

# Raspberry Pi mit Read-Only-Filesystem optimieren - Ideal für den portablen Einsatz

Beitrag von „Sys\_RoBOTer“ vom 6. Dezember 2020, 07:00

[Zitat von Hamspirit.de](#)

Das Dateisystem des Raspberry Pi reagiert mitunter empfindlich auf plötzliches Ausschalten ohne vorheriges Runterfahren des Systems. So habe ich schon einige Zeit damit zugebracht das korrupte Dateisystem der SD-Karte an meinem Computer wieder zu bereinigen. Das gelingt zwar meistens, doch ich hatte auch schon Fehler, die nicht mehr korrigiert werden konnten und in einer Neuinstallation endeten.

Fehler am Dateisystem entstehen immer dann, wenn während eines Schreibvorgangs die Spannungsversorgung unterbrochen wird. Um die Gefahr eines korrupten Dateisystems zukünftig zu verringern, habe ich meinen portablen Raspberry Pi Zero mit einem Read-Only Filesystem ausgestattet. In dem später standardmäßig aktiven Lesemodus ist die Gefahr eines Filesystemcrashes wesentlich geringer.

Die Konfiguration des Read-Only-Filesystems ist mit wenigen Handgriffen erledigt und schließt sich quasi an die manuelle Installation meines Multimode-Hotspots an.

## Ballast loswerden

Auch bei der hier verwendeten Lite-Variante von Raspbian wird einiges an unnötigen Programmen und Diensten mitinstalliert. Mit den folgenden Befehlen wird das System aktualisiert und entfernt was nicht benötigt wird.

Code

```
apt-get update
apt-get dist-upgrade
apt-get remove-purge cronolog at triggerhappy phys-swaps fileake-hwclock kamba-common
apt-get autoremove --purge
```

## RAM-Disk als temporäres Dateisystem

Unter Linux (bzw. UNIX) gilt der Grundsatz, dass alles eine Datei ist. Vor diesem Hintergrund ist es auch nicht verwunderlich, dass einige Systemprozesse in bestimmte bestimmte Pfade schreiben müssen. Dies gilt beispielsweise bei Logs, Lockfiles und dem Dienst DHCP. Diese Ausnahmen werden mithilfe eines temporären Dateisystems im Arbeitsspeicher abgebildet. Wichtiger Hinweis an dieser Stelle, da der RAM ein

flüchtiger Speicher ist, stehen diese Dateien nach einem Neustart nicht mehr zur Verfügung. Das ist aber auch nicht sonderlich problematisch, denn die nötigen Systemdateien werden während des Startvorgangs neu erzeugt.

Code

```
rm          -rf          /var/lib/dhcp/          /var/spool          /var/lock
ln          -s          /tmp          /var/lib/dhcp
ln          -s          /tmp          /var/spool
ln          -s          /tmp          /var/lock
mv          /etc/resolv.conf          /tmp/
ln -s /tmp/resolv.conf /etc/resolv.conf
```

## Filesystem-Tabelle anpassen

In der Datei `/etc/fstab` sind einige Änderungen nötig. Einerseits müssen die Mountpoints für `/boot` und `/` auf `ro` (readonly) gestellt werden. Darüber hinaus müssen die Mountpoints für `/var/log`, `/var/tmp` und `/tmp` an das temporäre Dateisystem gebunden werden.

Code

```
proc /proc proc defaults 0 0@@@WCF_PRE_LINEBREAK@@@PARTUUID=d8cc668c-01 /boot vfat ro,defa
```

## Boot-Parameter anpassen

In der Datei `/boot/bootline.txt` müssen die beiden Optionen `fastboot` und `noswap` hinzugefügt werden. Die Option `fastboot` sorgt dafür, dass während des Systemstarts keine Überprüfung des Filesystems erfolgt. Da das Dateisystem zukünftig nur lesbar ist, die Dateisystemfehler aber bei Schreibzugriffen verursacht werden, ist die Überprüfung des Filesystems überflüssig. Die Option `noswap` sorgt dafür, dass Inhalte des Arbeitsspeichers nicht in eine Auslagerungsdatei bzw. Partition geschrieben werden.

Code

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2 rootfstype=ex
```

Sind alle Änderungen gemacht, kann das System neu gestartet werden. Schreibzugriffe werden dann wirkungsvoll unterbunden. Doch was ist, wenn man hier und da Änderungen an Dateien vornehmen möchte oder auch Updates einspielen möchte? Auch dafür gibt es eine Lösung.

## Schreibmodus temporär aktivieren

Wenn Änderungen am System durchgeführt werden sollen, lässt sich das Dateisystem natürlich wieder beschreibbar machen. Das lässt sich mittels des Kommandos `mount`

und folgenden Optionen erledigen.

Code

```
mount -o remount,rw /boot
mount -o remount,rw /boot
```

Bis zum nächsten Neustart ist das Dateisystem wieder schreibbar, um etwaige Änderungen zu verarbeiten. Hat man alle gewünschten Änderungen abgeschlossen, startet man das System neu oder nutzt diese beiden Befehle, um es wieder in den Read-Only-Modus zu versetzen.

Code

```
mount -o remount,ro /boot
mount -o remount,ro /boot
```

Etwas mehr Komfort lässt sich durch eine Anpassung der */etc/bash.bashrc* implementieren. Werden die folgenden Zeilen in */etc/bash.bashrc* eingefügt, dann zeigt die Bash den aktuellen Status des Dateisystems in der Kommandozeile an. Außerdem werden die Befehle für den *remount* über die Aliase *ro* und *rw* abgebildet, damit der Wechsel noch einfacher von statten geht.

Code

```
set_bash_prompt(){
fs_mode=$(mount | sed -n -e "s/^\/dev\/.* on \\/.*(r[w|o]).*/\1/p")
PS1='\[\033[01;32m\]\u@h${fs_mode:+($fs_mode)}\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\] '
}

alias ro='sudo mount -o remount,ro / ; sudo mount -o remount,ro /boot'
alias rw='sudo mount -o remount,rw / ; sudo mount -o remount,rw /boot'

PROMPT_COMMAND=set_bash_prompt
```

## Fazit

Für meinen portablen Raspberry Pi Zero habe ich mit dem Read-Only-Filesystem eine gute Lösung gefunden und wieder etwas über Linux gelernt. Das Dateisystem im Nurlesenmodus war eine Funktion, die ich von Pi-Star her kannte und schätzen gelernt habe. Wer noch mehr über das Read-Only Filesystem erfahren möchte, sollte sich die beiden Links ansehen, die mir bei der Umsetzung geholfen haben.

[Protect your Raspberry PI SD card, use Read-Only filesystem](#)

[Raspbian Lite für den Read-Only-Betrieb konfigurieren](#)

*Dieser Artikel erschien zuerst im März 2019 in der gedruckten Ausgabe unseres Magazins.*

Alles anzeigen

Quelle: <https://www.hamspirit.de/10871...er-den-portablen-einsatz/>