

UNIX use

Hans Buchmann FHNW/ISE

27. Februar 2020

Ziel

Entwicklung von Programmen auf dem **BeagleBoneGreen**

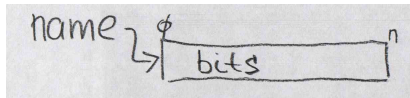
- ▶ alles ist ein File
 - ▶ 0 – te Näherung
 - ▶ File: *stream of bits*
 - ▶ Filesysteme
 - ▶ mount
 - ▶ sshfs
 - ▶ Cross development
 - ▶ *Host* ↔ **BeagleBoneGreen**
- Remark: Keine Toolchain auf dem **BeagleBoneGreen**

Wichtig

- ▶ wo ist was ?
 - ▶ Verzeichnisstruktur
- ▶ wo sind wir ?
 - ▶ *Host*
oder
 - ▶ **BeagleBoneGreen**

Alles ist ein File

stream of bits



- ▶ *name* Referenz auf die Bits (Bytes)
 - ▶ Bits(bytes) der Reihe nach
 - ▶ indexiert $0 \dots n - 1$
- ▶ File
 - ▶ Datenquelle
 - ▶ liefert Daten: Bits(Bytes)
 - ▶ Datensenke
 - ▶ absorbiert Daten: Bits(Bytes)

Ein paar Befehle

- ▶ `cat name`
 - ▶ *concatenate files and print on the standard output*
 - ▶ zeigt den Inhalt
- ▶ `hexdump -C name`
 - ▶ *display file contents in hexadecimal, decimal, octal, or ascii*
 - ▶ zeigt die Bits hexadezimal
- ▶ `dd if=... of=... count=...`
 - ▶ *convert and copy a file*
 - ▶ brauchen wir oft

Devices sind auch Files am Beispiel SD-Karte

`/dev/mmcblki`¹ $i = 0, 1, 2 \dots$

Remark: Name vom Betriebssystem bestimmt

Datenquelle `hexdump -C /dev/mmcblk0`

Datensenke `cp name /dev/mmcblk0`

Remark: Aufpassen

¹als Name ist auch `/dev/sdX` möglich

Devices sind auch Files

z.B Zufallszahlen

`/dev/random` sammelt das Rauschen: langsam

Remark: Name vom Betriebssystem bestimmt

Datenquelle `hexdump -C /dev/random`

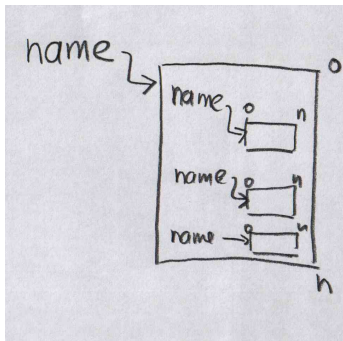
`/dev/urandom` berechnete (Pseudo) Zahlen: schnell

Datenquelle `play -b 16 -e signed-integer \
-t raw -r 44000 /dev/urandom`

Remark: der Befehl `play` hat viele Optionen

Filesystem

Files für Files



► File der weitere Files enthält

► Verschiedene Filesysteme

`vfat` Microsoft

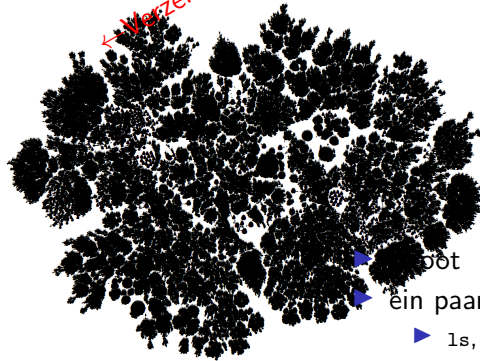
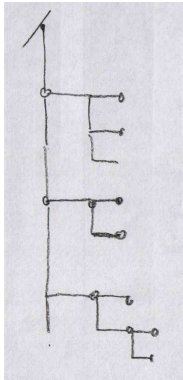
`ext4` UNIX

... noch viele andere

`cat /proc/filesystems`

Vereichnisstruktur

Hierarchie

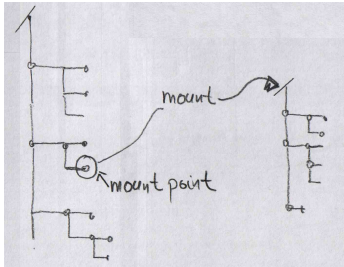


← Verzeichnisse einer Workstation

▶ root
▶ ein paar Befehle
▶ ls, tree, cd

```
mount fileSystem mountPoint
```

Verbindet Filesysteme

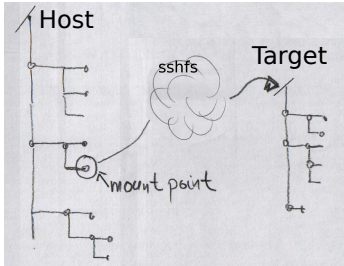


▶ `mount /dev/mmcblk0p1 mountPoint`

▶ `umount mountPoint`

Remark: *mountPoint* sieht wie ein normales Verzeichnis aus

`sshfs user@target:path mountPoint`
via ssh



- ▶ `sshfs user@target:path mountPoint`
 - ▶ braucht ssh
- ▶ `fusermount -u mountPoint`
 - ▶ `umount`

Remark: `mountPoint` Sieht wie ein normales Verzeichnis aus

Crossdevelopment

- ▶ zwei Rechner
 - Host** der Entwicklungsrechner
 - Target** **BBG** der Zielrechner
- ▶ Development
 - ▶ Wo sind die Files
- ▶ CrossDevelopment
 - ▶ Wo sind die Files

Development noch nicht Cross

- ▶ Source file `src/hello-world-c.c`
 - ▶ C
 - ▶ POSIXunabhängig von Plattform
- ▶ Object file (Maschinencode) `hello-world-c.o`
 - ▶ erzeugt mit: `gcc -c ../src/hello-world-c.c -o hello-world-c.o`
 - ▶ Maschinencode:
 - ▶ file `hello-world-c.o`
 - ▶ `objdump -d hello-world-c.o`
- ▶ Executable file `hello-world-c`
 - ▶ `gcc hello-world-c.o -o hello-world-c`
 - ▶ Maschinencode:
 - ▶ file `hello-world-c`
 - ▶ `objdump -d hello-world-c`

Was es braucht ? Files

- ▶ Source file
 - ▶ wo ist der *include file* `stdio.h`
- ▶ Object File
 - ▶ `nm hello-world-c.o`
 - ▶ wo ist `puts`
- ▶ Executable
 - ▶ `nm hello-world-c`
 - ▶ `ldd hello-world-c`
 - ▶ wo sind die Bibliotheken

Wo sind die Files ?

irgendwo in einem Unterverzeichnis von /

- ▶ Include Files `gcc -c -v ../src/hello-world-c.c -o hello-world-c.o`
 - ▶ `stdio.h` ?
 - ▶ `stddef.h` ?
- ▶ Bibliotheken
 - ▶ z.B. `libc.so`

Development der Normalfall

- ▶ Host==Target
- ▶ root Host==root Target

CrossDevelopment

- ▶ immer noch POSIX
- ▶ Host!=Target

Verzeichnisstruktur

Host

```
2-unix-use ..... somewhere on the host
├─ target-root ..... different possibilities
├─ config
│   └─ Makefile for making BeagleBoneGreen executables
├─ src .....c,c++
├─ tc ..... toolchain
└─ work .....connected with BeagleBoneGreen current dir
```

CrossDevelopment

Target BeagleBoneGreen

- ▶ Makefile
 - ▶ `PREFFIX=../tc/bin/arm-linux-gnueabihf-`
- ▶ target-root
 - ▶ `sshfs debian@192.168.7.2:/ target-root`

Verzeichnisstruktur BeagleBoneGreen



Ziel

- ▶ `hello-world-c/hello-world-cpp` auf dem *Host* und auf dem **BBG**
- ▶ `primes` auf dem *Host* und auf dem **BBG**

The big Picture

- ▶ Source File: `hello-world-c.c`
- ▶ falls es nicht klappt ?
 - ▶ wo ist der File ?

Einrichten:Host

Verzeichnisstruktur

2-unix-use

- └─ config git/Makefile
- └─ src git/die source files
- └─ tc selber machen/toolchain link
- └─ work selber machen

Einrichten:BBG

Verzeichnisstruktur

```
/ ..... root
├── home .....root
│   ├── debian ..... das home
│   │   └── work ..... verbunden mit Host via sshfs
```


Verbinden

mount *Host* → **BBG**

```
sshfs debian@192.168.7.2:work work
```

shell *Host* → **BBG**

```
ssh debian@192.168.7.2
```

Development

hello-world-c.c

Host	Target	OS	Toolchain	Verbindung	Bemerk
BBG	BBG	Debian	mitgeliefert		
<i>Host</i>	BBG	Debian	tc-tinl-gcc-9.2.0-2019.10.09.tar.gz	sshfs	
<i>Host</i>	BBG	minimal	tc-tinl-gcc-9.2.0-2019.10.09.tar.gz	SD-Card	später
<i>Host</i>	BBG	minimal	tc-tinl-gcc-9.2.0-2019.10.09.tar.gz	curlftpfs	später

Remark: Toolchain auf der Cloud: **tinL**