Errata zum Buch

Embedded Linux

Von Raf Jesse
ISBN 978-3-95845-061-5

ISBN: 978-3-95845-061-5

Stand: 19. April 2016

Trotz aller Bemühungen kann es zu Fehlern oder Unklarheiten im Text kommen. Aufmerksame Leser, wie Herr Glembotzki, haben mich bereits auf einige Fehler hingewiesen bzw. Verbesserungsvorschläge gemacht: Vielen Dank für Ihre Mühe! In einer neuen Auflage werden Fehler natürlich korrigiert. Leser der aktuellen Auflage sollen an den Erkenntnissen aber ebenfalls teilhaben können: Für sie wurde diese Rubrik eingeführt.

Wenn Ihnen ebenfalls Fehler auffallen oder Sie ergänzende Informationen benötigen, schreiben Sie mir einfach eine Email an die Adresse

embedded@ralf-jesse.de

Ich werde Ihnen so schnell wie möglich antworten. Versprochen! Wenn Sie es wünschen, werde ich Sie namentlich (aber ohne Angabe der Email-Adresse, unter der Sie mich kontaktieren) erwähnen. Beachten Sie hierzu bitte auch die **Datenschutzerklärung** auf meiner Website http://www.ralf-jesse.de. Ihre ausdrückliche Erlaubnis zur Veröffentlichung Ihres Namens ist zwingend erforderlich. Alle Emails werden persönlich von mir beantwortet, Fehlerkorrekturen werden zusätzlich auf dieser Website veröffentlicht.

Darüber hinaus bitte ich Sie, bei Angaben zu Fehlern das Kapitel und die Seitenzahl anzugeben. Ich bedanke mich bereits jetzt für Ihre Unterstützung.

Allgemeine Hinweise und Ergänzungen

Bei den folgenden Punkten handelt es sich nicht um Fehler im herkömmlichen Sinn: Es handelt sich hierbei um allgemeine Erläuterungen und Ergänzungen, die im Buch etwas zu kurz gekommen sind.

Umbrüche in URLs (Verschiedene Kapitel)

Sie werden sehr häufig URLs finden, die derart lang sind, dass sie auf zwei Zeilen umgebrochen werden mussten. Diese Umbrüche werden durch das Satzprogramm verursacht. Als Autor habe ich hierauf keinen Einfluss. Wenn Sie beispielsweise auf Seite 136 in der URL den Umbruch "***cross-tool***-ng" sehen, so heißt die URL eigentlich "**crosstool**-ng" (Der zweite Bindestrich ist aber korrekt).

Warum kein Dual-Boot-System

Hierbei handelt es sich nicht um einen Fehler. Betrachten Sie diesen Eintrag bitte nur als ergänzende Erläuterung.

Diese Frage wurde mir von einem interessierten Leser gestellt. Man könnte z.B. auch ein Dual-Boot-System verwenden und dann in jedem der beiden Systeme VirtualBox (oder eine andere virtuelle Maschine) installieren. Der Grund für meine Vorgehensweise liegt ausschließlich – in meiner persönlichen Vorliebe. Darüber hinaus habe ich aber auch keine Lust, mir zwei Windows-Lizenzen zu kaufen, nur damit ich eine davon in einer virtuellen Maschine installiere. So verwende ich als Haupt-Arbeitssystem ein Microsoft Windows 7 (Home Premium) und installiere hier VirtualBox von der Firma Oracle. In der virtuellen Maschine arbeite ich dann mit Linux Mint. Sie können hier natürlich eine andere individuelle Konfiguration verwenden.

Kapitel 1

Ergänzungen und Korrekturen zu Kapitel 1.

SoC (Seite 21)

Die ausführliche Bezeichnung des Begriffes *SoC* kenne ich als *Silicon on a Chip*. In der Literatur findet man hierfür häufig auch *System on a Chip*. Beide Bezeichnungen sind zulässig: Lassen Sie sich hier also nicht verwirren. Falsch ist hingegen die Langform *Silicium on a Chip*.

Zur Erheiterung: Angeblich wurde Ray Noorda, der ehemalige Chef der Netzwerkfirma Novell einmal gefragt, wie die ausgeschriebene Form von PCMCIA lautet. Seine Antwort soll gelautet haben: "Ganz klar, PCMCIA steht für *People Can't Memorize Computer Industries Abbreviations*."

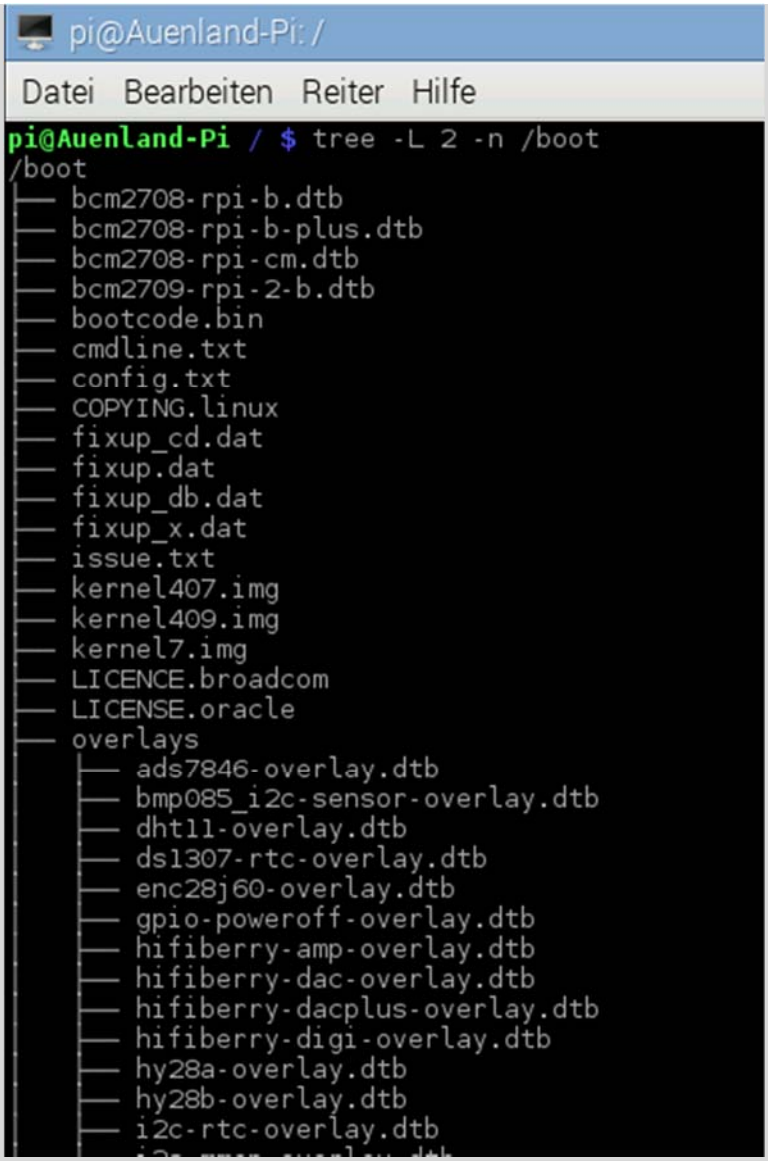
Dem kann ich mich nur anschließen ;-)

Kapitel 2

Ergänzungen und Korrekturen zu Kapitel 2.

Ausgabe von tree -L 2 -n /boot (Seite 62)

Abbildung 2.16 zeigt nicht die Ausgabe von **tree -L 2 -n /boot** sondern die von **ls /boot**. Die korrekte Abbildung sieht folgendermaßen aus:



Korrekte Version: tree -L 2 -n /boot

Das Kommado less (Seite 63)

Es fehlt die Erklärung, wofür das Kommando **less** (Kapitel 2) verwendet wird. **less** ist eine andere Form des Kommandos **more**, das eine seitenweise Ausgabe einer Datei auf dem Bildschirm ausgibt. Mit **more** kann (konnte früher) der Dateiinhalt nur in Vorwärtsrichtung ausgegeben werden. Hatte man einmal zuviel die Leertaste (Space) gedrückt, so war man an der interessierenden Stelle vorbei und konnte nicht zurück. Das Kommando **less** ermöglicht hingegen auch das Rückwärtsrollen eines Dateiinhaltes.

Kapitel 4

Ergänzungen und Korrekturen zu Kapitel 4.

sudo apt-get install Code::Blocks (Seite 116)

Das funktioniert so natürlich nicht: Hier muss **sudo install apt-get codeblocks** stehen. Ich hatte den Fehler beim Korrekturlesen zwar selber schon bemerkt, dann aber offensichtlich vergessen, ihn auch zu beheben.

crosstool-ng vs. buildroot (Seite 134ff.)

Hier habe ich nicht deutlich genug hervorgehoben, dass zwischen **crosstool-ng** und **buildroot** Welten liegen. crosstool-ng wird verwendet, um eine Cross-Toolchain zusammenzustellen. Die Konfiguration dieses Tools führt nur zu einer Zusammenstellung einer individuellen Cross-Toolchain (also Compiler, Linker, Debugger, etc.). **crosstool-ng** "baut" dabei keine neue Toolchain, sondern sorgt nur dafür, eine neue Toolchain aus bereits existierenden Tools zu erzeugen.

Bei **buildroot** hingegen handelt es sich um ein Tool, mit dem ein individueller Kernel oder ein individuelles root-Dateisystem erzeugt werden kann.

Kapitel 5

Ergänzungen und Korrekturen zu Kapitel 5.

git und die Option --depth 1 (Seite 148)

Bei der ersten Beschreibung von **git** wird die Option **--depth 1** verwendet, ohne ihre Funktion zu erläutern. Dies wird zwar später nachgeholt, ich halte es aber für besser – genau wie der Leser, der mich hierauf aufmerksam machte –, die erforderlichen Erklärungen dort anzubringen, wo sie erstmalig verwendet werden.

Die Option **--depth 1** sorgt dafür, dass nur die zum Zeitpunkt der Anwendung aktuellste Version eines Softwarepakets aus dem Repository heruntergeladen wird. Bei sehr großen Paketen, wie dem Sourcecode des Linux-Kernels, wird der Download erheblich schneller vonstattengehen verglichen mit dem Verzicht auf diese Option.

.config-Datei nicht vorhanden (Seite 154)

Seit der Kernelversion 4.0 wird die Datei **.config** nicht mehr unmittelbar installiert. Zum Erzeugen von **.config** muss zunächst das Kommando **sudo modprobe configs** ausgeführt werden. Im Anschluss ist **.config.gz** wieder im Verzeichnis **/proc** zu finden und kann auf die im Buch beschriebene Weise für die Erzeugung individueller Kernel verwendet werden.

Der Grund für diese neue Vorgehensweise ist, dass das Erzeugen eines neuen Kernels vermutlich weniger häufig durchgeführt wird als ursprünglich von den Kernel-Entwicklern angenommen. Durch die neue Vorgehensweise kann der Kernel kleiner ausfallen.

imagetool-uncompressed.py (Seite 163)

Der Hinweis auf das Script **imagetool-uncompressed.py** wird erst 14 Seiten später, also auf Seite 163, relevant. Ein Leser merkte an, dass das Script **mkrpi** korrekt funktioniert, er aber einen Hinweis auf das erwähnte Script vermisse. Mein Fehler an der Stelle: Es wäre besser gewesen, bei der Erklärung der Funktionsweise von **mkrpi** noch einmal auf diesen Hinweis auf Seite 149 zu verweisen.

Das Script **imagetool-uncompressed.py** erzeugt aus der Datei **zImage** das Kernel-Image. Dies war aber nur bis zur Version **Wheezy** von Raspbian notwendig. Mit der Einführung der neuen Version **Jessie** muss **zImage** nicht mehr entpackt werden: Das Entpacken erfolgt jetzt automatisch während des Startvorgangs von Raspbian.

Häufige Eingabe des Passwortes erforderlich (Seite 167)

Mir wurde ebenfalls gemeldet, dass beim Einsatz von **mkrpi** sehr häufig das Passwort des Zielsystems eingegeben werden muss. Dies kommt durch den (automatisierten) Kopiervorgang des Kernel-Images und des Modularchivs. Hier wird das Kommando **scp** (*secure copy*) verwendet, das sämtliche Sicherheitsfunktionen von **ssh** (*secure shell*) verwendet. Dieses Verhalten ist normal, da **scp** immer prüft, ob Sie überhaupt berechtigt sind, Dateien auf das Zielsystem zu übertragen.

Dass Sie hierzu berechtigt sind, weisen Sie durch die Eingabe des Benutzernamens und des Passwortes des Zielsystems nach. Werden der Benutzername oder das Passwort falsch eingegeben, so wird die Dateiübertragung verweigert. Zudem wird beim Aufruf von **scp** ein Timeout gestartet, innerhalb dessen die beiden Eingaben erfolgt sein müssen. Der Timeout läuft ab und verursacht eine Fehlermeldung, wenn innerhalb dieser Zeit die Eingaben nicht korrekt erfolgt sind.

Um diesen Vorgang zu optimieren, enthält das Buch in *Anhang D* eine Kurzanleitung zu **scp**. Wenn Sie dieser Kurzanleitung folgen, erübrigt sich die Eingabe von Benutzernamen und Passwort. Leider habe ich es versäumt, bei der Beschreibung des Scripts **mkrpi** auf diesen Anhang zu verweisen.

Fehlermeldung beim Entpacken von modules.tgz (Seite 175)

Beim Entpacken von **modules.tgz** mit **tar czf modules.tgz *** wird der Inhalt des Archivs in das Verzeichnis **lib** kopiert. Dies funktioniert aber nur, wenn dieses Verzeichnis bereits existiert. Anderenfalls wird das Script mit einer Fehlermeldung beendet. **mkrpi** sollte prüfen, ob das genannte Verzeichnis existiert. Ist dies nicht der Fall, so muss dieses Verzeichnis mit den korrekten Berechtigungsflags (Besitzer = root, Gruppe = root) erzeugt werden.

Bei der Entwicklung des Scripts ist mir dieses Verhalten nicht aufgefallen: Ich kann aber nicht ausschließen, dass ich bei den Tests übersehen habe, dass das Verzeichnis auf meinem Entwicklungs-PC bereits vorhanden war. Eine korrigierte Version des Scripts wird nachgeliefert.

Tipp für die ganz Eiligen:

Fügen Sie vor dem Entpacken den Test

```
if ![-d lib]; then

   mkdir lib

   ...

fi
```

Kapitel 6

Ergänzungen und Korrekturen zu Kapitel 6.

Erklärung von EOF (Seite 196)

Unter embedded@ralf-jesse.de wurde ich gefragt, wo das Kommando EOF beschrieben wird.

EOF ist kein Befehl oder Kommando! Es steht für *End Of File* und markiert das Ende einer Datei. Hierbei handelt es sich um ein Makro (eine Konstante), die in den verschiedenen Compilern unterschiedlich definiert sein kann.

Nach meiner Rückfrage an den Leser stellte sich heraus, dass hiermit die folgenden Zeilen gemeint waren:

```
sh -c 'cat >> /root/.bashrc << EOF ...'
```

Die Beschreibung soll hier nachgeholt werden:

Jede Datei in allen mir bekannten Betriebssystemen (also nicht nur Unix/Linux und hiermit zwangsläufig das auf BSD basierende Mac OS X, sondern auch Windows) markieren das Ende einer

Datei durch das EOF-Zeichen. Wäre dies nicht der Fall, könnten Sie niemals in einer `do-while`-Schleife (z.B. in Java oder C/C++) den Inhalt von Dateien umkopieren oder auslesen: Das Programm würde in einer Endlosschleife verharren. Hier passiert nun Folgendes:

1. An das Ende der Datei `/root/.bashrc` sollen die Umgebungsvariablen LC_ALL, LANGUAGE und LANG mitsamt ihrer Werte angehängt werden. Das Anhängen der Umgebungsvariablen erfolgt mit Hilfe des Kommandos `cat` (Abkürzung von*concatenate* = verbinden/anhängen).
2. Damit die Umgebungsvariablen nicht den bereits vorhandenen Inhalt der Datei überschreiben, muss zunächst das Dateiende (EOF) gesucht werden. Dies geschieht mit `<< EOF`.
3. In den folgenden Zeilen werden die Umgebungsvariablen an das nun gefundene Ende der Datei automatisch eingetragen (dies passiert auf die gleiche Weise, als würden Sie zeilenweise die Umgebungsvariablen mitsamt ihrer Werte per Tastatur eingeben). Das alte EOF wird hierbei natürlich überschrieben.
4. Damit ist aber die Datei noch nicht vollständig: Es muss noch ein neues EOF angehängt werden. Dies erfolgt durch die vorletzte Zeile, in der nur noch EOF steht.
5. Durch die Option `-c` in Zusammenhang mit `sh -c` wird der gesamte in Hochkommas eingefasste Text (also `'cat >> /root/.bashrc << EOF ...'`) wie eine normale Zeichenkette behandelt und erst innerhalb der Shell sh "expandiert". "Expandieren" bedeutet hier, dass die Zeichenkette (String) nicht als Text, sondern als Kommando von der Shell interpretiert wird.

Ich hatte "das Kaliber" dieses Scriptteils wohl doch etwas unterschätzt: Sie sehen, dass die Beschreibung seiner Funktion umfangreicher als das Script ist. Das wird in der zweiten Auflage nachgereicht.

Hinzufügen eines neuen Standardusers (Seite 196)

Im markierten Bereich (letzte Box auf Seite 196) wird die Zeile

```
usermod -a -G sudo,staff,kmem,plugdev pi
```

mit einem Punkt abgeschlossen. Der Punkt darf dort nicht stehen. Er wurde beim Korrekturlesen übersehen.

Partitionstabelle (Seite 200)

Die Partitionstabelle muss **zwingend** genauso abgetippt werden, wie sie im Buch abgebildet ist. Fehlt auch nur eine der Leerzeilen, so funktioniert es nicht. Für die zweite Auflage überlege ich, die einzelnen Zeilen durchzunummerieren.

Größe der Imagedatei (Seite 201)

Lassen Sie sich nicht davon verwirren, dass das finale Image nur ca. 2 GB groß ist, obwohl vorher (Abschnitt 6.1.3, Seite 198) 5 GB reserviert wurden. Das Kommando **dd** arbeitet Byte für Byte, was sehr lange dauern kann (ca. 30 Minuten). Ich bin hier einen Kompromiss zwischen der maximalen Größe von 16 GB auf der von mir verwendeten SD-Karte und der erwarteten Größe des Images eingegangen. Kann **dd** seine Arbeit aufgrund einer zu geringen Größe des reservierten Bereiches nicht alle Daten kopieren, bricht es einfach ab. Die Folge: Sie müssen die ganze Arbeit wiederholen.

rpiimage.img und rpiimg.img (Seite 202)

Ein Leser hat mich darauf aufmerksam gemacht, dass das Image in Abbildung 6.7 auf Seite 198 und später auf Seite 202 verschiedene Namen hat. Dies ist nicht schön, stellt aber keinen Fehler dar. Auf Seite 198 wird nur der Speicherplatz für das Image auf der SD-Karte reserviert. Dieser Bereich ist aber nach dem Reservieren leer und stellt nicht das spätere >>echte<< Image dar. Die Namen

können beliebig gewählt werden und dürfen auch identisch sein. Gerade Neulinge kann aber die Ähnlichkeit beider Namen verwirren.

Schreiben der Boot-Partition (Seite 203)

Vor dem Kommando

```
sudo cp -R bootfs/* /mnt/bootfs
```

fehlt der Hinweis, dass das Zielverzeichnis vorher mit `mkdir` erzeugt werden muss. In Abschnitt 6.1.5, in dem das Schreiben des root-Dateisystems beschrieben wird, ist das Kommando `mkdir` enthalten.