

chroot in Raspberry Pi Architektur (z.B. Pi ARM-Binaries auf Intel/AMD ausführen, kein Scherz!)

Update QEMU:

```
apt-get install qemu qemu-user-static binfmt-support
```

Jetzt das Wurzel-Dateisystem für Raspberry Pi mounten (meistens die 2. Partition auf der SD_Karte):

```
sudo mkdir /pi
sudo mount /dev/sdb2 /pi
```

Nun das statisch gelinkte Binary vom qemu-arm-static (ausführbar für Intel/AMD) in die chroot-Umgebung kopieren:

```
sudo cp /usr/bin/qemu-arm-static /pi/usr/bin
```

(Optional:) Hinzufügen ARM ELF interpreter für das Hostsystem, dann kann man ARM-Binaries auf dem Intel/AMD-System installieren:

```
sudo dpkg --add-architecture armhf
sudo apt-get update
sudo apt-get install libc6:armhf
```

Mount /sys, /dev... ins chroot, damit ps, mount usw. funktionieren:

```
for i in proc sys dev tmp; do mount --bind $i /pi/$i; done
```

/tmp zu mounten ist wichtig, um auch graphische Programme starten zu können, da sich dort der Socket für den X-Server befindet. Die .Xauthority-Datei des angemeldeten Benutzers sollte ins chroot kopiert werden, oder auf dem Hostsystem per `xhost +` das Display für die Benutzung freigegeben werden.

Achtung: Wurde der X-Server nicht mit „-nolisten tcp“ gestartet, kann nach `xhost +` jeder von außen auf das Display zugreifen.

Wichtig: Das Binfmt-Subsystem starten (Erkennen von Binaries aufgrund ihrer Signatur und automatisches Starten der zuständigen Mime-Handler):

```
/etc/init.d/binfmt-support start
```

Und nun ins ARM-Chroot wechseln.

```
sudo chroot /chroot-dir
```

(bei eingeschalteter binfmt-Emulation, sonst:)

```
sudo chroot /chroot-dir /chroot-dir/usr/bin/qemu-arm-static
/bin/bash
```

Jetzt kann im ARM-System „normal“ gearbeitet werden. Alle Ressourcen des Hauptsystems können genutzt werden (Achtung beim Schreiben auf Datenträger!)

Nach Verlassen der chroot-Umgebung die bind-gemounteten Ordner wieder aushängen:

```
for i in proc sys dev tmp; do umount /pi/$i; done
```