

# Ein ganzes GNU/Linux

Hans Buchmann FHNW/IME

25. Mai 2015

## Um was geht es ?

- ▶ ein GNU/Linux von Grund auf bauen
  - ▶ nicht mehr so schwer wie auch schon
- ▶ nicht völlig automatisiert
- ▶ Alternative zu **yocto** & Co.

# Ziel

## GNU/Linux auf dem **RaspberryPi**

- ▶ command based
- ▶ Ethernet
- ▶ ssh
- ▶ sshfs
- ▶ moderne Toolchain inkl. *c++11* **C++**

# Komponenten RaspberryPi und *Host*

## RaspberryPi

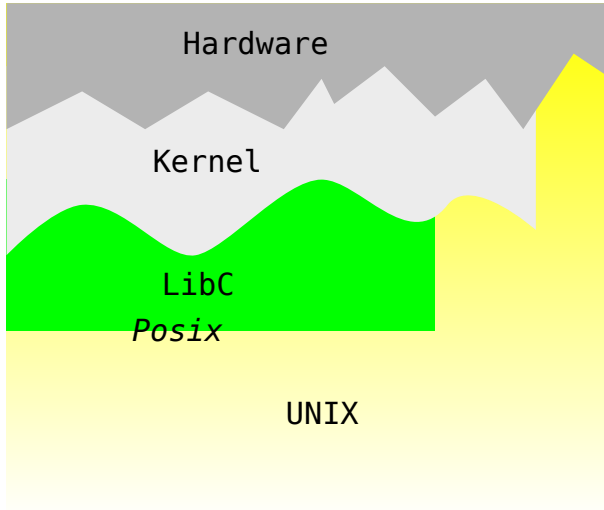
Kernel ein File

root ein Filesystem

## *Host*

Toolchain binutils, gcc

# Übersicht



# Die Komponenten für RaspberryPi

Hardware **RaspberryPi**

Kernel zugeschnitten auf **RaspberryPi**

▶ <https://github.com/raspberrypi/linux.git>

root das Filesystem

LibC eglibc

▶ <http://www.eglibc.org/home>

UNIX busybox

▶ <http://www.busybox.net/>

... Weitere UNIX basierte Komponenten

▶ das configure, make, make  
install Triple

# Toolchain

binutils linker & Co.

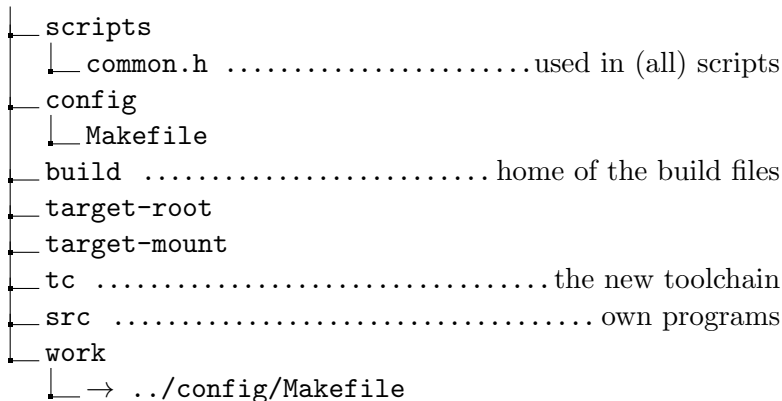
gcc compiler

- ▶ libgcc die Bibliothek für den Compiler

## Remark(s):

- ▶ die Toolchain muss zweimal gebaut werden
  - ▶ für den **kernel** und libc
  - ▶ für UNIX/**POSIX**
- ▶ das target
  - ▶ cpu-vendor-os

# Die Verzeichnisstruktur





# Build

## die Schritte 1

- ▶ binutils
- ▶ gcc-bare
- ▶ kernel mit ein paar *rules*
  - ▶ bcmrpi\_defconfig
  - ▶ zImage
  - ▶ headers\_install
- ▶ eglibc
- ▶ gcc
- ▶ Test
  - ▶ im Verzeichnis work

# Build

## die Schritte 2

- ▶ `busybox`
  - ▶ Installation auf SD-Card
  - ▶ `fakeroot`
- ▶ `openssh` die volle Implementation
  - ▶ `zlib`
  - ▶ `openssl`
  - ▶ `openssh`
- ▶ `sshfs`

# Skripts und Argumente

## initiales System

binutils.sh

gcc-bare.sh

kernel.sh

bcmrpi\_defconfig

zImage

headers\_install

eglibc.sh

gcc.sh

busybox.sh

busybox

install

target-root.sh

kernel libc

C/C++ POSIX

# Target

## erster Versuch

- ▶ transfer auf SD Karte
- ▶ Internet

# Skripts und Argumente

ssh

`zlib.sh`

`openssl.sh`      die kryptographischen Algorithmen

`openssh.sh`

**Remark:** `openssh.sh` hängt von `zlib.sh` und `openssl.sh` ab

# Target

## ssh

- ▶ ssh
- ▶ sshd
- ▶ Keys

sshfs

funktioniert noch nicht

- ▶ Die Bibliothek `glib`
- ▶ Ersatz
  - ▶ `sftp`

# Modules

- ▶ siehe 6-modules



# Modules

## Verzeichnisstruktur

### Host

```
BUILD_HOME
├── modules
│   ├── Makefile ..... for own modules
│   └── *.c/h ..... the own sources
└── scripts
    └── module.sh
```

### RaspberryPi

```
/
└── work
```

# Herstellung Workflow

## Host

- ▶ `sh scripts/module.sh module`
- ▶ `scp, sftp, ftp`

## RaspberryPi

- ▶ `insmod`
- ▶ `rmmmod`

# Teil I

## Installation TFT Display

# Um was geht es ?

## Adafruit TFT Touchscreen

- ▶ Installation auf einem minimalen System
- ▶ git update
- ▶ Konfiguration
- ▶ Kernelcompilation
- ▶ Driver
- ▶ Module
- ▶ *framebuffer*

- ▶ <https://learn.adafruit.com/adafruit-2-8-pitft-capacitive-touch/downloads>
- ▶ Zwei Chips
  - ▶ STMPE610 für den *touchscreen*
  - ▶ ILI9341 für die Graphik
- ▶ Verbunden per SPI

- ▶ [http://adafruit-download.s3.amazonaws.com/2015-02-16-raspbian-pitft28r\\_150312.zip](http://adafruit-download.s3.amazonaws.com/2015-02-16-raspbian-pitft28r_150312.zip)
- ▶ Verbindung per ssh
- ▶ Host dhcp
- ▶ Framebuffer /dev/fb\*
- ▶ Backlight Control

# Kernel

neue Version

- ▶ `git pull`
- ▶ `drivers/staging/fbtft/fb_ili9341.c`