1 `-M` = "Shrink the file system to minimize its size as much as possible, given the files stored in the file system." Finally found how to do this! – endolith Jan 17, 2021 at 0:30

11

I originally posted the same answer here, on StackExchange Ask Ubuntu. I re-propose the same answer here, it can be useful.

The key information was the use of the command `truncate`. Following the full solution in order to not lose the answer.

A preliminary step consists in cloning the SD card in your PC:

1. use `lsblk` to see which devices are available and if their partitions are mounted
2. unmount all partitions of the device you want to copy on your pc. For example:

   ```
   umount /dev/sdc1
   umount /dev/sdc2
   ```

3. create a copy of the whole sd card with all the partitions unmounted

   ```
   dd if=/dev/sdc of=/path/to/file/myimage.img
   ```

Shrinking images on Linux

Context of the problem:

Having a `myimage.img` bigger then the hardware support (if it is smaller there should be no problem; however, using the same strategy, you can better fit the image in the hardware support).

The secret is to use standard Linux tools and instruments: GParted, `fdisk` and `truncate`.

Requirements:

- A Linux PC
- The `.img` you want to shrink (`myimage.img` in this example)

Creating loopback device:

GParted is an application typically used to manage partition tables and filesystems. In order to *shrink* the image, GParted is going to be used along the first part of the answer.

> GParted operates on devices, not simple files like images. This is why we first need to create a device for the image. We do this using the loopback-functionality of Linux.

Let's enable enable the loopback:

```
sudo modprobe loop
```

Let's request a new (free) loopback device:

```
sudo losetup -f
```

The command returns the path to a free loopback device:

```
/dev/loop0
```

Let's create a device of the image:

```
sudo losetup /dev/loop0 myimage.img
```

The device `/dev/loop0` represents `myimage.img`. We want to access the partitions that are on the image, so we need to ask the kernel to load those too:

```
sudo partprobe /dev/loop0
```

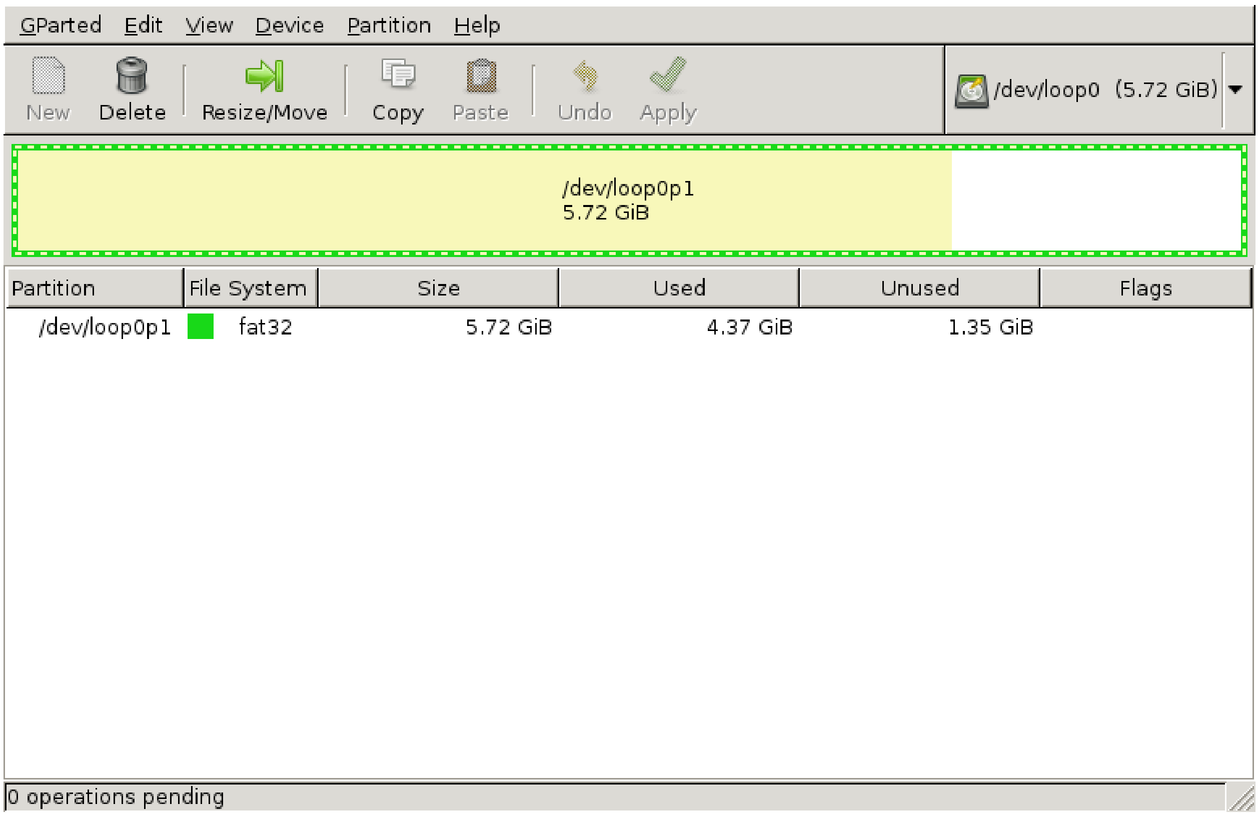
This should give us the device `/dev/loop0p1`, which represents the first partition in `myimage.img`. We do not need this device directly, but GParted requires it.

Resize partition using GParted:

Let's load the new device using GParted:

```
sudo gparted /dev/loop0
```

When the GParted application opens, it should appear a window similar to the following:



Partition	File System	Size	Used	Unused	Flags
/dev/loop0p1	🟩 fat32	5.72 GiB	4.37 GiB	1.35 GiB	

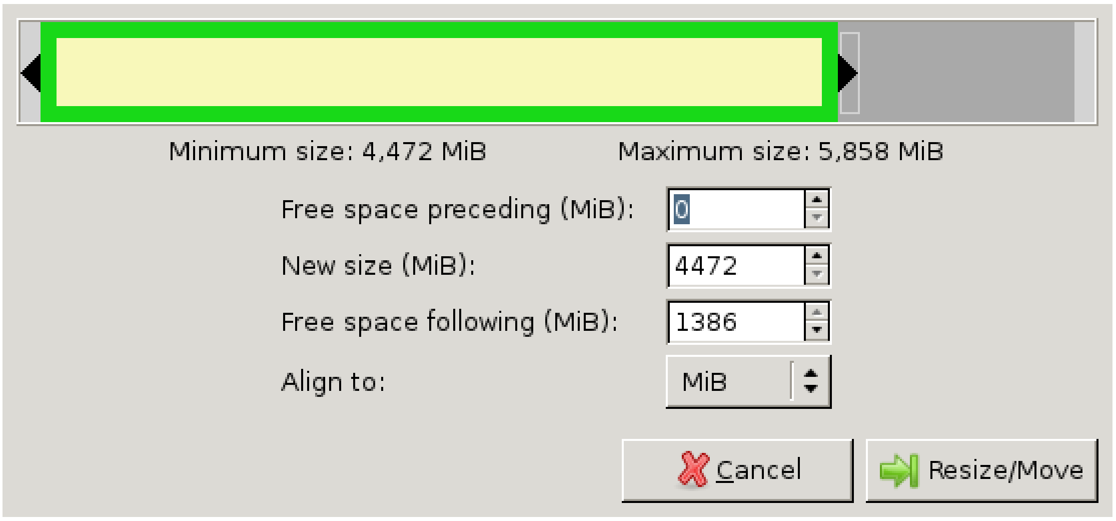
0 operations pending

Now notice a few things:

- There is one partition.
- The partition allocates the entire disk/device/image.
- The partition is filled partly.

We want to resize this partition so that is fits its content, but not more than that.

Select the partition and click Resize/Move. A window similar to the following will pop up:


Minimum size: 4,472 MiB Maximum size: 5,858 MiB

Free space preceding (MiB): 0

New size (MiB): 4472

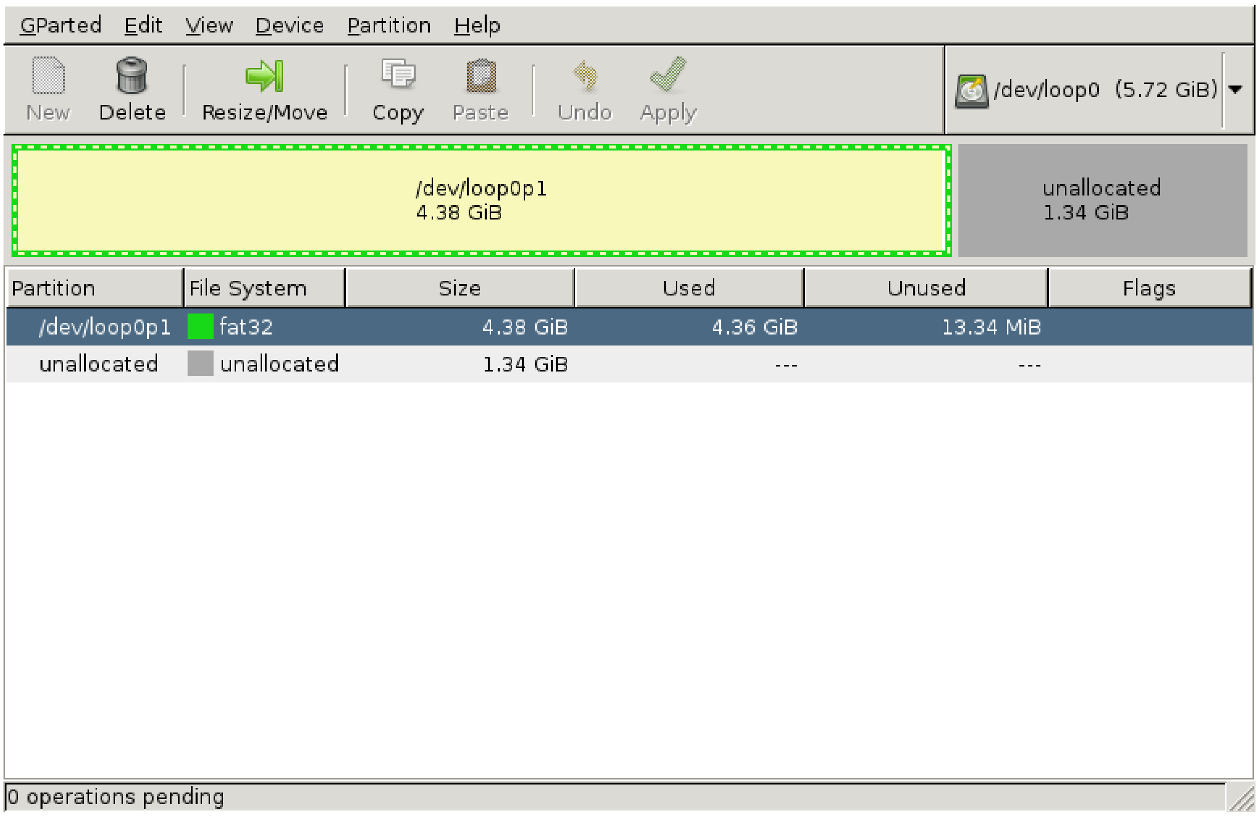
Free space following (MiB): 1386

Align to: MiB

Drag the right bar to the left as much as possible.

Note that sometimes GParted will need a few MB extra to place some filesystem-related data. You can press the up-arrow at the New size-box a few times to do so. For example, I pressed it 10 times (=10MiB) for FAT32 to work. For NTFS you might not need to at all.

Finally press Resize/Move. You will return to the GParted window. This time it will look similar to the following:


GParted Edit View Device Partition Help
New Delete Resize/Move Copy Paste Undo Apply
/dev/loop0 (5.72 GiB)
/dev/loop0p1
4.38 GiB
unallocated
1.34 GiB

Partition	File System	Size	Used	Unused	Flags
/dev/loop0p1	🟩 fat32	4.38 GiB	4.36 GiB	13.34 MiB	
unallocated	▨ unallocated	1.34 GiB	---	---	

0 operations pending

Notice that there is a part of the disk unallocated. This part of the disk will not be used by the partition, so we can shave this part off of the image later. GParted is a tool for disks, so it doesn't shrink images, only partitions, we have to do the shrinking of the image ourselves.

Press Apply in GParted. It will now move files and finally shrink the partition, so it can take a minute or two, but most of the time it finishes quickly. Afterwards close GParted.

Now we don't need the loopback-device anymore, so unload it:

```
sudo losetup -d /dev/loop0
```

Shaving the image:

Now that we have all the important data at the beginning of the image it is time to shave off that unallocated part. We will first need to know where our partition ends and where the unallocated part begins. We do this using `fdisk`:

```
fdisk -l myimage.img
```

Here we will see an output similar to the following:

```
Disk myimage.img: 6144 MB, 6144000000 bytes, 12000000 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: 0x000ea37d
```

```
      Device Boot      Start         End      Blocks   Id  System
myimage.img1            2048     9181183     4589568    b  W95 FAT32
```

Note two things in the output:

- The partition ends on block 9181183 (shown under `End`)
- The block-size is 512 bytes (shown as sectors of `1 * 512`)

We will use these numbers in the rest of the example. The block-size (512) is often the same, but the ending block (9181183) will differ for you. The numbers mean that the partition ends on byte 9181183*512 of the file. After that byte comes the unallocated-part. Only the first 9181183*512 bytes will be useful for our image.

Next we shrink the image-file to a size that can just contain the partition. For this we will use the `truncate` command (thanks uggla!). With the truncate command need to supply the size of the file in bytes. The last block was 9181183 and block-numbers start at 0. That means we need (9181183+1)*512 bytes. This is important, else the partition will not fit the image. So now we use truncate with the calculations:

```
truncate --size=$[(9181183+1)*512] myimage.img
```

Share Improve this answer Follow

answered Sep 8, 2020 at 8:10
Leos313
231 ● 2 ● 6

RE: `$[` expression in the `truncate` invocation: unix.stackexchange.com/questions/209833 – genpfault Apr 5, 2021 at 22:37

6

I also tried it with **qemu-img**, and it worked like a charm:

```
qemu-img resize test.img 2G
```

We are resizing the `test.img` to make it **2G** (2GB).

Worked flawless for me.

Share Improve this answer Follow

answered Jun 4, 2018 at 16:42
phrogg
215 ● 3 ● 10

1 Awesome, worked flawless for me too, compressing a Raspbian image. fully working! – ffleandro Mar 19, 2019 at 14:48

Well, everything is working, however after booting, running `df -h` shows the original size. How is this possible? – ffleandro Mar 19, 2019 at 15:06