano terminal text editor. Change the PARTUUID to the one of the USB drive partition, once you opened the file in Nano:

```
sudo nano /mnt/usbdrive/etc/fstab
```

```
File Edit View Search Terminal Help
GNU nano 3.2 /mnt/usbdrive/etc/fstab Modified
proc /proc proc defaults 0 0
PARTUUID=738a4d67-01 /boot vfat defaults 0 2
PARTUUID=a332dbd2-01 / ext4 defaults,noatime 0 1
# a swapfile is not a swap partition, no line here
# use dphys-swapfile swap[on|off] for that
```

Look for the line with `/` (which means root file system on Linux) and change the PARTUUID on that line.

## Unmount the partitions and delete the mount points

At this point you made all the necessary changes for moving the Raspberry PI root file system to the USB drive. As a little cleanup, we can unmount all three partitions again. Since we no longer need the mount point directories, we can delete these as well. Start with unmounting the three partitions:

```
sudo umount /mnt/usbdrive
sudo umount /mnt/sdbboot
sudo umount /mnt/sdrootfs
```

Next delete the mount point directories:

```
sudo rmdir /mnt/usbdrive
sudo rmdir /mnt/sdbboot
sudo rmdir /mnt/sdrootfs
```

## Boot the Raspberry PI

As the grand finale, the time has come to see if we can now boot the Raspberry PI with the root file system on the USB drive. Remove both the SD card and the USB drive from your PC. Next, insert them into your Raspberry PI. Once done, connect the power supply to the Raspberry PI and wait a little bit for the boot process to complete.

Once the Raspberry PI completed booting, we can remotely log in to it via SSH. Instead of the default `pi` user, I created a new user called `pragmalin`, so I connect with the command:

```
ssh pragmaline@raspberrypi
```

After logging in via SSH, we can run the following command to verify that the Raspberry PI's root file system is actually on the USB drive:

```
lsblk -p
```

```
File Edit View Search Terminal Help
pragmalin@tinka:~$ ssh pragmalin@raspberrypi
pragmalin@raspberrypi's password:
Linux raspberrypi 4.19.118-v7l+ #1311 SMP Mon Apr 27 14:26:42 BST 2020 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Aug 3 17:02:26 2020 from 2003:e6:870e:1500:e523:fe81:1254:3a96
pragmalin@raspberrypi:~$ lsblk -p
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
/dev/sda                            8:0      1 114.6G  0 disk
└─/dev/sda1                        8:1      1 114.6G  0 part /
/dev/mmcblk0                       179:0    0   29.7G  0 disk
└─/dev/mmcblk0p1                 179:1    0   256M  0 part /boot
└─/dev/mmcblk0p2                 179:2    0   29.5G  0 part

pragmalin@raspberrypi:~$
```

## Wrap up

After going through the steps outlined in this article, you can now run your Raspberry PI with the root file system on a USB drive. Although these steps required a bit of extra time to get your Raspberry PI setup, you pretty much eradicated the SD card corruption problem. This opens the path to using your Raspberry PI as a low cost and energy efficient Linux server system that can run 24/7 all year round.

Keep in mind that the Raspberry PI 4 features USB 3.0. So if you invest in a good 2.5" SATA SSD drive, combined with a USB 3.0 to 2.5" SATA III adapter, you'll actually get better I/O performance compared to having the root file system on the SD card.