

Linux From Scratch

Version 6.1.1

Gerard Beekmans

Linux From Scratch: Version 6.1.1

von Gerard Beekmans

Copyright © 1999–2005 Gerard Beekmans

Copyright © 1999–2005, Gerard Beekmans

Alle Rechte vorbehalten.

Weiterverteilung und Benutzung in Quell- und Binärform, mit oder ohne Modifikationen, ist erlaubt, solange die folgenden Bedingungen eingehalten werden:

- Weitergegebenes Material in jeglicher Form muss den obigen Copyrighthinweis, die Liste der Bedingungen und den folgenden Ausschlussvermerk beibehalten
- Weder der Name „Linux From Scratch“ noch die Namen der Mitwirkenden dürfen ohne vorherige schriftliche Genehmigung zu Werbezwecken für abgeleitetes Material benutzt werden
- Jegliches von Linux From Scratch abgeleitetes Material muss einen Verweis auf das Projekt „Linux From Scratch“ enthalten

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS „AS IS“ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Inhaltsverzeichnis

Einleitung	vii
1. Vorwort	vii
2. Warum sollte man dieses Buch lesen?	viii
3. Voraussetzungen	x
4. Mindestanforderungen an das Host-System	xi
5. Konventionen in diesem Buch	xii
6. Aufbau	xiii
7. Errata	xiv
I. Einführung	1
1. Einführung	2
1.1. Vorgehensweise zur Installation von LFS	2
1.2. Änderungsprotokoll	4
1.3. Ressourcen	6
1.4. Hilfe	7
II. Vorbereitungen zur Installation	10
2. Vorbereiten einer neuen Partition	11
2.1. Einführung	11
2.2. Erstellen einer neuen Partition	12
2.3. Erstellen eines Dateisystems auf der neuen Partition	13
2.4. Einhängen (mounten) der neuen Partition	14
3. Pakete und Patches	15
3.1. Einführung	15
3.2. Alle Pakete	16
3.3. Erforderliche Patches	20
4. Abschluss der Vorbereitungen	23
4.1. Die Variable \$LFS	23
4.2. Erstellen des Ordners \$LFS/tools	24
4.3. Hinzufügen des LFS-Benutzers	25
4.4. Vorbereiten der Arbeitsumgebung	26
4.5. Informationen zu SBUs	29
4.6. Über die Testsuites	30
5. Erstellen eines temporären Systems	31
5.1. Einführung	31
5.2. Technische Anmerkungen zur Toolchain	33
5.3. Binutils-2.15.94.0.2.2 - Durchlauf 1	38
5.4. GCC-3.4.3 - Durchlauf 1	40
5.5. Linux-Libc-Header-2.6.11.2	42
5.6. Glibc-2.3.4	43
5.7. Anpassen der Toolchain	48
5.8. Tcl-8.4.9	50
5.9. Expect-5.43.0	52
5.10. DejaGNU-1.4.4	54
5.11. GCC-3.4.3 - Durchlauf 2	55
5.12. Binutils-2.15.94.0.2.2 - Durchlauf 2	60

5.13. Gawk-3.1.4	62
5.14. Coreutils-5.2.1	63
5.15. Bzip2-1.0.3	64
5.16. Gzip-1.3.5	65
5.17. Diffutils-2.8.1	66
5.18. Findutils-4.2.23	67
5.19. Make-3.80	68
5.20. Grep-2.5.1a	69
5.21. Sed-4.1.4	70
5.22. Gettext-0.14.3	71
5.23. Ncurses-5.4	72
5.24. Patch-2.5.4	73
5.25. Tar-1.15.1	74
5.26. Texinfo-4.8	75
5.27. Bash-3.0	76
5.28. M4-1.4.3	77
5.29. Bison-2.0	78
5.30. Flex-2.5.31	79
5.31. Util-linux-2.12q	80
5.32. Perl-5.8.7	81
5.33. Stripping	82
III. Installation des LFS-Systems	83
6. Installieren der grundlegenden System-Software	84
6.1. Einführung	84
6.2. Einhängen der virtuellen Kernel-Dateisysteme	85
6.3. Betreten der chroot-Umgebung	86
6.4. Ändern des Besitzers	87
6.5. Erstellen der Ordnerstruktur	88
6.6. Erstellen notwendiger symbolischer Links	89
6.7. Erstellen der Dateien passwd, group und der Logdateien	90
6.8. Bestücken von /dev mit Gerätedateien	92
6.9. Linux-Libc-Header-2.6.11.2	94
6.10. Man-pages-2.01	95
6.11. Glibc-2.3.4	96
6.12. Erneutes Anpassen der Toolchain	104
6.13. Binutils-2.15.94.0.2.2	106
6.14. GCC-3.4.3	109
6.15. Coreutils-5.2.1	113
6.16. Zlib-1.2.3	119
6.17. Mktmp-1.5	121
6.18. Iana-Etc-1.04	122
6.19. Findutils-4.2.23	123
6.20. Gawk-3.1.4	125
6.21. Ncurses-5.4	126
6.22. Readline-5.0	128
6.23. Vim-6.3	130
6.24. M4-1.4.3	133
6.25. Bison-2.0	134
6.26. Less-382	135

6.27. Groff-1.19.1	136
6.28. Sed-4.1.4	139
6.29. Flex-2.5.31	140
6.30. Gettext-0.14.3	142
6.31. Inetutils-1.4.2	145
6.32. IPRoute2-2.6.11-050330	147
6.33. Perl-5.8.7	150
6.34. Texinfo-4.8	152
6.35. Autoconf-2.59	154
6.36. Automake-1.9.5	157
6.37. Bash-3.0	160
6.38. File-4.13	162
6.39. Libtool-1.5.14	163
6.40. Bzip2-1.0.3	164
6.41. Diffutils-2.8.1	166
6.42. Kbd-1.12	167
6.43. E2fsprogs-1.37	169
6.44. Grep-2.5.1a	172
6.45. GRUB-0.96	173
6.46. Gzip-1.3.5	175
6.47. Hotplug-2004_09_23	177
6.48. Man-1.5p	179
6.49. Make-3.80	181
6.50. Module-Init-Tools-3.1	182
6.51. Patch-2.5.4	184
6.52. Procps-3.2.5	185
6.53. Psmisc-21.6	187
6.54. Shadow-4.0.9	189
6.55. Sysklogd-1.4.1	194
6.56. Sysvinit-2.86	196
6.57. Tar-1.15.1	199
6.58. Udev-056	200
6.59. Util-linux-2.12q	202
6.60. Informationen zu Debugging Symbolen	206
6.61. Erneutes Stripping	207
6.62. Aufräumen	208
7. Aufsetzen der System-Bootskripte	209
7.1. Einführung	209
7.2. LFS-Bootskripte-3.2.1	210
7.3. Wie funktionieren diese Boots-kripte?	212
7.4. Umgang mit Geräten und Modulen an einem LFS-System	214
7.5. Einrichten des setclock-Skripts	218
7.6. Einrichten der Linux Konsole	219
7.7. Einrichten des sysklogd-Skripts	221
7.8. Erstellen der Datei /etc/inputrc	222
7.9. Die Startdateien von Bash	224
7.10. Einrichten des localnet-Skripts	227
7.11. Erstellen der Datei /etc/hosts	228
7.12. Einrichten des network-Skripts	229

8. Das LFS-System bootfähig machen	231
8.1. Einführung	231
8.2. Erstellen der Datei /etc/fstab	232
8.3. Linux-2.6.11.12	233
8.4. Das LFS-System bootfähig machen	237
9. Ende	239
9.1. Ende	239
9.2. Lassen Sie sich zählen	240
9.3. Neustarten des Systems	241
9.4. Was nun?	242
IV. Anhänge	243
A. Akronyme und Begriffe	244
B. Danksagungen	247
Stichwortverzeichnis	250

Einleitung

1. Vorwort

Meine Abenteuer mit Linux begannen 1998 mit dem Herunterladen und Installieren meiner ersten Distribution. Nach einer Weile fielen mir einige Dinge auf, die ich gerne verbessern wollte. Zum Beispiel gefielen mir weder die Zusammenstellung der Bootskripte noch die Voreinstellungen vieler Programme. Ich probierte ein paar alternative Distributionen aus, aber alle hatten neben den Vorteilen auch Nachteile. Schlussendlich wurde mir klar das ich mein eigenes Linux von Grund auf selbst erstellen musste um wirklich zufrieden zu sein.

Im einzelnen bedeutete dies nun, dass ich keinerlei vorkompilierte Pakete, CD-Roms oder Bootdisketten jeglicher Art für die Installation der grundlegenden Werkzeuge verwenden würde. Ich wollte mein bereits laufendes Linux-System als Grundlage benutzen, um darauf mein angepasstes Linux zu entwickeln. Dieses „perfekte“ Linux-System sollte die Stärken der verschiedenen Distributionen ohne deren Schwächen vereinen. Zu Beginn war die Umsetzung der Idee ziemlich entmutigend. Aber ich blieb engagiert bei der Sache. Ich wollte schließlich ein Linux-System, das meinen Ansprüchen gerecht wurde, anstatt einer Standard-Distribution, die nicht meinen Wünschen entsprach.

Um das meinen Wünschen entsprechende Linux zu erstellen musste ich erstmal viele Probleme mit gegenseitigen Abhängigkeiten und jede Menge Kompilierfehler beheben. Als ich damit fertig war, hatte ich jedoch ein voll funktionsfähiges und anpassbares Betriebssystem. Meine Vorgehensweise ermöglicht das Erstellen sehr kompakter Linux-Systeme, die schneller sind und weniger Speicher verbrauchen als viele herkömmliche Betriebssysteme. Ich nannte dieses System Linux From Scratch, oder einfach kurz LFS.

Ich teilte meine Erfahrungen mit anderen Mitgliedern der Linux-Gemeinschaft und es stellte sich schnell ein wachsendes Interesse an der Fortsetzung meiner Arbeit mit Linux heraus: Selbstgebaute LFS-Systeme entsprechen nicht einfach nur Spezifikationen und Anforderungen von Anwendern, sondern sind auch eine ideale Lernbasis für Programmierer und Systemadministratoren mit der man seine Linux-Kenntnisse erweitern kann. Aus diesem breiten Interesse heraus entstand dann das Projekt Linux From Scratch.

Das Buch *Linux From Scratch* vermittelt dem Leser das Wissen und nötige Anleitungen um ein eigenes Linux-System zu entwerfen und zu erstellen. Es hebt das Projekt Linux From Scratch und die Vorteile dieses Systems hervor. Der Leser kann alle Eigenschaften des Systems selber vorgeben, inklusive dem Layout der Ordnerstruktur, Skript-Einstellungen und Sicherheit. Das System wird direkt aus dem Quellcode kompiliert und man kann selber entscheiden, wo, warum und wie Programme installiert werden. Dieses Buch ermöglicht es jedem, Linux-Systeme an die eigenen Bedürfnisse anzupassen und mehr Kontrolle über das System zu erlangen.

Ich wünsche Ihnen viel Freude bei der Arbeit an Ihrem eigenen LFS-System. Genießen Sie die Vorteile eines Systems, das wirklich *Ihr Eigen* ist.

--

Gerard Beekmans
gerard@linuxfromscratch.org

2. Warum sollte man dieses Buch lesen?

Es gibt viele gute Gründe, dieses Buch zu lesen. Die meisten Leser möchten lernen, wie man ein Linux-System direkt aus den Quellen erstellt. Oft wird die Frage gestellt: „Warum soll man sich die Mühe machen, ein Linux-System selbst zu erstellen, wenn man einfach ein fertiges Linux herunterladen und installieren kann?“. Das ist eine berechtigte Frage und gleichzeitig auch der Anstoß für dieses Kapitel.

Ein wichtiges Ziel von LFS ist, dem Leser beizubringen wie Linux intern funktioniert. Der Selbstbau eines Linux-Systems veranschaulicht Ihnen, was Linux seinen Herzschlag verleiht und wie die Komponenten zusammenarbeiten und voneinander abhängen. Das Beste daran ist, dass Sie durch den Lernprozess in die Lage versetzt werden, Linux an Ihre eigenen Anforderungen und Vorlieben anzupassen.

Einer der grössten Vorteile von LFS ist, dass Sie mehr Kontrolle über Ihr System erhalten, ohne sich auf die Linux-Version von jemand anders verlassen zu müssen. Mit LFS sitzen *Sie selbst* am Steuer und können jeden Aspekt Ihres Systems beeinflussen, wie zum Beispiel das Ordner-Layout oder die Einrichtung der Bootskripte. Auch bestimmen Sie, wo, warum und wie Programme installiert werden.

Ein weiterer Vorteil von LFS ist die Möglichkeit, Linux sehr kompakt zu halten. Wenn Sie eine übliche Linux-Distribution verwenden, installieren Sie für gewöhnlich viele Programme die Sie nie benutzen werden. Diese liegen dann unnützlich auf der Festplatte und verbrauchen Speicherplatz (oder CPU-Ressourcen). Es ist leicht, ein LFS-System unter 100 MB zu installieren. Das ist immer noch zu groß? Einige LFS-Mitglieder haben an einem sehr kleinen Embedded-Linux gearbeitet. Sie haben einen Apache-Webserver auf einem Linux From Scratch mit gerade mal 8 MB belegtem Festplattenspeicher installiert. Durch weitere Einschränkungen könnte das System auf bis zu 5 MB oder weniger schrumpfen. Versuchen Sie das mal mit einer herkömmlichen Linux-Distribution.

Man könnte die verschiedenen Linux-Distributionen mit einem Hamburger aus einer Fast-Food-Kette vergleichen—man weiß nie genau was man isst. LFS auf der anderen Seite wäre nicht der Burger, sondern vielmehr das Rezept. Man kann das Rezept überprüfen, ungewollte Zutaten weglassen und eigene Zutaten nach Geschmack und Belieben hinzufügen. Wenn man zufrieden ist bereitet man es zu. Und auch hier kann man variieren—braten, backen, tiefgefrieren, grillen oder roh essen, ganz wie man will.

Es gibt noch weitere Analogien: Vergleichen Sie LFS z. B. mit einem Fertighaus. LFS wäre in dem Fall der Plan für den Grundriss, aber bauen müssen Sie das Haus selber. Jeder kann den Plan ganz nach Belieben ändern.

Nicht zuletzt ist auch Sicherheit ein Vorteil eines selbstgebauten Linux-Systems. Wer ein Linux-System selber aus den Quellen kompiliert, kann sämtliche Quelltexte sichten und alle für wichtig erachteten Sicherheitspatches installieren. Man muss nicht warten bis jemand anders Binärpakete zur Behebung von Sicherheitslöchern bereitstellt. Solange Sie die Patches nicht selber prüfen und installieren, ist auch nicht sichergestellt, dass das Binärpaket korrekt kompiliert wurde und dass es das Problem auch wirklich behebt.

Das erklärte Ziel von Linux From Scratch ist, ein vollständiges, lauffähiges und grundsolides System zu erstellen. Wenn Sie nur interessiert sind, was genau beim Hochfahren Ihres Computers geschieht, dann empfehlen wir das HOWTO „From Power Up To Bash Prompt“; Sie bekommen es unter <http://axiom.anu.edu.au/~okeefe/p2b/> oder auf der Webseite des Linux Documentation Project unter <http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html>. Mit Hilfe dieses HOWTOs wird ein blankes System installiert, das dem in diesem Buch sehr ähnlich ist, sich aber ausschließlich auf das Erstellen eines Systems konzentriert, das eine Bash-Shell booten kann. Halten Sie sich am besten Ihr Ziel vor Augen: wenn Sie Linux installieren und nebenbei dazulernen möchten, dann ist Linux From Scratch für Sie geeignet.

Es gibt einfach zu viele gute Gründe für das Erstellen eines eigenen LFS-Systems um sie hier alle aufzuzählen, die hier genannten Gründe sind nur die Spitze des Eisberges. Während Sie mit LFS arbeiten und Erfahrungen sammeln, werden Sie selbst schnell feststellen wie wertvoll Informationen und Wissen über Linux sind.

3. Voraussetzungen

Der Selbstbau eines LFS ist keine leichte Aufgabe. Man benötigt ein entsprechendes Vorwissen zur Administration von Unix-Systemen, sonst fällt es schwer, bestimmte Kommandos zu verstehen oder auf Fehlersuche zu gehen. Sie als Leser sollten als absolutes Minimum zumindest mit der Kommandozeile (Shell) umgehen können (dazu gehört das Kopieren und Verschieben von Dateien und Ordnern, Auflisten von Ordner- und Dateiinhalte und das Wechseln des aktuellen Ordners). Außerdem setzen wir voraus, dass Sie grundsätzlich wissen, wie man Linux-Software benutzt und installiert.

Weil das LFS-Buch dieses Vorwissen als *absolutes Minimum* voraussetzt, werden Sie in den verschiedenen LFS Support-Foren höchstwahrscheinlich keine Hilfe bekommen, wenn Sie Fragen ohne das notwendige Basiswissen stellen. Möglicherweise bleiben Ihre Fragen einfach nur unbeantwortet oder man verweist Sie auf diesen Text.

Bevor Sie ein LFS-System erstellen, lesen Sie bitte die folgenden HOWTOs:

- Software-Building-HOWTO <http://www.tldp.org/HOWTO/Software-Building-HOWTO.html>
Das Software-Building-HOWTO ist ein umfangreiches Handbuch zum Erstellen und Installieren „allgemeiner“ UNIX-Software unter Linux.
- The Linux Users' Guide <http://www.linuxhq.com/guides/LUG/guide.html>
Dieses Handbuch erklärt die Verwendung ausgewählter Linux-Software.
- The Essential Pre-Reading Hint http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt
Dies ist eine LFS-Anleitung, die speziell für neue Linux-Anwender geschrieben wurde. Sie enthält eine Linksammlung sehr guter Informationsquellen zu allen möglichen Themen. Jeder der LFS installieren möchte, sollte zumindest den Großteil der dort behandelten Themen verstehen.

4. Mindestanforderungen an das Host-System

Der Host muss mindestens auf Kernel 2.6.2 (kompiliert mit GCC-3.0 oder höher) laufen. Es gibt zwei Hauptgründe für diese hohen Anforderungen. Erstens stürzt die Native POSIX Threading Library (NPTL) Testsuite ab, wenn der Host-Kernel nicht mit GCC 3.0 oder höher kompiliert wurde. Zweitens wird Kernel 2.6.2 benötigt, weil erst ab dieser Version Udev unterstützt wird. Udev erstellt Gerätedateien dynamisch basierend auf dem `sysfs`-Dateisystem. Die Unterstützung für dieses Dateisystem wurde in die meisten Kernel-Treiber erst vor kurzem eingebaut. Wir müssen sicherstellen, dass alle kritischen System-Gerätedateien korrekt erzeugt werden.

Um herauszufinden, ob der Host-Kernel den Anforderungen genügt, können Sie dieses Kommando verwenden:

```
cat /proc/version
```

Das Erzeugt diese oder eine ähnliche Ausgabe:

```
Linux version 2.6.2 (user@host) (gcc version 3.4.0) #1  
Tue Apr 20 21:22:18 GMT 2004
```

Wenn aus der Ausgabe des obigen Kommandos zu entnehmen ist, dass der Host-Kernel jünger als 2.6.2 ist oder dass er nicht mit mindestens GCC 3.0 kompiliert wurde, dann muss auf dem Host erstmal ein solcher Kernel installiert und gebootet werden. Es gibt zwei Möglichkeiten, das Problem zu beheben: Überprüfen Sie, ob der Hersteller Ihrer Host-Distribution einen entsprechenden Kernel zur Verfügung stellt. Wenn ja, installieren Sie diesen. Falls der Hersteller jedoch keinen passenden Kernel ausliefert (oder Sie diesen aus irgendwelchen Gründen nicht installieren möchten), dann können Sie selbst einen 2.6er Kernel kompilieren. Eine Hilfestellung dazu finden Sie in Kapitel 8 (vorausgesetzt der Host verwendet GRUB als Bootloader). Den zweiten Lösungsweg kann man zudem auch als Test Ihrer Fähigkeiten verstehen: Wenn Sie keinen neuen Kernel von Hand installieren können, hat das Buch wahrscheinlich noch keinen Nutzen für Sie.

5. Konventionen in diesem Buch

Das Buch hält sich an einige typografische Konventionen, die zum allgemein besseren Verständnis beitragen sollen. Es folgen einige Beispiele:

```
./configure --prefix=/usr
```

Solange nicht anders angegeben, muss Text in dieser Textform exakt so eingegeben werden, wie er zu sehen ist. Diese Darstellung wird auch in den Erklärungstexten verwendet, um sich eindeutig auf bestimmte Kommandos zu beziehen.

```
install-info: unknown option '--dir-file=/mnt/lfs/usr/info/dir'
```

Diese Textform (Text mit fester Zeichenbreite) stellt Bildschirmausgaben dar. Text in dieser Form ist oft die Ausgabe von vorher eingegebenen Kommandos. Außerdem wird diese Textform für Dateinamen wie z. B. `/etc/ld.so.conf` verwendet.

Hervorhebung

Diese Textform wird für verschiedene Zwecke benutzt und soll wichtige Details hervorheben.

<http://www.linuxfromscratch.org/>

Auf diese Weise werden Links dargestellt, sowohl innerhalb des Buches als auch zu externen Seiten wie z. B. HOWTOs, Download-Adressen und Webseiten.

```
cat > $LFS/etc/group << "EOF"
root:x:0:
bin:x:1:
.....
EOF
```

Solche Textabschnitte werden hauptsächlich beim Erstellen von Konfigurationsdateien verwendet. Der obige Block erzeugt die Datei `$LFS/etc/group` mit dem nachfolgenden Inhalt bis die Zeichenfolge EOF erkannt wird. Normalerweise müssen Sie Text in dieser Form exakt so eingeben wie er zu sehen ist.

[ZU ERSETZENDER TEXT]

Dies ist Text den Sie nicht einfach blindlings abschreiben oder kopieren und einfügen dürfen.

`passwd(5)`

Diese Textform wird verwendet, um sich auf eine Man-page zu beziehen. Die Zahl in Klammern bezeichnet eine bestimmte Sektion in **man**. **passwd** z. B. hat zwei Man-pages. Nach der LFS-Anleitung werden diese nach `/usr/share/man/man1/passwd.1` und `/usr/share/man/man5/passwd.5` installiert. Beide Man-pages enthalten unterschiedliche Informationen und Themenbereiche. Wenn Sie also `passwd(5)` lesen, bezieht sich das Buch explizit auf `/usr/share/man/man5/passwd.5`. Das Kommando **man passwd** zeigt die erste gefundene Man-page zu `passwd` an. Das wäre in diesem Fall `/usr/share/man/man1/passwd.1`. Um in diesem Beispiel die richtige Man-page aus Sektion 5 anzuzeigen müssen Sie das Kommando **man 5 passwd** verwenden. Die meisten Man-pages haben keine doppelten Seiten-Namen in unterschiedlichen Sektionen, daher ist **man [Programmname]** meistens ausreichend.

6. Aufbau

Das Buch ist in die folgenden Abschnitte unterteilt.

6.1. Teil I - Einführung

Teil I erläutert einige wichtige Hinweise zur Installation und schafft Grundlagen zur allgemeinen Verwendung des Buches.

6.2. Teil II - Vorbereitungen zur Installation

Teil II bereitet den eigentlichen Installationsvorgang vor—Anlegen einer Partition, Herunterladen der Pakete und Kompilieren der temporären Werkzeuge.

6.3. Teil III - Installation

Teil III führt Sie Schritt für Schritt durch die eigentliche Installation von LFS—Kompilieren und Installieren aller Pakete, Aufsetzen der Bootskripte und Installieren des Kernels. Das resultierende Linux-System ist die Basis, auf der später weitere Software installiert wird und auf der das System ganz nach Ihrem Belieben erweitert werden kann. Am Ende des Buches finden Sie zu Referenzzwecken eine Liste aller Programme, Bibliotheken und wichtiger Dateien, die während der Arbeit mit diesem Buch installiert wurden.

7. Errata

Die für LFS verwendete Software wird laufend aktualisiert und erweitert. Nach Erscheinen des Buches könnten Sicherheitshinweise und Fehlerbereinigungen hinzugekommen sein. Bevor Sie mit dem Bau von LFS beginnen, sollten Sie unter <http://www.linuxfromscratch.org/lfs/errata/6.1.1/> nachsehen, ob Änderungen an den Installationsanleitungen oder an Softwareversionen nötig sind. Bitte notieren Sie alle nötigen Änderungen und wenden Sie diese in den entsprechenden Kapiteln an.

Teil I. Einführung

Kapitel 1. Einführung

1.1. Vorgehensweise zur Installation von LFS

Sie werden LFS mit Hilfe einer bereits laufenden Linux-Distribution (wie z. B. Debian, Mandrake, Red Hat oder SuSE) installieren. Das bestehende System (der Host) wird als Einstiegspunkt benutzt, denn Sie brauchen Programme wie Compiler, Linker und eine Shell, um Ihr neues System zu erstellen. Normalerweise sind alle notwendigen Programme bereits installiert, wenn Sie bei der Installation Ihrer Distribution die Kategorie „Entwicklung“ ausgewählt haben.

Falls Sie nur wegen LFS kein neues Host-System installieren möchten dann sollten Sie die Linux From Scratch Live-CD verwenden. Die CD enthält ein voll funktionsfähiges Host-System mit allen notwendigen Werkzeugen für eine erfolgreiche Installation. Zudem enthält sie auch alle Quellpakete, Patches und eine Online-Version dieses Buches. Wenn Sie die CD verwenden, brauchen Sie auch keine Netzwerkverbindung weil nichts mehr heruntergeladen werden muss. Weitere Informationen zu der CD finden Sie unter <http://www.linuxfromscratch.org/livecd/>. Dort können Sie auch eine Kopie der CD herunterladen.

Kapitel 2 beschreibt das Anlegen einer neuen Linux-Partition und eines Dateisystems, auf dem Ihr neues LFS-System kompiliert und installiert wird. In Kapitel 3 erfahren Sie, welche Pakete und Patches Sie herunterladen müssen. Kapitel 4 erklärt das Einrichten einer funktionsfähigen Arbeitsumgebung für die kommenden Arbeitsschritte. Bitte lesen Sie Kapitel 4 aufmerksam durch! Es behandelt ein paar mögliche Schwierigkeiten, die Ihnen vor der Arbeit mit Kapitel 5 und den folgenden bekannt sein sollten.

Kapitel 5 beschreibt dann die Installation der Pakete für die grundlegende Entwicklungsumgebung (im weiteren Verlauf des Buches *Toolchain* genannt). Die Toolchain ist eine Sammlung der nötigsten Werkzeuge und wird später in Kapitel 6 verwendet um das endgültige System zu erstellen. Einige dieser Pakete werden zum Auflösen rekursiver Abhängigkeiten benötigt—zum Beispiel brauchen Sie einen Compiler, um einen Compiler zu kompilieren.

Kapitel 5 erklärt auch, wie die erste Version der Basiswerkzeuge, inklusive Binutils und GCC, erzeugt wird. „Erste Version“ bedeutet in diesem Zusammenhang, dass die Pakete noch ein zweites Mal kompiliert und installiert werden. Als zweiten Schritt kompilieren Sie Glibc, die C-Bibliothek. Glibc wird mit den Programmen der im ersten Schritt erstellten Basiswerkzeuge kompiliert. Im dritten Schritt erstellen Sie dann die zweite Version der Basiswerkzeuge. Sie linken die Programme dynamisch gegen die gerade frisch installierte Glibc. Die verbleibenden Pakete aus Kapitel 5 werden alle diesen zweiten Durchlauf der Toolchain verwenden und dynamisch gegen die neue, hostunabhängige Glibc gelinkt. Wenn dies erledigt ist, ist der weitere Installationsvorgang — mit Ausnahme des Kernels — nicht mehr von der Linux-Distribution auf dem Host-System abhängig.

Dies scheint erstmal eine Menge Arbeit zu sein um sich von der Host-Distribution zu lösen. Eine vollständige Erklärung finden Sie in Abschnitt 5.2, „Technische Anmerkungen zur Toolchain“.

In Kapitel 6 wird das endgültige LFS-System erstellt. Wir benutzen das Programm **chroot** (chroot = change root = wechseln der Basis), um eine Shell in einer virtuellen Umgebung zu starten. In der neuen Shell ist der Basisordner auf die LFS-Partition eingestellt. Chroot'en ist so ähnlich wie Neustarten und Einhängen der LFS-Partition als root-Dateisystem. Das Erstellen eines bootfähigen Systems würde allerdings zusätzliche Arbeit erfordern und ist an dieser Stelle absolut unnötig. Außerdem hat chroot'en den Vorteil, dass Sie das Host-Betriebssystem weiter nebenher verwenden können während Sie in der Shell das LFS installieren. Während Sie also warten bis alle Pakete kompiliert sind, können Sie einfach auf ein anderes VT (Virtuelles Terminal) oder auf den X-Desktop wechseln und dort wie gewohnt weiterarbeiten.

Zum Abschluss der Installation werden in Kapitel 7 die Bootskripte eingerichtet; der Kernel sowie der Bootloader werden in Kapitel 8 behandelt. Wenn Sie das Buch zuende gelesen haben finden Sie in Kapitel 9 Links auf weiterführende Hilfeseiten. Abschließend ist der Computer bereit für einen Neustart mit dem neuen LFS-System.

Nun kennen Sie die allgemeine Vorgehensweise in stark zusammengefasster Form. Die jeweiligen Kapitel beinhalten natürlich detailliertere Informationen. Machen Sie sich keine Gedanken, falls hier noch etwas unklar sein sollte; alle offene Fragen werden sich im weiteren Verlauf klären.

1.2. Änderungsprotokoll

Dies Linux From Scratch 6.1.1 vom 30. November 2005. Wenn dieses Buch älter als ein halbes Jahr ist, gibt es vielleicht schon eine neuere, bessere Version. Bitte besuchen Sie einen unserer Spiegel-Server unter <http://www.linuxfromscratch.org/>.

Die folgende Liste enthält alle Änderungen seit der Veröffentlichung der Vorgängerversion. Der obere Teil ist eine Zusammenfassung und danach folgt das detaillierte Protokoll mit den Original-Kommentaren der Entwickler.

- Aktualisierung der Version auf:
 - Perl 5.8.7
 - Zlib 1.2.3
- Hinzugefügt:
 - binutils-2.15.94.0.2.2-gcc4-1.patch
 - bzip2-1.0.3-install_docs-1.patch
 - bzip2-1.0.3-bzgrep_security-1.patch
 - glibc-2.3.4-rtld_search_dirs-1.patch
 - glibc-2.3.4-tls_assert-1.patch
 - texinfo-4.8-tempfile_fix-1.patch
 - util-linux-2.12q-umount_fix-1.patch
 - vim-6.3-security_fix-2.patch
- Entfernt:
 - zlib-1.2.2-security_fix-1.patch;
- November 26, 2005 [matt]: Remove the incorrect note about not having to dump/check a journalled filesystem (fixes bug 1662).
- November 26, 2005 [manuel]: PDF fix.
- November 26, 2005 [manuel]: Changed --strip-path to --strip-components in the unpack of module-init-tools-testsuite package.
- November 24, 2005 [matt]: LFS-6.1.1-pre2 release.
- November 24, 2005 [matt]: Fix an issue with Glibc that prevents some programs (including OpenOffice.org) from running.
- November 23, 2005 [gerard]: Corrected reference to 'man page' to 'HTML documentation' in chapter 6/sec
- November 18, 2005 [manuel]: Fixed the unpack of the module-init-tools-testsuite package.

- November 18, 2005 [manuel]: PDF fixes.
- November 17, 2005 [matt]: LFS-6.1.1-pre1 release.
- November 12, 2005 [matt]: Improve the heuristic for determining a locale that is supported by both Glibc and packages outside LFS (bug 1642). Many thanks to Alexander Patrakov for highlighting the numerous issues and for reviewing the various suggested fixes.
- November 12, 2005 [matt]: Omit running Bzip2's testsuite as a separate step, as **make** runs it automatically (bug 1652).
- November 7, 2005 [matt]: Stop Udev from killing udevd processes on the host system (fixes bug 1651). Thanks to Alexander Patrakov for the report and the fix.
- November 5, 2005 [matt]: Add a note to the toolchain sanity check in chapter 5 to explain that if TCL fails to build, it's an indication of a broken toolchain (bug 1581).
- November 4, 2005 [matt]: Correct the instructions for running Module-Init-Tools' testsuite (fixes bug 1597). Thanks to Greg Schafer, Tushar Teredesai and to Randy McMurchy for providing the patch.
- October 29, 2005 [manuel]: PDF fixes.
- October 23, 2005 [manuel]: Added Bash documentation installation. Added notes about libiconv and Cracklib. Fixed the installation of Sed documentation. Replaced a patch for IPRoute2 by a sed command.
- October 19, 2005 [manuel]: Updated the acknowledgements to current trunk version. Ported some redaction changes in preface and chapter01 pages. Moved chapter02 to part II. Added -v switches. Ported several typos and redaction fixes from trunk.
- October 19, 2005 [manuel]: Updated the stylesheets, Makefile and related files to current trunk versions.
- October 15, 2005 [matt]: Use an updated version of the Udev rules file (fixes bug 1639).
- October 15, 2005 [matt]: Add a cdrom group as required by the Udev rules file
- October 14th, 2005 [ken]: Added a patch to allow binutils to be built from a host running gcc-4, updated glibc instructions for the rtdl patch, updated space/time for perl and zlib.
- October 14th, 2005 [matt]: Added a patch to fix a security vulnerability in util-linux.
- October 14th, 2005 [matt]: Added the updated vim security patch.
- October 14th, 2005 [jhuntwork]: Added the bzip2 security and install docs patches.
- October 14th, 2005 [jhuntwork]: Added the tempfile patch for texinfo.
- October 14th, 2005 [ken]: Update packages and patches in the changelog to only reflect changes since 6.1. Update zlib.
- October 13th, 2005 [ken]: Fix known errors in lists of installed files and bump the perl version.

1.3. Ressourcen

1.3.1. FAQ

Wenn Sie beim Erstellen von LFS Schwierigkeiten oder Fragen haben oder wenn Sie einen (Rechtschreib-) Fehler im Buch finden, dann lesen Sie bitte die FAQ (Frequently Asked Questions - häufig gestellte Fragen) unter <http://www.linuxfromscratch.org/faq/>.

1.3.2. Mailinglisten

Auf dem Server `linuxfromscratch.org` werden einige Mailinglisten für die Entwicklung des LFS-Projektes betrieben. Unter anderem befinden sich dort auch die Entwickler- und Support-Mailinglisten. Falls die FAQ Ihnen mit Ihrem Problem nicht helfen kann, sollten Sie im nächsten Schritt die Mailinglisten unter <http://www.linuxfromscratch.org/search.html> durchsehen.

Welche Listen es gibt, wie Sie eine Liste abonnieren können, wo Sie die Archive finden und vieles mehr erfahren Sie unter <http://www.linuxfromscratch.org/mail.html>.

1.3.3. News-Server

Alle Mailinglisten von `linuxfromscratch.org` sind auch über das NNTP-Protokoll verfügbar. E-Mails an die Mailinglisten werden in die dazugehörige Newsgruppe kopiert und umgekehrt.

Der News-Server hat die Adresse `news.linuxfromscratch.org`.

1.3.4. IRC

Viele Mitglieder aus der LFS-Gemeinschaft bieten ihre Hilfe über unseren IRC-Server an. Bevor Sie hier Hilfe suchen lesen Sie bitte zumindest die FAQ und die Archive unserer Mailinglisten und suchen dort nach einer Antwort auf Ihre Frage. Der IRC-Server hat die Adresse `irc.linuxfromscratch.org`. Der Support-Chatraum heißt `#LFS-support`.

1.3.5. Referenzen

Weitere Informationen und nützliche Tipps zu Software-Paketen finden Sie in der LFS Paket-Referenz unter <http://www.linuxfromscratch.org/~matthew/LFS-references.html>.

1.3.6. Softwarespiegel

Das LFS-Projekt hat viele über die ganze Welt verteilte Softwarespiegel. Diese stellen die Website zur Verfügung und vereinfachen das Herunterladen der benötigten Programme. Bitte besuchen Sie <http://www.linuxfromscratch.org/mirrors.html>, dort können Sie eine Liste der aktuellen Softwarespiegel einsehen.

1.3.7. Kontakt

Bitte senden Sie alle Fragen und Kommentare direkt an eine der LFS-Mailinglisten (siehe oben).

1.4. Hilfe

Wenn Sie beim Lesen des Buches auf ein Problem stoßen, sollten Sie als erstes in der FAQ unter <http://www.linuxfromscratch.org/faq/#generalfaq> nachlesen—die meisten Fragen werden hier schon beantwortet. Falls nicht, versuchen Sie die Ursache des Problems zu finden. Die folgende Anleitung könnte Ihnen bei der Fehlersuche behilflich sein: <http://www.linuxfromscratch.org/hints/downloads/files/errors.txt>.

Falls Sie Ihr Problem nicht in der FAQ finden, dann durchsuchen Sie am besten die Mailinglisten unter <http://www.linuxfromscratch.org/search.html>.

Wenn das nicht hilft, ist man im Internet Relay Chat (IRC) und auf den Mailinglisten (Abschnitt 1.3, „Ressourcen“) gern bereit, Ihnen zu helfen. Allerdings erhalten wir jeden Tag viele Anfragen, die durch einfaches Lesen der FAQ oder Durchlesen der Mailinglisten beantwortet werden könnten. Wir können Ihnen am besten helfen, wenn Sie zuerst selbst ein wenig auf Fehlersuche gehen. Wenn Ihre eigenen Recherchen keine Ergebnisse zutage bringen, dann unterstützen Sie uns bitte, indem Sie alle relevanten Informationen direkt mitsenden.

1.4.1. Dinge, die Sie angeben sollten

Neben einer kurzen Zusammenfassung des Problems ist es wichtig, dass Sie uns noch folgende Dinge mitteilen:

- Die Version dieses Buches (in diesem Fall Version 6.1.1),
- Host-Distribution und -Versionsnummer, die Sie zur Installation von LFS verwenden,
- die Software oder der Abschnitt, der Ihnen Probleme bereitet,
- die exakte Fehlermeldung bzw. die genauen Symptome, die Sie sehen,
- ob Sie von den Anleitungen im Buch abgewichen sind, und wenn ja, wie.



Anmerkung

Beachten Sie: Wir werden Ihnen auch helfen wenn Sie von den Anleitungen im Buch abgewichen sind. Schließlich ist die freie Wahl ein wichtiger Grundsatz von LFS. Ihr Hinweis hilft uns lediglich, die möglichen Ursachen für Ihr Problem besser zu erkennen.

1.4.2. Probleme mit configure-Skripten

Wenn beim Durchlaufen der **configure**-Skripte ein Problem auftritt, schauen Sie bitte zuerst in die Datei `config.log`. Sie enthält Fehlermeldungen, die auf dem Bildschirm normalerweise nicht angezeigt werden. Geben Sie die *relevanten* Fehlermeldungen mit an, wenn Sie um Hilfe bitten.

1.4.3. Kompilierprobleme

Sowohl Bildschirmausgaben als auch der Inhalt einiger Dateien sind für uns nützlich, um Ihnen bei der Fehlersuche zu helfen. Die Ausgaben des **configure**-Skriptes und die des Befehls **make** können sehr hilfreich sein. Bitte kopieren Sie aber nicht einfach blindlings die gesamte Ausgabe; andererseits sollte es aber auch nicht zu wenig sein. Als Beispiel für sinnvolle Informationen soll Ihnen folgende Ausgabe von **make** helfen:

```

gcc -DALIASPATH="/mnt/lfs/usr/share/locale:.\\"
-DLOCALEDIR="/mnt/lfs/usr/share/locale\"
-DLIBDIR="/mnt/lfs/usr/lib\"
-DINCLUDEDIR="/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2

```

In diesem Fallbeispiel kopieren viele leider nur den unteren Teil:

```
make [2]: *** [make] Error 1
```

Das reicht uns aber nicht, um Ihnen bei der Fehlerdiagnose helfen zu können, denn es sagt uns nur, *dass* etwas schiefgelaufen ist, aber nicht *was*. Sie müssen den ganzen oben gezeigten Block angeben, denn er enthält das ausgeführte Kommando und die dazugehörige Fehlermeldung(en).

Eric S. Raymond hat zu diesem Thema einen sehr guten Artikel geschrieben. Sie finden ihn unter <http://catb.org/~esr/faqs/smart-questions.html>. Lesen und befolgen Sie bitte seine Tipps. So erhöhen Sie Ihre Chance, dass Sie auf Ihre Frage eine Antwort erhalten, mit der Sie auch etwas anfangen können.

Teil II. Vorbereitungen zur Installation

Kapitel 2. Vorbereiten einer neuen Partition

2.1. Einführung

In diesem Kapitel bereiten Sie die Partition vor, die später Ihr neues LFS-System enthalten wird. Sie erstellen die Partition, erzeugen darauf ein Dateisystem und hängen sie anschließend ein (mounten).

2.2. Erstellen einer neuen Partition

Wie die meisten Betriebssysteme wird auch LFS auf einer separaten Partition installiert. Sie sollten für LFS bereits eine leere Partition haben, oder eine neue Partition anlegen. Ein LFS kann aber auch in einer bereits belegten Partition installiert werden, sodass mehrere Betriebssysteme nebeneinander existieren. Das Dokument http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt erklärt, wie man dies einrichtet. Im Buch gehen wir allerdings nur darauf ein, wie man LFS auf eine leere, dedizierte Partition installiert.

Für ein Minimal-System benötigen Sie eine Partition mit etwa 1,3 GB Platz. Das reicht aus, um die Quellpakete zu speichern und alle Pakete zu installieren. Wenn Sie Ihr LFS später als primäres Betriebssystem nutzen möchten, brauchen Sie zum Nachinstallieren weiterer Pakete mehr Platz (ca. 2 bis 3 GB). Das LFS-System selbst benötigt selbstverständlich nicht so viel Speicher. Der größte Teil wird als temporärer Speicher benötigt: Das Kompilieren von Paketen kann eine Menge Festplattenplatz in Anspruch nehmen, der aber nach dem Kompilierungsvorgang wieder freigegeben wird.

Manchmal ist zu wenig Random-Access-Memory (RAM, Arbeitsspeicher) verfügbar, daher sollte man eine kleine Partition als Swap-Partition einrichten—das ist Speicherplatz, den der Kernel zum Auslagern selten genutzter Daten verwendet. Das schafft Platz im Arbeitsspeicher für wichtigere Dinge. Die Swap-Partition in Ihrem LFS kann dieselbe sein wie die, die Sie bereits für ihr Host-System nutzen. Falls Sie also schon eine funktionsfähige Swap-Partition haben, müssen Sie keine zusätzliche erstellen.

Rufen Sie ein Partitionierungsprogramm wie zum Beispiel **cfdisk** oder **fdisk** auf. Als Argument übergeben Sie die Festplatte, auf der Sie die neue Partition erstellen möchten—zum Beispiel `/dev/hda` für die primäre Integrated Drive Electronics (IDE) Festplatte. Erstellen Sie eine native Linux-Partition (und eine Swap-Partition falls nötig). Bitte lesen Sie die Man-Page zu **cfdisk** oder **fdisk**, wenn Ihnen die Bedienung dieser Programme unklar ist.

Merken Sie sich die Bezeichnung Ihrer neuen Partition — sie könnte `hda5` oder ähnlich lauten. Das Buch bezeichnet diese Partition im weiteren Verlauf als die LFS-Partition. Wenn Sie (nun) eine Swap-Partition haben, merken Sie sich auch deren Bezeichnung. Sie werden sie später in die Datei `/etc/fstab` eintragen.

2.3. Erstellen eines Dateisystems auf der neuen Partition

Nun haben Sie eine leere Partition und können darauf ein Dateisystem anlegen. Das meistverbreitete Dateisystem unter Linux ist das Second Extended Filesystem (ext2); aber im Zuge der heute üblichen großen Festplatten gewinnen Journaling-Dateisysteme immer mehr an Beliebtheit. An dieser Stelle werden Sie dennoch ein ext2-Dateisystem erstellen. Unter <http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html> finden Sie Anleitungen zum Einrichten anderer Dateisysteme.

Zum Erzeugen eines ext2-Dateisystems auf der LFS-Partition führen Sie bitte das folgende Kommando aus:

```
mke2fs -v /dev/[xxx]
```

Ersetzen Sie `xxx` durch den Namen der LFS-Partition (wie zum Beispiel `hda5`).



Anmerkung

Einige Distributionen haben Zusatzfunktionen in ihre Werkzeuge zum Erzeugen von Dateisystemen (e2fsprogs) eingebaut. Dies kann später beim Booten Ihres neuen LFS zu Probleme führen, weil diese Erweiterungen in den von LFS installierten e2fsprogs nicht installiert sind. Sie könnten z. B. eine Fehlermeldung wie „unsupported filesystem features; upgrade your e2fsprogs“ erhalten. Mit dem folgenden Kommando können Sie herausfinden, ob Ihr Host-System solche zusätzlichen Funktionen verwendet:

```
debugfs -R feature /dev/[xxx]
```

Wenn die Ausgabe mehr Funktionen als `dir_index`; `filetype`; `large_file`; `resize_inode` oder `sparse_super` enthält, dann sind in Ihrem Host-System zusätzliche Erweiterungen installiert. Sie sollten spätere Probleme vermeiden indem Sie das normale Paket e2fsprogs kompilieren und die daraus resultierenden Programme zum Erzeugen des Dateisystems auf Ihrer LFS-Partition verwenden:

```
cd /tmp
tar -xjvf /Pfad/zu/den/Quellen/von/e2fsprogs-1.37.tar.bz2
cd e2fsprogs-1.37
mkdir -v build
cd build
../configure
make #ANMERKUNG: Führen Sie bitte nicht 'make install' aus!
./misc/mke2fs -v /dev/[xxx]
cd /tmp
rm -rfv e2fsprogs-1.37
```

Wenn Sie eine Swap Partition erstellt haben, müssen Sie diese mit dem untenstehenden Befehl initialisieren (dies bezeichnet man auch als formatieren). Wenn Sie eine bereits existierende Swap-Partition verwenden, muss diese nicht initialisiert werden.

```
mkswap -v /dev/[yyy]
```

Bitte ersetzen Sie `yyy` durch den Namen Ihrer Swap-Partition.

2.4. Einhängen (mounten) der neuen Partition

Nachdem Sie nun ein Dateisystem erzeugt haben, sollten Sie natürlich auch darauf zugreifen können. Dazu müssen Sie erst einen Mountpunkt wählen und es dann dort einhängen (mounten). Wir gehen davon aus, dass das Dateisystem unter `/mnt/lfs` eingehängt wird. Sie können sich aber auch jeden anderen Ordner aussuchen.

Wählen Sie nun einen Mountpunkt und tragen Sie ihn in die Umgebungsvariable `LFS` ein. Dazu können Sie diesen Befehl verwenden:

```
export LFS=/mnt/lfs
```

Als nächstes erzeugen Sie den Ordner den Sie als Mountpunkt gewählt haben und hängen das LFS-Dateisystem ein:

```
mkdir -pv $LFS
mount -v /dev/[xxx] $LFS
```

Bitte setzen Sie statt `xxx` die Bezeichnung der LFS-Partition ein.

Falls Sie sich für mehrere LFS-Partitionen entschieden haben (z. B. eine für `/` und eine andere für `/usr`), dann gehen Sie für die restlichen Partitionen gleichermaßen vor:

```
mkdir -pv $LFS
mount -v /dev/[xxx] $LFS
mkdir -v $LFS/usr
mount -v /dev/[yyy] $LFS/usr
```

Natürlich müssen Sie auch hier wieder für `xxx` und `yyy` die korrekten Bezeichnungen einsetzen.

Die Zugriffsrechte für die neue Partition sollten nicht zu restriktiv sein (wie zum Beispiel mit den Optionen `„nosuid“`, `„nodev“` oder `„noatime“`). Rufen Sie `mount` ohne Parameter auf damit Sie sehen, mit welchen Optionen Ihre LFS-Dateisysteme eingehängt wurden. Wenn Optionen wie `nosuid`, `nodev` oder `noatime` aktiviert wurden, müssen Sie die Partition erneut einhängen und diese Optionen weglassen.

Jetzt haben Sie genügend Platz zum Arbeiten geschaffen und können mit dem Herunterladen der Pakete beginnen.

Kapitel 3. Pakete und Patches

3.1. Einführung

Die folgende Liste enthält alle Pakete, die Sie für ein minimales Linux-System benötigen. Die Versionsnummern sind Versionen, von denen wir *wissen*, dass Sie funktionieren. Wenn Sie noch wenig Erfahrung mit LFS haben sollten Sie lieber keine anderen Versionen probieren. Die Anleitungen und Kommandos könnten evtl. mit neueren Versionen nicht mehr funktionieren. Oft gibt es auch gute Gründe dafür, nicht die allerneueste Version einzusetzen: zum Beispiel bei bekannten Problemen für die es noch keine Lösung gibt.

Wir können nicht für die ständige Verfügbarkeit der Download-Ressourcen garantieren. Falls sich eine Download-Adresse nach Erscheinen des Buches geändert haben sollte, nutzen Sie bitte Google oder eine andere Suchmaschine und suchen nach dem entsprechenden Paket (<http://www.google.com/>). Sollten Sie auch hier erfolglos sein, dann nutzen Sie bitte eine der alternativen Download-Möglichkeiten wie unter <http://www.linuxfromscratch.org/lfs/packages.html> beschrieben.

Sie müssen alle heruntergeladenen Pakete und Patches an einem Ort speichern, auf den Sie während der ganzen Zeit bequem zugreifen können. Außerdem benötigen Sie einen Arbeitsordner zum Entpacken und Kompilieren der Quellen. Am besten benutzen Sie den Ordner `$LFS/sources` sowohl zum Speichern der Quellen und Patches *als auch* als Arbeitsordner. Damit haben Sie alles Nötige immer auf der LFS-Partition und in allen Arbeitsschritten des Buches verfügbar.

Sie sollten folgendes Kommando als Benutzer *root* auszuführen, bevor Sie mit dem Herunterladen der Pakete beginnen:

```
mkdir -v $LFS/sources
```

Machen Sie den Ordner für jeden beschreibbar und sticky. Der „Sticky“-Modus bewirkt, dass jeweils nur der Besitzer einer Datei diese auch löschen kann, selbst wenn mehrere Benutzer Schreibrechte in dem Ordner haben. Das folgende Kommando schaltet Schreib- und Sticky-Berechtigungen ein:

```
chmod -v a+wt $LFS/sources
```

3.2. Alle Pakete

Bitte laden Sie die folgenden Pakete herunter:

- Autoconf (2.59) - 908 (KB):
<http://ftp.gnu.org/gnu/autoconf/>
- Automake (1.9.5) - 748 KB:
<http://ftp.gnu.org/gnu/automake/>
- Bash (3.0) - 1,824 KB:
<http://ftp.gnu.org/gnu/bash/>
- Bash Dokumentation (3.0) - 1,994 KB:
<http://ftp.gnu.org/gnu/bash/>
- Binutils (2.15.94.0.2.2) - 11,056 KB:
<http://www.kernel.org/pub/linux/devel/binutils/>
- Bison (2.0) - 916 KB:
<http://ftp.gnu.org/gnu/bison/>
- Bzip2 (1.0.3) - 596 KB:
<http://www.bzip.org/>
- Coreutils (5.2.1) - 4,184 KB:
<http://ftp.gnu.org/gnu/coreutils/>
- DejaGNU (1.4.4) - 852 KB:
<http://ftp.gnu.org/gnu/dejagnu/>
- Diffutils (2.8.1) - 648 KB:
<http://ftp.gnu.org/gnu/diffutils/>
- E2fsprogs (1.37) - 3,100 KB:
<http://prdownloads.sourceforge.net/e2fsprogs/>
- Expect (5.43.0) - 416 KB:
<http://expect.nist.gov/src/>
- File (4.13) - 324 KB:
<ftp://ftp.gw.com/mirrors/pub/unix/file/>



Anmerkung

Wenn Sie diese Anmerkung lesen ist File (4.13) möglicherweise nicht mehr in dieser Version verfügbar. Der Hauptdownloadserver ist dafür bekannt, alte Versionen zu löschen, sobald neuere verfügbar sind. Bitte nutzen Sie eine der alternativen Download-Adressen wie z. B. <http://www.linuxfromscratch.org/lfs/download.html#ftp>.

- Findutils (4.2.23) - 784 KB:
<http://ftp.gnu.org/gnu/findutils/>

- Flex (2.5.31) - 672 KB:
<http://prdownloads.sourceforge.net/lex/>
- Gawk (3.1.4) - 1,696 KB:
<http://ftp.gnu.org/gnu/gawk/>
- GCC (3.4.3) - 26,816 KB:
<http://ftp.gnu.org/gnu/gcc/>
- Gettext (0.14.3) - 4,568 KB:
<http://ftp.gnu.org/gnu/gettext/>
- Glibc (2.3.4) - 12,924 KB:
<http://ftp.gnu.org/gnu/glibc/>
- Glibc-Linuxthreads (2.3.4) - 236 KB:
<http://ftp.gnu.org/gnu/glibc/>
- Grep (2.5.1a) - 520 KB:
<http://ftp.gnu.org/gnu/grep/>
- Groff (1.19.1) - 2,096 KB:
<http://ftp.gnu.org/gnu/groff/>
- GRUB (0.96) - 768 KB:
<ftp://alpha.gnu.org/gnu/grub/>
- Gzip (1.3.5) - 284 KB:
<ftp://alpha.gnu.org/gnu/gzip/>
- Hotplug (2004_09_23) - 40 KB:
<http://www.kernel.org/pub/linux/utils/kernel/hotplug/>
- Iana-Etc (1.04) - 176 KB:
<http://www.sethwicklein.net/projects/iana-etc/downloads/>
- Inetutils (1.4.2) - 752 KB:
<http://ftp.gnu.org/gnu/inetutils/>
- IPRoute2 (2.6.11-050330) - 276 KB:
<http://developer.osdl.org/dev/iproute2/download/>
- Kbd (1.12) - 624 KB:
<http://www.kernel.org/pub/linux/utils/kbd/>
- Less (382) - 216 KB:
<http://ftp.gnu.org/gnu/less/>
- LFS-Bootskripte (3.2.1) - 32 KB:
<http://downloads.linuxfromscratch.org/>
- Libtool (1.5.14) - 1,604 KB:
<http://ftp.gnu.org/gnu/libtool/>
- Linux (2.6.11.12) - 35,792 KB:
<http://www.kernel.org/pub/linux/kernel/v2.6/>

- Linux-Libc-Header (2.6.11.2) - 2,476 KB:
<http://ep09.pld-linux.org/~mmazur/linux-libc-headers/>
- M4 (1.4.3) - 304 KB:
<http://ftp.gnu.org/gnu/m4/>
- Make (3.80) - 904 KB:
<http://ftp.gnu.org/gnu/make/>
- Man (1.5p) - 208 KB:
<http://www.kernel.org/pub/linux/utils/man/>
- Man-pages (2.01) - 1,640 KB:
<http://www.kernel.org/pub/linux/docs/manpages/>
- Mktmp (1.5) - 68 KB:
<ftp://ftp.mktmp.org/pub/mktmp/>
- Module-Init-Tools (3.1) - 128 KB:
<http://www.kernel.org/pub/linux/utils/kernel/module-init-tools/>
- Module-Init-Tools-Testsuite (3.1) - 34 KB:
<http://www.kernel.org/pub/linux/utils/kernel/module-init-tools/>
- Ncurses (5.4) - 1,556 KB:
<ftp://invisible-island.net/ncurses/>
- Patch (2.5.4) - 156 KB:
<http://ftp.gnu.org/gnu/patch/>
- Perl (5.8.7) - 9,628 KB:
<http://ftp.funet.fi/pub/CPAN/src/>
- Procps (3.2.5) - 224 KB:
<http://procps.sourceforge.net/>
- Psmisc (21.6) - 188 KB:
<http://prdownloads.sourceforge.net/psmisc/>
- Readline (5.0) - 1,456 KB:
<http://ftp.gnu.org/gnu/readline/>
- Sed (4.1.4) - 632 KB:
<http://ftp.gnu.org/gnu/sed/>
- Shadow (4.0.9) - 1,084 KB:
<ftp://ftp.pld.org.pl/software/shadow/>



Anmerkung

Wenn Sie diese Anmerkung lesen ist Shadow (4.0.9) möglicherweise nicht mehr in dieser Version verfügbar. Der Hauptdownloadserver ist dafür bekannt, alte Versionen zu löschen, sobald neuere verfügbar sind. Bitte nutzen Sie eine der alternativen Download-Adressen wie z. B. <http://www.linuxfromscratch.org/lfs/download.html#ftp>.

- Sysklogd (1.4.1) - 72 KB:
<http://www.infodrom.org/projects/sysklogd/download/>
- Sysvinit (2.86) - 88 KB:
<ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/>
- Tar (1.15.1) - 1,580 KB:
<http://ftp.gnu.org/gnu/tar/>
- Tcl (8.4.9) - 2,748 KB:
<http://prdownloads.sourceforge.net/tcl/>
- Texinfo (4.8) - 1,492 KB:
<http://ftp.gnu.org/gnu/texinfo/>
- Udev (056) - 476 KB:
<http://www.kernel.org/pub/linux/utils/kernel/hotplug/>
- Udev Regel-Konfiguration - 5 KB:
<http://downloads.linuxfromscratch.org/udev-config-4.rules>
- Util-linux (2.12q) - 1,344 KB:
<http://www.kernel.org/pub/linux/utils/util-linux/>
- Vim (6.3) - 3,620 KB:
<ftp://ftp.vim.org/pub/vim/unix/>
- Vim (6.3) Sprachdateien (optional) - 540 KB:
<ftp://ftp.vim.org/pub/vim/extra/>
- Zlib (1.2.3) - 415 KB:
<http://www.zlib.net/>

Gesamtgröße der Pakete: 146 MB

3.3. Erforderliche Patches

Zusätzlich brauchen Sie auch einige Patches. Diese beheben z. B. kleine Fehler, die vom jeweiligen Betreuer des Pakets noch nicht behoben wurden, oder beinhalten Modifikationen und Anpassungen an unser LFS. Die folgenden Patches werden zum Erstellen von LFS benötigt:

- Bash Avoid Wcontinued Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/bash-3.0-avoid_WCONTINUED-1.patch
- Verschiedene Fehlerbereinigungen für Bash - 23 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/bash-3.0-fixes-3.patch>
- Patch zum Bauen von Binutils von einem Rechner mit Gcc4 - 2 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/binutils-2.15.94.0.2.2-gcc4-1.patch>
- Bzip2 Dokumentations-Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/bzip2-1.0.3-install_docs-1.patch
- Bzip2 Bzgrep Sicherheits-Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/bzip2-1.0.3-bzgrep_security-1.patch
- Coreutils Suppress Uptime, Kill, Su Patch - 15 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/coreutils-5.2.1-suppress_uptime_kill_su-1.patch
- Coreutils Uname Patch - 4 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/coreutils-5.2.1-uname-2.patch>
- Expect Spawn Patch - 7 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/expect-5.43.0-spawn-1.patch>
- Flex Brokenness Patch - 156 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/flex-2.5.31-debian_fixes-3.patch
- GCC Linkonce Patch - 12 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/gcc-3.4.3-linkonce-1.patch>
- GCC No-Fixincludes Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/gcc-3.4.3-no_fixincludes-1.patch
- GCC Specs Patch - 14 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/gcc-3.4.3-specs-2.patch>
- Glibc Rtdl Search Dirs Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/glibc-2.3.4-rtdl_search_dirs-1.patch
- Glibc Fix Testsuite Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/glibc-2.3.4-fix_test-1.patch
- Glibc TLS Assertion Patch - 6 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/glibc-2.3.4-tls_assert-1.patch
- Gzip Sicherheits-Patch - 2 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/gzip-1.3.5-security_fixes-1.patch
- Inetutils Kernel Headers Patch - 1 KB:

http://www.linuxfromscratch.org/patches/lfs/6.1.1/inetutils-1.4.2-kernel_headers-1.patch

- Inetutils No-Server-Man-Pages Patch - 4 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/inetutils-1.4.2-no_server_man_pages-1.patch
- Mktmp Tempfile Patch - 3 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/mktemp-1.5-add_tempfile-2.patch
- Perl Libc Patch - 1 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/perl-5.8.7-libc-1.patch>
- Readline Fixes Patch - 7 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/readline-5.0-fixes-1.patch>
- Sysklogd Fixes Patch - 27 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/sysklogd-1.4.1-fixes-1.patch>
- Tar Sparse Fix Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/tar-1.15.1-sparse_fix-1.patch
- Texinfo Tempfile Fix Patch - 2 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/texinfo-4.8-tempfile_fix-1.patch
- Util-linux Cramfs Patch - 3 KB:
<http://www.linuxfromscratch.org/patches/lfs/6.1.1/util-linux-2.12q-cramfs-1.patch>
- Util-linux Umount Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/util-linux-2.12q-umount_fix-1.patch
- Vim Sicherheits-Patch - 8 KB:
http://www.linuxfromscratch.org/patches/lfs/6.1.1/vim-6.3-security_fix-2.patch

Die LFS-Gemeinschaft hat noch zahlreiche weitere Patches erstellt. Die meisten beheben kleine Probleme oder schalten Funktionen ein, die in der Voreinstellung abgeschaltet sind. Durchstöbern Sie ruhig die Patch-Datenbank unter <http://www.linuxfromscratch.org/patches/> und laden Sie zusätzliche Patche herunter.

Kapitel 4. Abschluss der Vorbereitungen

4.1. Die Variable \$LFS

Bei der Arbeit mit dem Buch werden Sie häufig mit der Umgebungsvariable `LFS` zu tun haben. Diese Variable sollte immer definiert sein und den Mountpunkt enthalten, den Sie für die LFS-Partition ausgewählt haben. Überprüfen Sie mit dem folgenden Kommando bitte nochmals, ob `LFS` korrekt gesetzt ist:

```
echo $LFS
```

Die Ausgabe muss dem Pfad zu Ihrer LFS-Partition entsprechen! Wenn Sie unserem Beispiel gefolgt sind lautet der Pfad `/mnt/lfs`. Wenn hier etwas nicht stimmt können Sie die Variable jederzeit neu setzen:

```
export LFS=/mnt/lfs
```

Durch diese Variable haben Sie den Vorteil, dass Sie ein Kommando wie z. B. `mkdir $LFS/tools` genau so eingeben können wie Sie es im Buch lesen. Während die Shell den Befehl verarbeitet, wird sie „`$LFS`“ durch den echten Wert `„/mnt/lfs“` ersetzen.

Wenn Sie Ihre Arbeitsumgebung verlassen haben, müssen Sie anschließend den Inhalt von `$LFS` nochmals prüfen. Das gilt auch, wenn Sie z. B. „`su`“ zu `root` oder einem anderen Benutzer ausführen.

4.2. Erstellen des Ordners `$LFS/tools`

Alle kompilierten Programme aus Kapitel 5 werden unter `$LFS/tools` installiert. Dadurch werden sie von den Programmen getrennt, die später in Kapitel 6 installiert werden. Die hier kompilierten Programme sind nur übergangsweise Hilfsmittel und sollen nicht Teil des endgültigen LFS-Systems werden. Durch die Installation in einen gesonderten Ordner lassen sie sich später leichter wieder entfernen. Außerdem wird so sichergestellt, dass die Programme nicht versehentlich in Ihrem produktiven Host-System enden (in Kapitel 5 könnte das sehr leicht passieren).

Erstellen Sie den Ordner indem Sie als *root* dieses Kommando ausführen:

```
mkdir -v $LFS/tools
```

Im nächsten Schritt erstellen Sie auf Ihrem *Host-System* einen symbolischen Link nach `/tools`. Er zeigt auf den Ordner, den Sie gerade auf der LFS-Partition erstellt haben. Führen Sie dieses Kommando als *root* aus:

```
ln -sv $LFS/tools /
```



Anmerkung

Das obige Kommando ist in dieser Form korrekt; der Befehl **ln** hat verschiedene Syntax-Varianten — bitte lesen Sie erst **info coreutils ln** und `ln(1)` bevor Sie einen vermeintlichen Fehler berichten.

Dieser symbolische Link ermöglicht uns, die Toolchain so zu kompilieren, dass sie immer `/tools` referenziert. Das hat für uns den Vorteil, dass Compiler, Assembler und Linker sowohl in diesem Kapitel (in dem Sie noch einige Programme vom Host-System benutzen) *als auch* im nächsten Kapitel (wenn Sie in die LFS-Partition „chroot'ed“ haben) funktionieren werden. Das liegt daran, dass die Programme immer den gleichen Pfad benutzen können.

4.3. Hinzufügen des LFS-Benutzers

Als *root* eingeloggt können selbst kleine Fehler ein System beschädigen oder gar zerstören. Daher sollten Sie die Pakete in diesem Kapitel mit Hilfe eines unprivilegierten Benutzers kompilieren. Natürlich können Sie Ihren bisherigen Benutzernamen dazu verwenden, aber das Bereitstellen einer sauberen Arbeitsumgebung ist leichter, wenn Sie dazu den Benutzer *lfs* in der ebenfalls neuen Gruppe *lfs* anlegen und diesen für den ganzen Installationsvorgang benutzen. Bitte führen Sie als *root* dieses Kommando aus, um die neue Gruppe und den Benutzer anzulegen:

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
```

Die Bedeutung der Kommandozeilen-Parameter:

-s /bin/bash

Dies macht die **bash** zur voreingestellten Shell für den Benutzer *lfs*.

-g lfs

Dieser Parameter macht den neuen Benutzer zum Mitglied der Gruppe *lfs*.

-m

Dadurch wird der Persönliche Ordner für *lfs* gleich mitangelegt.

-k /dev/null

Dieser Parameter verhindert das Kopieren der Dateien aus einem Skeleton-Ordner (Voreinstellung ist */etc/skel*). Als Quelle für den Skeleton-Ordner wird einfach das Null-Gerät eingestellt.

lfs

Dies ist der Name der erzeugten Gruppe und Benutzer.

Wenn Sie als *root* angemeldet sind und zum Benutzer *lfs* wechseln, brauchen Sie dafür kein Passwort. Wenn Sie sich allerdings mit dem Benutzer *lfs* richtig anmelden möchten, müssen Sie dem Benutzer zuerst ein Passwort zuweisen:

```
passwd lfs
```

Geben Sie *lfs* Vollzugriff auf *\$LFS/tools*. Dazu machen Sie *lfs* am besten zum Besitzer des Ordners:

```
chown -v lfs $LFS/tools
```

Wenn Sie, wie vorgeschlagen, einen extra Arbeitsordner eingerichtet haben, dann geben Sie dem Benutzer *lfs* auch dort die Besitzrechte:

```
chown -v lfs $LFS/sources
```

Als nächstes melden Sie sich bitte als *lfs* an. Dazu können Sie eine virtuelle Konsole, den Display-Manager oder das folgende Kommando verwenden:

```
su - lfs
```

Das „-“ weist **su** an, eine Login-Shell anstelle einer Nicht-Login-Shell zu starten. Der Unterschied zwischen den beiden Arten wird in *bash(1)* und **info bash** erklärt.

4.4. Vorbereiten der Arbeitsumgebung

Um Ihre Arbeitsumgebung für die weiteren Schritte vorzubereiten erstellen Sie zwei Dateien für die **bash**. Geben Sie als Benutzer *lfs* das folgende Kommando ein, um die neue Datei `.bash_profile` zu erzeugen:

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' /bin/bash
EOF
```

Wenn Sie sich als Benutzer *lfs* anmelden, ist die erste Shell üblicherweise eine *Login-Shell*. Diese liest erst die Datei `/etc/profile` Ihres Host-Systems ein (sie enthält meistens Einstellungen zu Umgebungsvariablen), und danach `.bash_profile`. Das Kommando **exec env -i.../bin/bash** in der zweiten Datei ersetzt die laufende Shell durch eine neue mit einer vollständig leeren Umgebung, mit Ausnahme der Variablen `HOME`, `TERM` und `PS1`. Dadurch wird sichergestellt, dass keine ungewollten und potentiell gefährlichen Umgebungsvariablen vom Host-System auf unsere Arbeitsumgebung Einfluss nehmen können. Die hier angewendete Technik mag ein wenig befremdlich wirken, führt aber zu unserem Ziel: einer absolut reinen Arbeitsumgebung.

Die neue Instanz der Shell ist eine *Nicht-Login-Shell*; diese liest weder `/etc/profile` noch `.bash_profile` ein. Stattdessen liest sie die Datei `.bashrc`, erstellen Sie sie nun:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL PATH
EOF
```

Das Kommando **set +h** schaltet die Hash-Funktion der **bash** ab. Normalerweise ist das sogenannte Hashing der Bash eine nützliche Funktion—**Bash** benutzt eine Hash-Tabelle, um sich die Pfade zu ausführbaren Dateien zu merken und vermeidet auf diese Weise ein ständiges Durchsuchen aller Ordner. Beim Bau von LFS müssen Sie jedoch alle neu installierten Werkzeuge sofort nutzen können. Durch Abschalten der Hash-Funktion wird für „interaktive“ Kommandos (**make**, **patch**, **sed**, **cp** und so weiter) immer die neueste verfügbare Version benutzt.

Das Setzen der Dateierzeugungs-Maske (`umask`) auf 022 bewirkt, dass neu erzeugte Dateien nur durch ihren Besitzer beschreibbar sind, aber für alle anderen les- und ausführbar (wenn der Systemaufruf `open(2)` die üblichen Datei-Modi benutzt, werden alle neu erzeugten Dateien die Rechte 644 und Ordner die Rechte 755 erhalten).

Die Variable `LFS` sollte natürlich auf den von Ihnen gewählten Mountpunkt der LFS-Partition gesetzt sein.

Die Variable `LC_ALL` beeinflusst die Lokalisierung einiger Programme, so dass deren Ausgaben den Konventionen des entsprechenden Landes folgen. Wenn Ihr Host-System eine ältere Glibc-Version als 2.2.4 verwendet, könnte es Probleme geben, wenn `LC_ALL` nicht auf „POSIX“ oder „C“ gesetzt ist. Durch Setzen von `LC_ALL` auf „POSIX“ oder „C“ (die beiden Werte haben die gleiche Wirkung) sollte es beim Hin- und Herwechseln in der chroot-Umgebung keine Probleme geben.

Durch das Voranstellen von `/tools/bin` an die Umgebungsvariable `PATH` werden alle in Kapitel 5 installierten Programme beim Durchsuchen der Pfade als erstes gefunden und von der Shell sofort benutzt. Zusammen mit dem Abschalten der Hash-Funktion der **Bash** wird so das Risiko minimiert, dass eventuell alte

Programme vom Host-System benutzt werden, obwohl schon eine neuere Version aus Kapitel 5 auf dem System existiert.

Um die Arbeitsumgebung endgültig fertig zu stellen, muss das soeben erzeugte Profil eingelesen werden:

```
source ~/.bash_profile
```

4.5. Informationen zu SBUs

Die meisten Leser möchten gerne vorher wissen, wie lange das Kompilieren und Installieren der Pakete dauert. Linux From Scratch wird aber auf so unterschiedlichen Systemen gebaut, dass es unmöglich ist, echte, auch nur annähernd akkurate Zeiten anzugeben: Das größte Paket (Glibc) braucht auf schnellen Maschinen nicht einmal 20 Minuten, aber auf langsamen Maschinen drei Tage oder mehr. Anstatt Ihnen also Zeiteinheiten zu nennen, haben wir uns für die Standard Binutils Unit entschieden (Abgekürzt: SBU).

Das funktioniert so: Das erste zu kompilierende Paket ist Binutils in Kapitel 5. Die Zeit, die Ihr Computer zum Kompilieren dieses Pakets braucht, entspricht einer „Standard Binutils Unit“ bzw. „SBU“. Alle weiteren Kompilierzeiten werden relativ zu dieser Zeit angegeben.

Nehmen Sie als Beispiel ein Paket mit 4,5 SBU. Wenn das Kompilieren der Binutils 10 Minuten gedauert hat, dann dauert es *ungefähr* 45 Minuten, um das Beispieldpaket zu bauen. Glücklicherweise sind die meisten Kompilierzeiten kürzer als die der Binutils.

Grundsätzlich sind SBUs relativ ungenau weil sie auf vielen Faktoren basieren, inklusive der GCC-Version des Host-Systems. Auf Mehrprozessormaschinen können SBUs sogar noch ungenauer sein. SBUs sollen Ihnen eine ungefähre Vorstellung davon geben, wieviel Zeit das Installieren eines Pakets benötigt. Die Angaben können allerdings unter Umständen stark abweichen.

Wenn Sie sich aktuelle Zeitangaben für bestimmte Computerkonfigurationen ansehen möchten, schauen Sie doch mal unter <http://www.linuxfromscratch.org/~bdubbs/>.

4.6. Über die Testsuites

Die meisten Pakete enthalten auch eine Testsuite. Es ist prinzipiell immer eine gute Idee, eine solche Testsuite für neu kompilierte Programme auch durchlaufen zu lassen. So stellen Sie sicher, dass alles korrekt kompiliert wurde. Wenn eine Testsuite alle ihre Tests erfolgreich durchläuft, können Sie ziemlich sicher sein, dass das Paket so funktioniert, wie es der Entwickler vorgesehen hat. Dennoch ist das natürlich kein Garant für absolute Fehlerfreiheit.

Manche Tests sind wichtiger als andere. So zum Beispiel die Tests der Toolchain-Pakete—GCC, Binutils und Glibc (die C Bibliothek)—sind von höchster Bedeutung, weil diese Pakete eine absolut zentrale Rolle für die Funktion des gesamten Systems spielen. Aber seien Sie gewarnt: die Testsuites von GCC und Glibc brauchen sehr viel Zeit, vor allem auf langsamer Hardware. Dennoch wird dringend empfohlen, sie durchlaufen zu lassen!



Anmerkung

Die Erfahrung hat gezeigt, dass man in Kapitel 5 vom Durchlaufen lassen der Testsuites im Grunde nicht viel gewinnt. Das Host-System hat immer einen gewissen Einfluss auf die Tests in dem Kapitel und das verursacht seltsame und unerklärliche Fehler. Und nicht nur das, die in Kapitel 5 erzeugten Werkzeuge sind ohnehin nur temporär und werden später wieder gelöscht. Daher empfehlen wir Ihnen, die Testsuites in Kapitel 5 *nicht* durchlaufen zu lassen. Die Anleitungen dafür sind dennoch vorhanden, um Testern und Entwicklern eine Hilfe zu sein, aber für alle anderen Anwender sind sie nur optional.

Ein weit verbreitetes Problem beim Durchlaufen der Testsuites von Binutils und GCC sind zu wenig zur Verfügung stehende Pseudo-Terminals (PTYs). Ein typisches Symptom dafür sind ungewöhnlich viele fehlgeschlagene Tests. Das kann verschiedene Ursachen haben. Die häufigste Ursache ist, dass das `devpts`-Dateisystem des Host-Systems nicht funktioniert. Dies wird später in Kapitel 5 ausführlicher behandelt.

Manchmal verursachen Testsuites eines Pakets auch falschen Alarm. Sehen Sie im LFS-Wiki unter <http://www.linuxfromscratch.org/lfs/build-logs/6.1.1/> nach und prüfen Sie, ob diese Fehler normal sind. Das gilt für alle Tests im gesamten Buch.

Kapitel 5. Erstellen eines temporären Systems

5.1. Einführung

In diesem Kapitel werden Sie ein Minimal-Linux kompilieren und installieren. Das System wird gerade genug Werkzeuge beinhalten, um in Kapitel 6 mit dem Bau des endgültigen LFS beginnen zu können. Wir verzichten hierbei weitestgehend auf jeglichen Komfort.

Das Erstellen des Minimal-Systems erfolgt in zwei Schritten: Zuerst erzeugen Sie eine brandneue, Host-unabhängige Toolchain (Compiler, Assembler, Linker und Bibliotheken und ein paar nützliche Werkzeuge). Mit Hilfe der Toolchain können dann im weiteren Verlauf die essentiellen Werkzeuge kompiliert werden.

Die in diesem Kapitel kompilierten Dateien werden im Ordner `$LFS/tools` installiert und sind damit von den restlichen Dateien des Systems sauber getrennt. Die hier kompilierten Programme sind schließlich nur temporär und sollen nicht mit in unser endgültiges LFS-System einfließen.



Wichtig

Die Anweisungen zum Kompilieren setzen voraus, dass Sie **Bash** als Shell einsetzen. Bevor Sie ein Paket installieren, müssen Sie das jeweilige Tar-Archiv bereits als Benutzer *lfs* entpackt und mit **cd** in den Ordner gewechselt haben. Danach können Sie die jeweilige Installationsanleitung durcharbeiten.

Einige der Pakete werden vor dem Kompilieren gepatcht, aber nur um ein potentiell Problem zu umgehen. Meist wird ein Patch sowohl in diesem als auch im folgenden Kapitel benötigt, manchmal aber auch nur in einem von beiden. Machen Sie sich keine Gedanken, wenn die Installationsanweisungen für einen Patch zu fehlen scheinen. Außerdem werden Sie manchmal beim Installieren eines Patches Warnungen über *offset* oder *fuzzy* sehen. Diese Warnungen sind nicht wichtig, der Patch wird dennoch sauber installiert.

Beim Kompilieren vieler Pakete werden Sie alle möglichen Compiler-Warnungen auf dem Bildschirm bemerken. Das ist normal und kann einfach ignoriert werden. Es handelt sich eben nur um Warnungen—meistens aufgrund der Verwendung veralteter (aber dennoch korrekter) C- oder C++-Syntax. Die C-Standards haben sich im Laufe der Zeit oft verändert, und einige Pakete benutzen immer noch alte Standards, aber das ist kein wirkliches Problem.



Wichtig

Solange nichts anderes angegeben wird, sollten Sie die Quell- und Kompilierordner jedesmal nach dem Installieren eines Pakets löschen. Das Entfernen der Quellen verhindert mögliche Fehlkonfigurationen, falls ein Paket später noch einmal installiert werden muss. Nur bei drei Paketen müssen Sie die Quell- und Kompilierordner für eine Weile aufbewahren, denn ihr Inhalt wird später noch benötigt. An den Stellen finden Sie einen entsprechenden Hinweis, übersehen Sie sie nicht.

Bevor Sie fortfahren, stellen Sie bitte mit folgendem Kommando sicher, dass die `LFS`-Umgebungsvariable korrekt gesetzt ist:

```
echo $LFS
```

Die Ausgabe muss den Pfad zum Mountpunkt Ihrer LFS-Partition anzeigen. Wenn Sie unserem Beispiel gefolgt sind, sollte er `/mnt/lfs` lauten.

5.2. Technische Anmerkungen zur Toolchain

Dieser Abschnitt soll Ihnen einige technische Details zum gesamten Kompilier- und Installationsprozess erläutern. Sie müssen nicht alles in diesem Abschnitt sofort verstehen, das Meiste ergibt sich von selbst sobald Sie die ersten Pakete installiert haben. Scheuen Sie sich nicht, zwischendurch noch einmal hierhin zurückzublättern und nachzulesen wenn etwas unklar ist.

In Kapitel 5 soll eine gut funktionierende temporäre Arbeitsumgebung erschaffen werden, in die Sie sich später abkapseln und von wo aus Sie in Kapitel 6 ohne Schwierigkeiten ein sauberes endgültiges LFS-System erstellen können. Sie werden sich so weit wie möglich vom Host-System abschotten und eine in sich geschlossene Toolchain erzeugen. Bitte beachten Sie, dass der gesamte Vorgang dafür ausgelegt ist, die Risiken für neue Leser zu minimieren und gleichzeitig den Lerneffekt zu maximieren.



Wichtig

Bevor Sie fortfahren, sollten Sie den Namen der Plattform kennen, auf der Sie LFS installieren; diesen bezeichnet man oft auch als das *Ziel-Tripplet*. Für die meisten Leser wird das Ziel-Tripplet zum Beispiel *i686-pc-linux-gnu* sein. Sie können Ihr Ziel-Tripplet herauszufinden, indem Sie das Skript **config.guess** auszuführen; es wird mit den Quellen vieler Pakete mitgeliefert. Entpacken Sie die Binutils-Quellen und führen Sie das Skript aus: **./config.guess**. Notieren Sie die Ausgabe.

Auch den Namen des *dynamischen Linkers* für Ihre Plattform sollten Sie kennen (manchmal wird der Linker auch als *dynamischer Lader* bezeichnet). Bitte verwechseln Sie den dynamischen Linker nicht mit dem Standard-Linker **ld** aus dem Paket Binutils. Der dynamische Linker kommt mit Glibc und seine Aufgabe ist es, die von einem Programm benötigten gemeinsamen Bibliotheken zu finden und zu laden, das Programm zur Ausführung vorzubereiten und schließlich das Programm selbst auszuführen. Im Regelfall wird der Name des dynamischen Linkers `ld-linux.so.2` sein. Für weniger gängige Systeme könnte der Name auch `ld.so.1` sein und auf neueren 64-Bit-Plattformen könnte er sogar völlig verschieden sein. Sie müssten den Namen Ihres dynamischen Linkers herausfinden können, wenn Sie auf Ihrem Host-System in den Ordner `/lib` schauen. Um wirklich sicher zu gehen, können Sie eine beliebige Binärdatei auf Ihrem Host-System überprüfen: **readelf -l <Name einer Binärdatei> | grep interpreter**. Notieren Sie die Ausgabe. Eine Referenz, die alle Plattformen abdeckt, finden Sie in der Datei `shlib-versions` im Basisordner des Glibc-Quellordners.

Hier ein paar technische Anmerkungen zum Kompiliervorgang in Kapitel 5:

- Der Kompiliervorgang ist im Grunde ähnlich wie Cross-Kompilieren. Dabei funktionieren Programme im selben Prefix in Kooperation und benutzen dazu ein wenig GNU-„Magie“.
- Durch vorsichtiges Anpassen des Suchpfades für den Standard-Linker erreichen Sie, dass Programme nur gegen die gewünschten Bibliotheken gelinkt werden.
- Durch vorsichtiges Anpassen von **gcc**'s `specs`-Datei teilen Sie dem Compiler mit, welcher Dynamische Linker verwendet wird.

Als erstes wird Binutils installiert, da sowohl GCC als auch Glibc beim Durchlaufen des **configure**-Skriptes einige Tests zum Assembler und Linker durchführen und auf dem Ergebnis basierend bestimmte Funktionen

ein- bzw. ausschalten. Das ist wichtiger als man zunächst denken mag. Ein falsch eingerichteter GCC oder Glibc kann zu Fehlern in der Toolchain führen, die erst am Ende der Installation des LFS-Systems bemerkt werden. Zum Glück weisen Fehlschläge beim Durchlaufen der Testsuites im Regelfall auf solche Probleme hin, bevor zuviel Zeit vergeudet wird.

Binutils installiert seinen Assembler an zwei Stellen, `/tools/bin` und `/tools/$ZIEL_TRIPPLET/bin`. In Wirklichkeit sind die Programme an der einen Stelle mit denen an der anderen durch einen harten Link verknüpft. Ein wichtiger Aspekt des Linkers ist seine Suchreihenfolge für Bibliotheken. Genaue Informationen erhalten Sie mit `ld` und dem Parameter `--verbose`. Zum Beispiel: `ld --verbose | grep SEARCH` gibt die aktuellen Suchpfade und ihre Reihenfolge aus. Sie können sehen, welche Dateien tatsächlich von `ld` verlinkt werden, indem Sie ein Dummy-Programm kompilieren und den Parameter `--verbose` angeben. Zum Beispiel: `gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded` zeigt, dass alle Dateien beim Linken erfolgreich geöffnet werden konnten.

Das nächste zu installierende Paket ist GCC. Während des Durchlaufs von `configure` sehen Sie zum Beispiel:

```
checking what assembler to use...
    /tools/i686-pc-linux-gnu/bin/as
checking what linker to use... /tools/i686-pc-linux-gnu/bin/ld
```

Das ist aus den oben genannten Gründen wichtig. Hier wird auch deutlich, dass GCC's `configure`-Skript nicht die `PATH`-Ordner durchsucht, um herauszufinden, welche Werkzeuge verwendet werden sollen. Dennoch werden beim tatsächlichen Ausführen von `gcc` nicht unbedingt die gleichen Suchpfade verwendet. Welchen Standard-Linker `gcc` wirklich verwendet, kann man mittels `gcc -print-prog-name=ld` herausfinden.

Detaillierte Informationen erhält man von `gcc`, indem man den Parameter `-v` beim Kompilieren eines Dummy-Programmes übergibt. `gcc -v dummy.c` zum Beispiel gibt Informationen über den Präprozessor, Komilierungs- und Assemblierungsphasen inklusive `gcc`'s Suchpfaden und der Reihenfolge aus.

Das nächste zu installierende Paket ist Glibc. Die wichtigsten Überlegungen zum Kompilieren von Glibc beschäftigen sich mit dem Compiler, Binutils und den Kernel-Headern. Der Compiler ist normalerweise kein Problem, weil Glibc immer den `gcc` nimmt, der in den `PATH`-Ordnern gefunden wird. Die Binutils und die Kernel-Header können da schon etwas schwieriger sein. Daher gehen wir kein Risiko ein und benutzen die verfügbaren `configure`-Optionen, um die korrekten Entscheidungen zu erzwingen. Nach dem Durchlauf von `./configure` können Sie den Inhalt von `config.make` im Ordner `glibc-build` nach den Details durchsuchen. Sie werden ein paar interessante Dinge finden, wie zum Beispiel `CC="gcc -B/tools/bin/"` zum Kontrollieren der verwendeten Binutils, oder die Parameter `-nostdinc` und `-isystem` zum Kontrollieren des Suchpfades des Compilers. Diese Besonderheiten heben einen wichtigen Aspekt von Glibc hervor—Sie ist kompiliertechnisch gesehen eigenständig und nicht von Voreinstellungen der Toolchain abhängig.

Nach der Installation von Glibc nehmen Sie noch ein paar Anpassungen vor; dadurch stellen Sie sicher, dass Suchen und Verlinken nur innerhalb unseres Prefix `/tools` stattfindet. Sie installieren einen angepassten `ld`, welcher einen fest angegebenen Suchpfad auf `/tools/lib` hat. Dann bearbeiten Sie die `specs`-Datei von `gcc` so, dass sie auf den neuen Dynamischen Linker in `/tools/lib` verweist. Der letzte Schritt ist entscheidend für den gesamten Ablauf. Wie oben bereits angemerkt, wird ein fest eingestellter Pfad zum Dynamischen Linker in jeder ausführbaren ELF-Datei eingebettet. Sie können das überprüfen, indem Sie dieses Kommando ausführen: `readelf -l <Name der ausführbaren Datei> | grep interpreter`. Durch das Anpassen der `specs`-Datei von `gcc` stellen wir sicher, dass jedes von nun an kompilierte Programm bis zum Ende des Kapitels unseren neuen Dynamischen Linker in `/tools/lib` benutzt.

Weil unbedingt der neue Linker verwendet werden muss, wird der Specs-Patch auch im zweiten Durchlauf von GCC angewendet. Hierbei darf kein Fehler passieren, denn sonst würden die GCC-Programme selbst den Linker aus `/lib` im Host-System verwenden. Eine saubere Trennung vom Host-System wäre dann nicht mehr gegeben und unser Ziel wäre verfehlt.

Im zweiten Durchlauf der Binutils können Sie den configure-Parameter `--with-lib-path` benutzen, um den Bibliotheksuchpfad von **ld** zu kontrollieren. Von diesem Punkt an ist die Toolchain unabhängig. Die verbleibenden Pakete aus Kapitel 5 kompilieren alle mit der neuen Glibc in `/tools` und alles ist in Ordnung.

Aufgrund ihrer bereits erwähnten eigenständigen Natur ist die Glibc das erste wichtige Paket, das Sie nach dem Eintreten in die chroot-Umgebung in Kapitel 6 installieren. Wenn die Glibc erstmal nach `/usr` installiert ist, werden Sie schnell ein paar Voreinstellungen in der Toolchain ändern und dann schreiten Sie mit dem Erstellen des endgültigen LFS-Systems fort.

5.3. Binutils-2.15.94.0.2.2 - Durchlauf 1

Binutils ist eine Sammlung von Software-Entwicklungswerkzeugen. Dazu gehören zum Beispiel Linker, Assembler und weitere Programme für die Arbeit mit Objektdateien.

Geschätzte Kompilierzeit: 1.0 SBU

Ungefähr benötigter Festplattenplatz: 179 MB

Die Installation ist abhängig von: Bash, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed und Texinfo

5.3.1. Installation von Binutils

Es ist wichtig, dass Binutils als erstes Paket kompiliert wird, weil Glibc und GCC verschiedene Tests bezüglich Linker und Assembler durchführen und erst daraufhin bestimmte Funktionen aktivieren.

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch `-march` und `-mcpu`.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. `CFLAGS` und `CXXFLAGS`) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Wenn Sie LFS auf einem Host mit Gcc-4 oder neuer erstellen, dann müssten Sie den ersten Durchlauf von Binutils patchen, da er sonst auf dem Host-System nicht kompiliert.

```
patch -Np1 -i ../binutils-2.15.94.0.2.2-gcc4-1.patch
```

Die Dokumentation zu Binutils empfiehlt, Binutils außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../binutils-build
cd ../binutils-build
```



Anmerkung

Wenn die im Buch angegebenen SBU-Werte einen Nutzen haben sollen, müssen Sie nun die Zeit messen, die Sie zum Kompilieren von Binutils benötigen. Dies ist mit dem folgenden Kommando relativ einfach: `time { ./configure ... && make && make install; }`.

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.15.94.0.2.2/configure --prefix=/tools --disable-nls
```

Die Bedeutung der configure-Parameter:

`--prefix=/tools`

Dadurch wird das configure-Skript die Binutils-Programme für die Installation nach `/tools` vorbereiten.

`--disable-nls`

Deaktiviert die Internationalisierung; `i18n` wird für die temporären Werkzeuge nicht benötigt.

Fahren Sie mit dem Kompilieren des Pakets fort:

```
make
```

Der Kompiliervorgang ist nun abgeschlossen. Normalerweise würden Sie nun die Testsuite durchlaufen lassen, aber in diesem frühen Stadium ist die Testsuite-Umgebung (Tcl, Expect und DejaGNU) noch nicht verfügbar. Außerdem macht es wenig Sinn, die Tests nun laufen zu lassen, denn die Programme aus dem ersten Durchlauf werden sehr bald durch die aus dem zweiten Durchlauf ersetzt.

Installieren Sie das Paket:

```
make install
```

Bereiten Sie nun den Linker auf die späteren „Anpassungen“ vor:

```
make -C ld clean  
make -C ld LIB_PATH=/tools/lib
```

Die Bedeutung der make-Parameter:

```
-C ld clean
```

Dies weist das Programm make an, alle kompilierten Dateien im Unterordner `ld` zu löschen.

```
-C ld LIB_PATH=/tools/lib
```

Dieser Parameter kompiliert alles im Unterordner `ld` erneut. Die Angabe der Makefile-Variable `LIB_PATH` auf der Kommandozeile überschreibt den Standardwert und zeigt auf den temporären Ordner `tools`. Der Wert dieser Variable gibt den Standard-Bibliotheksuchpfad für den Linker an. Sie werden später in diesem Kapitel sehen, wie diese Vorbereitung zur Anwendung kommt.



Warnung

Entfernen Sie die Kompilier- und Quellordner von Binutils noch nicht. Sie benötigen Sie später in ihrem jetzigen Zustand.

Details zu diesem Paket finden Sie in Abschnitt 6.13.2, „Inhalt von Binutils“

5.4. GCC-3.4.3 - Durchlauf 1

Das Paket GCC enthält die GNU-Compiler-Sammlung. Darin sind die C- und C++-Compiler enthalten.

Geschätzte Kompilierzeit: 4.4 SBU

Ungefähr benötigter Festplattenplatz: 219 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed und Texinfo

5.4.1. Installation von GCC

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch *-march* und *-mcpu*.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. *CFLAGS* und *CXXFLAGS*) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Die Dokumentation zu GCC empfiehlt, GCC außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-3.4.3/configure --prefix=/tools \
  --libexecdir=/tools/lib --with-local-prefix=/tools \
  --disable-nls --enable-shared --enable-languages=c
```

Die Bedeutung der configure-Parameter:

--with-local-prefix=/tools

Der Sinn dieses Parameters ist es, */usr/local/include* aus dem Suchpfad von **gcc** zu entfernen. Dies ist nicht absolut zwingend erforderlich, jedoch sollen mögliche Einflüsse aus dem Host-System vermieden werden, daher ist dieser Parameter hier durchaus empfehlenswert.

--enable-shared

Dieser Parameter ermöglicht das Kompilieren von *libgcc_s.so.1* und *libgcc_eh.a*. Die alleinige Existenz von *libgcc_eh.a* stellt sicher, dass das configure-Skript für Glibc (das nächste zu kompilierende Paket) korrekte Ergebnisse erzielt.

--enable-languages=c

Dieser Parameter stellt sicher, dass nur der C-Compiler erzeugt wird.

Fahren Sie mit dem Kompilieren des Pakets fort:

```
make bootstrap
```

Die Bedeutung der make-Parameter:

bootstrap

Dieses make-Target kompiliert GCC nicht einfach nur, sondern kompiliert gleich mehrmals. GCC benutzt die im ersten Durchlauf erzeugten Programme, um sich damit im zweiten Durchlauf selbst zu kompilieren. Darauf folgt der dritte Kompilierungsvorgang. Abschließend werden die Ergebnisse des zweiten und dritten Kompilierungsvorgangs verglichen, um sicherzustellen, dass GCC sich selbst problemlos kompilieren konnte. Das bedeutet normalerweise, dass alles korrekt verlaufen ist.

Der Kompilierungsvorgang ist nun abgeschlossen. Normalerweise würden Sie nun die Testsuite durchlaufen lassen, aber in diesem frühen Stadium ist die Testsuite-Umgebung (Tcl, Expect und DejaGNU) noch nicht verfügbar. Außerdem macht es wenig Sinn, die Tests nun laufen zu lassen, weil die Programme aus dem ersten Durchlauf sehr bald durch die aus dem zweiten Durchlauf ersetzt werden.

Installieren Sie das Paket:

```
make install
```

Zum Abschluss erstellen Sie noch einen symbolischen Link. Viele Programme rufen das Programm **cc** anstelle von **gcc** auf. Dadurch werden Programme generisch gehalten und sind auf verschiedenen Unix-Systemen lauffähig. Denn nicht jedes System hat den GNU C-Compiler installiert. Der Aufruf von **cc** lässt dem Administrator die Wahl, welchen C-Compiler er installieren möchte, solange ein symbolischer Link auf den echten Compiler verweist:

```
ln -vs gcc /tools/bin/cc
```

Details zu diesem Paket finden Sie in Abschnitt 6.14.2, „Inhalt von GCC“

5.5. Linux-Libc-Header-2.6.11.2

Das Paket Linux-Libc-Header enthält die „bereinigten“ Header-Dateien des Linux-Kernels

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 26.9 MB

Die Installation ist abhängig von: Coreutils

5.5.1. Installation von Linux-Libc-Header

Über Jahre hinweg war es gängige Praxis, in `/usr/include` die Kernel-Header direkt aus dem Kernel-Archiv zu benutzen. Aber in den letzten Jahren sind die Kernel-Entwickler zu dem Schluss gekommen, dass dies keine gute Praxis ist. Als Konsequenz entstand das Projekt Linux-Libc-Header. Es wurde entworfen, um eine konsistente Programmierschnittstelle (API) zu den Kernel-Headern zu gewährleisten.

Installieren Sie die Header-Dateien:

```
cp -Rv include/asm-i386 /tools/include/asm
cp -Rv include/linux /tools/include
```

Wenn Sie keine i386-Architektur verwenden, passen Sie den Befehl entsprechend an.

Details zu diesem Paket finden Sie in Abschnitt 6.9.2, „Inhalt von Linux-Libc-Header“

5.6. Glibc-2.3.4

Glibc enthält die C-Bibliothek. Sie stellt Systemaufrufe und grundlegende Funktionen zur Verfügung (z. B. das Zuweisen von Speicher, Durchsuchen von Ordnern, Öffnen und Schließen sowie Schreiben von Dateien, Zeichenkettenverarbeitung, Mustererkennung, Arithmetik etc.)

Geschätzte Kompilierzeit: 11.8 SBU

Ungefähr benötigter Festplattenplatz: 454 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed und Texinfo

5.6.1. Installation von Glibc

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch `-march` und `-mcpu`.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. `CFLAGS` und `CXXFLAGS`) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Grundsätzlich gilt: Wenn Sie von dem hier beschriebenen Weg zum Kompilieren von Glibc abweichen, riskieren Sie die Stabilität Ihres gesamten LFS-Systems.

Glibc enthält zwei Testsuites die mit dem Kernel 2.6.11 oder neuer fehlschlagen. Das Problem sind hier die Tests selbst, nicht die C-Bibliothek oder der Kernel. Wenn Sie die Testsuite durchlaufen lassen möchten, wenden Sie diesen Patch an:

```
patch -Np1 -i ../glibc-2.3.4-fix_test-1.patch
```

Die Dokumentation von Glibc empfiehlt, zum Kompilieren einen gesonderten Ordner zu verwenden:

```
mkdir -v ../glibc-build
cd ../glibc-build
```

Als nächstes bereiten Sie Glibc zum Kompilieren vor:

```
../glibc-2.3.4/configure --prefix=/tools \
  --disable-profile --enable-add-ons \
  --enable-kernel=2.6.0 --with-binutils=/tools/bin \
  --without-gd --with-headers=/tools/include \
  --without-selinux
```

Die Bedeutung der configure-Parameter:

`--disable-profile`

Dadurch werden die Bibliotheken ohne Profiling-Informationen kompiliert. Lassen Sie diesen Parameter weg, wenn Sie mit den temporären Werkzeugen Profiling betreiben möchten.

`--enable-add-ons`

Dadurch verwendet Glibc NPTL als die Threading-Bibliothek.

`--enable-kernel=2.6.0`

Dadurch wird die Glibc mit Unterstützung für Kernel der Serie 2.6.x gebaut.

`--with-binutils=/tools/bin`

Dieser Parameter wird nicht wirklich benötigt, stellt aber sicher, dass in Hinsicht auf die Binutils-Programme beim Kompilieren von Glibc nichts schiefgehen kann.

`--without-gd`

Das verhindert das kompilieren des Programmes **memusagestat**, welches immer mit Bibliotheken auf dem Host-System verlinkt (libgd, libpng, libz usw.).

`--with-headers=/tools/include`

Dadurch wird Glibc mit den gerade in den tools-Ordner installierten Kernel-Headern kompiliert. Auf diese Weise werden alle Funktionen des Kernels erkannt und die Glibc kann entsprechend darauf optimiert werden.

`--without-selinux`

Wenn das Host-System SELinux-Funktionen hat (so z. B. Fedora Core 3), so würden die SELinux-Funktionen auch in Glibc einkompiliert. Die LFS-Werkzeuge unterstützen diese Erweiterungen aber nicht, daher wird eine so erzeugte Glibc nicht korrekt funktionieren.

Während dieser Phase sehen Sie möglicherweise eine Warnung:

```
configure: WARNING:
*** These auxiliary programs are missing or
*** incompatible versions: msgfmt
*** some features will be disabled.
*** Check the INSTALL file for required versions.
```

Das fehlende oder inkompatible Programm **msgfmt** ist normalerweise harmlos, aber manchmal kann es zu Fehlern beim Durchlaufen der Testsuite führen. **msgfmt** ist Teil von Gettext, welches auf dem Host-System installiert sein sollte. Wenn **msgfmt** zwar vorhanden, aber vollkommen inkompatibel ist, dann sollten Sie das Paket auf dem Host-System aktualisieren. Oder Sie fahren ohne das Paket fort und schauen, ob die Testsuite auch ohne problemlos durchläuft.

Kompilieren Sie das Paket:

```
make
```

Der Kompilervorgang ist nun abgeschlossen. Wie bereits erwähnt, wird empfohlen, die Testsuite für das temporäre System in diesem Kapitel nicht durchlaufen zu lassen. Falls Sie die Testsuite dennoch ausführen möchten, verwenden Sie dafür dieses Kommando:

```
make check
```

Eine Information über die kritischen Fehler finden Sie im Abschnitt 6.11, „Glibc-2.3.4“

Die Testsuite von Glibc ist stark von einigen Funktionen Ihres Host-Systems abhängig. Glibc-Fehler in diesem Kapitel sind normalerweise nicht kritisch. Erst in Kapitel 6 wird die endgültige Glibc installiert, dort sollten dann die meisten Tests erfolgreich durchlaufen. Allerdings können selbst in Kapitel 6 noch Fehler auftreten, zum Beispiel beim math-Test.

Wenn ein Fehler auftritt, notieren Sie ihn, dann rufen Sie **make check** erneut auf. Die Testsuite sollte dann dort fortfahren, wo sie unterbrochen wurde. Sie können dieses Stoppen und Starten umgehen, indem Sie **make -k**

check aufrufen. Aber stellen Sie in diesem Fall sicher, dass Sie die Ausgaben protokollieren, damit Sie später die Logdatei nach den aufgetretenen Fehlern durchsuchen können.

Auch wenn es nur eine harmlose Meldung ist, die Installationsroutine von Glibc wird sich über die fehlende Datei `/tools/etc/ld.so.conf` beschweren. Beheben Sie diese störende Warnung mit:

```
mkdir -v /tools/etc
touch /tools/etc/ld.so.conf
```

Installieren Sie das Paket:

```
make install
```

Verschiedene Länder und Kulturen haben auch unterschiedliche Konventionen zum Kommunizieren. Darunter sind einfache Konventionen wie zum Beispiel das Format für Datum und Uhrzeit, aber auch sehr komplexe Konventionen, wie zum Beispiel die dort gesprochene Sprache. Die „Internationalisierung“ von GNU-Programmen funktioniert mit Hilfe der sogenannten Locales. Installieren Sie nun die Glibc-Locales.



Anmerkung

Wenn Sie, wie empfohlen, die Testsuite in diesem Kapitel nicht laufen lassen, brauchen Sie auch die Locales nicht zu installieren. Sie werden sie dann im nächsten Kapitel installieren.

Wenn Sie die Glibc-Locales dennoch installieren möchten, führen Sie dieses Kommando aus:

```
make localedata/install-locales
```

Alternativ können Sie auch nur die von Ihnen benötigten oder gewünschten Locales installieren. Das geht mit dem Kommando `localedef`. Detailliertere Informationen dazu finden Sie in der Datei `INSTALL` aus den Quellen von Glibc. Es gibt jedoch einige Locales die für die Tests von weiteren Paketen vorausgesetzt werden: z. B. die `libstdc++`-Tests von GCC. Die folgenden Anweisungen installieren einen minimalen Satz von notwendigen Locales, um die nachfolgenden Tests erfolgreich durchführen zu können:

```
mkdir -pv /tools/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Details zu diesem Paket finden Sie in Abschnitt 6.11.4, „Inhalt von Glibc“

5.7. Anpassen der Toolchain

Jetzt, nachdem die temporären C-Bibliotheken installiert sind, wollen wir alle im Rest des Kapitels kompilierten Werkzeuge gegen diese Bibliotheken verlinken. Um das zu erreichen, müssen Sie den Linker und die specs-Datei des Compilers anpassen.

Installieren Sie zuerst den angepassten Linker (die Anpassung haben Sie ja bereits am Ende des ersten Binutils-Durchlaufs durchgeführt) indem Sie im Ordner `binutils-build` folgendes Kommando ausführen:

```
make -C ld install
```

Von diesem Punkt an wird alles ausschließlich gegen die Bibliotheken in `/tools/lib` verlinkt.



Anmerkung

Falls Sie die Warnung, die Binutils-Ordner nicht zu löschen, übersehen haben oder Sie vielleicht versehentlich gelöscht haben, dann ignorieren Sie das obige Kommando. In Folge daraus besteht ein gewisses Risiko, dass nachfolgende Programme gegen Bibliotheken auf dem Host-System gelinkt werden. Das ist nicht ideal, aber auch kein allzu großes Problem. Die Situation wird korrigiert, wenn Sie später den zweiten Durchlauf der Binutils installieren.

Nachdem nun der angepasste Linker installiert ist, müssen Sie die Binutils-Ordner löschen.

Als nächstes muss die specs-Datei von GCC ergänzt werden, so dass sie den neuen dynamischen Linker referenziert. Ein einfaches sed-Skript erledigt diese Aufgabe:

```
SPECFILE=`gcc --print-file specs` &&  
sed 's@ /lib/ld-linux.so.2@ /tools/lib/ld-linux.so.2@g' \  
  $SPECFILE > tempspecfile &&  
mv -f tempspecfile $SPECFILE &&  
unset SPECFILE
```

Alternativ können Sie die specs-Datei auch von Hand ändern: ersetzen Sie einfach jedes Vorkommen von „`./lib/ld-linux.so.2`“ durch „`./tools/lib/ld-linux.so.2`“.

Danach sollten Sie die specs-Datei überprüfen und sicherstellen, dass alle gewollten Änderungen durchgeführt wurden.



Wichtig

Wenn Sie auf einer Plattform arbeiten, bei der der Name des dynamischen Linkers nicht `ld-linux.so.2` lautet, müssen Sie natürlich statt „`ld-linux.so.2`“ den korrekten Namen des Linkers für Ihre Plattform einsetzen. Falls nötig, schauen Sie nochmal im Abschnitt Abschnitt 5.2, „Technische Anmerkungen zur Toolchain,“ nach.

Schließlich ist es möglich, dass einige Include-Dateien vom Host-System mit in den privaten Include-Ordner von GCC geraten sind. So etwas kann durch GCCs „`fixincludes`“-Routine geschehen, die beim Kompilieren von GCC ausgeführt wird. Dazu wird später noch näheres erklärt. Zunächst führen Sie das folgende Kommando aus, um dieses mögliche Problem zu beheben:

```
rm -vf /tools/lib/gcc/*/*/include/{pthread.h,bits/sigthread.h}
```



Achtung

An diesem Punkt ist es unbedingt notwendig, die korrekte Funktion der Toolchain (Kompilieren und Linken) zu überprüfen. Darum führen Sie nun einen kleinen „Gesundheitscheck“ durch:

```
echo 'main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
[Requesting program interpreter:
 /tools/lib/ld-linux.so.2]
```

Achten Sie besonders darauf, dass `/tools/lib` als Prefix zu Ihrem dynamischen Linker angegeben ist.

Wenn Sie keine oder eine andere als die obige Ausgabe erhalten haben, ist etwas schiefgelaufen. Sie müssen alle Ihre Schritte noch einmal überprüfen und den Fehler finden und korrigieren. Fahren Sie nicht fort, bevor Sie den Fehler nicht beseitigt haben. Als erstes führen Sie nochmals den Gesundheitscheck durch und benutzen `gcc` anstelle von `cc`. Wenn das funktioniert, fehlt der Link von `/tools/bin/cc`. Gehen Sie zurück zu Abschnitt 5.4, „GCC-3.4.3 - Durchlauf 1“ und reparieren Sie den symbolischen Link. Als zweites stellen Sie bitte sicher, dass Ihre Umgebungsvariable `PATH` richtig gesetzt ist. Sie können die Variable mit dem Kommando `echo $PATH` anzeigen; prüfen Sie, dass `/tools/bin` am Anfang der Liste steht. Wenn die `PATH` Variable falsch gesetzt ist, sind Sie möglicherweise nicht als *lfs* eingeloggt oder in Abschnitt 4.4, „Vorbereiten der Arbeitsumgebung“ ist etwas schiefgelaufen. Vielleicht hat auch beim Anpassen der `specs`-Datei etwas nicht richtig funktioniert. In diesem Fall wiederholen Sie die Anpassung.

Wenn Sie mit dem Ergebnis zufrieden sind, räumen Sie auf:

```
rm -v dummy.c a.out
```

Das Kompilieren von TCL im nächsten Abschnitt ist gleichzeitig auch ein zusätzlicher Test, ob die Toolchain korrekt erstellt wurde. Falls TCL nicht kompilierbar ist, weist das auf einen Fehler mit Binutils, GCC oder Glibc hin, nicht aber auf einen Fehler in TCL.

5.8. Tcl-8.4.9

Das Tcl Paket enthält die Tool Command Language.

Geschätzte Kompilierzeit: 0.9 SBU

Ungefähr benötigter Festplattenplatz: 23.3 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

5.8.1. Installation von Tcl

Dieses und die nächsten beiden Pakete werden nur installiert, damit Sie die Testsuites von GCC und Binutils laufen lassen können. Drei Pakete nur zu Testzwecken zu installieren könnte etwas übertrieben erscheinen, aber es ist wirklich sehr wichtig zu wissen, dass unsere grundlegendsten Programme und Werkzeuge richtig funktionieren. Selbst wenn wir die Testsuites in diesem Kapitel nicht ausführen (wie empfohlen), werden diese Pakete doch zumindest für die Tests im nächsten Kapitel 6 benötigt.

Bereiten Sie Tcl zum Kompilieren vor:

```
cd unix
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite ausführen möchten, führen Sie **TZ=UTC make test** aus. Es ist jedoch bekannt, dass die Testsuite von Tcl unter bestimmten Bedingungen fehlschlägt. Daher sind Fehler in der Testsuite nicht überraschend; wir betrachten diese Fehler nicht als kritisch. Der Parameter *TZ=UTC* setzt die Zeitzone für die Dauer des Durchlaufs der Testsuite auf Coordinated Universal Time (UTC), auch als Greenwich Mean Time (GMT) bekannt. Dadurch werden zeitbezogene Tests korrekt ausgewertet. Mehr Informationen zu der Umgebungsvariable TZ finden Sie später in Kapitel 7.

Installieren Sie das Paket:

```
make install
```



Warnung

Sie sollten den Quellordner `tcl8.4.9` noch nicht entfernen, weil das nächste Paket die internen Header-Dateien benötigt.

Setzen Sie die folgende Umgebungsvariable mit dem vollen Pfad des aktuellen Ordners. Das nächste Paket (expect) wird sie zum Auffinden der Header-Dateien von Tcl verwenden.

```
cd ..
export TCLPATH=`pwd`
```

Erstellen Sie einen nötigen symbolischen Link:

```
ln -sv tclsh8.4 /tools/bin/tclsh
```

5.8.2. Inhalt von Tcl

Installierte Programme: tclsh (Link auf tclsh8.4) uand tclsh8.4

Installierte Bibliothek: libtcl8.4.so

Kurze Beschreibungen

tclsh8.4 Die Tcl Kommando-Shell.

tclsh Ein Link auf **tclsh8.4**.

libtcl8.4.so Die Tcl-Bibliothek

5.9. Expect-5.43.0

Das Paket Expect führt vorprogrammierte Dialoge mit anderen interaktiven Programmen aus.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 4.0 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed und Tcl

5.9.1. Installation von Expect

Spielen Sie erst einen Patch ein; er behebt einen Fehler, der ansonsten Fehlalarme beim Durchlaufen von GCCs Testsuite verursachen könnte:

```
patch -Np1 -i ../expect-5.43.0-spawn-1.patch
```

Bereiten Sie Expect nun zum Kompilieren vor:

```
./configure --prefix=/tools --with-tcl=/tools/lib \  
--with-tclinclude=$TCLPATH --with-x=no
```

Die Bedeutung der configure-Parameter:

--with-tcl=/tools/lib

So stellen Sie sicher, dass das configure-Skript die Tcl-Installation in Ihrem temporären Ordner findet. Es sollte keine möglicherweise auf dem Host-System installierte Version gefunden werden.

--with-tclinclude=\$TCLPATH

Durch diesen Parameter wird Expect mitgeteilt, wo der Quellordner und die Header von Tcl zu finden sind. Dadurch wird ein Fehlschlagen von **configure** vermieden, falls es den Tcl-Quellordner nicht automatisch auffinden kann.

--with-x=no

Dies teilt dem configure-Skript mit, dass es nicht nach Tk (der grafischen Oberfläche zu Tcl) oder den X Window-Bibliotheken suchen soll; beide könnten eventuell auf dem Host-System existieren, fehlen aber in der temporären Arbeitsumgebung.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie **make test** aus. Es ist jedoch bekannt, dass die Testsuite in diesem Kapitel Probleme macht, die noch nicht ganz nachvollzogen wurden. Es ist daher nicht überraschend, wenn die Testsuite Fehler meldet, diese werden jedoch nicht als kritisch betrachtet.

Installieren Sie das Paket:

```
make SCRIPTS="" install
```

Die Bedeutung des make-Parameters:

```
SCRIPTS=""
```

Dies verhindert die Installation der mitgelieferten Expect-Skripte, sie werden hier nicht gebraucht.

Löschen Sie nun die Umgebungsvariable TCLPATH:

```
unset TCLPATH
```

Sie können nun die Quellordner von Tcl und Expect entfernen.

5.9.2. Inhalt von Expect

Installiertes Programm: expect

Installierte Bibliothek: libexpect-5.43.a

Kurze Beschreibungen

expect	Expect „Spricht“ mit anderen interaktiven Programmen. Es verwendet dafür ein anpassbares Skript.
<code>libexpect-5.43.a</code>	Enthält Funktionen, mit denen man Expect als TCL-Erweiterung oder direkt aus C/C++ (ohne TCL) nutzen kann

5.10. DejaGNU-1.4.4

Das Paket DejaGNU enthält ein Grundgerüst zum Testen anderer Programme.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 6.1 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

5.10.1. Installation von DejaGNU

Bereiten Sie DejaGNU zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren und installieren Sie das Paket:

```
make install
```

5.10.2. Inhalt von DejaGNU

Installiertes Programm: runtest

Kurze Beschreibungen

runtest Das Wrapper-Skript, das die korrekte **expect**-Shell findet und DejaGNU ausführt.

5.11. GCC-3.4.3 - Durchlauf 2

Geschätzte Kompilierzeit: 11.0 SBU

Ungefähr benötigter Festplattenplatz: 292 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed und Texinfo

5.11.1. Neuinstallation von GCC

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch `-march` und `-mcpu`.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. `CFLAGS` und `CXXFLAGS`) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Die Hilfsmittel zum Testen von GCC und Binutils sind nun installiert (Tcl, Expect und DejaGNU). Sie können GCC und Binutils nun erneut installieren, gegen die neue Glibc verlinken und testen. Eines muss noch beachtet werden: Die Testsuites sind stark von funktionierenden Pseudo-Terminals (PTYs) abhängig. Diese werden vom Host-System bereitgestellt. Heutzutage werden PTYs meist über das Dateisystem `devpts` implementiert. Ob Ihr Host-System korrekt eingerichtet ist, können Sie mit einem einfachen Test feststellen:

```
expect -c "spawn ls"
```

Das Ergebnis könnte so aussehen:

```
The system has no more ptys.
Ask your system administrator to create more.
```

Wenn Sie die obige Meldung sehen, ist Ihr Host-System nicht korrekt für PTYs eingerichtet. Solange Sie dieses Problem nicht behoben haben, brauchen Sie die Testsuites von GCC und Binutils gar nicht erst durchlaufen lassen. Wenn Sie mehr Informationen zum Einrichten von PTYs brauchen, schauen Sie am besten in die LFS-FAQ unter <http://www.linuxfromscratch.org/lfs/faq.html#no-ptys>.

Zuerst korrigieren Sie ein Problem und nehmen eine wichtige Anpassung vor:

```
patch -Np1 -i ../gcc-3.4.3-no_fixincludes-1.patch
patch -Np1 -i ../gcc-3.4.3-specs-2.patch
```

Der erste Patch schaltet das **fixincludes**-Skript von GCC ab. Dies wurde vorher bereits kurz erwähnt; nun folgt eine nähere Erklärung dazu. Unter normalen Umständen durchsucht GCC's **fixincludes**-Skript Ihr System nach zu reparierenden Header-Dateien. Dabei kann es vorkommen, dass das Skript der Meinung ist, einige Header-Dateien auf Ihrem Host-System müssten repariert werden. GCC repariert diese dann und kopiert sie in den privaten GCC Include-Ordner. Später dann, in Kapitel 6, nachdem Sie die neuere Glibc installiert haben, würde dieser private Include-Ordner vor den Systemweiten Include-Ordern durchsucht werden. GCC würde dann die reparierten Include-Dateien des Host-Systems finden, und diese passen dann höchstwahrscheinlich nicht zu der Glibc-Version, die Sie für das LFS-System verwendet haben.

Der zweite Patch ändert den GCC-Standardpfad zum dynamischen Linker (üblicherweise `ld-linux.so.2`). Außerdem entfernt er `/usr/include` aus dem Include-Suchpfad von GCC. Das Patchen an dieser Stelle statt des nachträglichen Anpassens der specs-Datei stellt sicher, dass beim Kompilieren von GCC unser neuer

dynamischer Linker verwendet wird. Das bedeutet, dass alle endgültigen (und auch temporären) Binärdateien beim Kompilervorgang gegen die neue Glibc gelinkt werden.

**Wichtig**

Diese Patches sind zwingende Voraussetzung für einen erfolgreichen Gesamtdurchlauf. Vergessen Sie nicht, sie zu installieren!

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Denken Sie daran, vor dem Kompilieren von GCC alle Umgebungsvariablen zurückzusetzen, die die Standard-Optimierungen überschreiben würden.

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-3.4.3/configure --prefix=/tools \
  --libexecdir=/tools/lib --with-local-prefix=/tools \
  --enable-clocale=gnu --enable-shared \
  --enable-threads=posix --enable-__cxa_atexit \
  --enable-languages=c,c++ --disable-libstdcxx-pch
```

Die Bedeutung der neuen Parameter zu configure:

`--enable-clocale=gnu`

Dieser Parameter stellt sicher, dass unter allen Umständen das korrekte locale-Modell für die C++ Bibliotheken ausgewählt wird. Falls das configure-Skript *de_DE* Locales findet, wird es das korrekte Modell gnu wählen. Falls aber *de_DE* nicht installiert ist, besteht das Risiko, dass aufgrund des fälschlicherweise ausgewählten Modells generic ABI-inkompatible C++-Bibliotheken erstellt werden.

`--enable-threads=posix`

Das schaltet die Behandlung von C++-Exceptions für Code mit Threads ein.

`--enable-__cxa_atexit`

Dieser Parameter erlaubt die Benutzung von `__cxa_atexit` anstelle von `atexit`, um C++-Destruktoren für lokale Statics und globale Objekte zu registrieren. Außerdem ist die Option für eine vollständig standardkonforme Behandlung von Destruktoren erforderlich. Das beeinflusst auch die C++ ABI; das Ergebnis sind gemeinsame C++-Bibliotheken und C++-Programme die interoperabel mit anderen Linux-Distributionen sind.

`--enable-languages=c,c++`

Dieser Parameter stellt sicher, dass sowohl der C- als auch der C++-Compiler erzeugt werden.

`--disable-libstdcxx-pch`

Verhindert das Erzeugen der vorkompilierten Header-Dateien (PCH, pre-compiled header) für `libstdc++`. Diese Funktion verbraucht viel Platz und wir benötigen sie nicht.

Kompilieren Sie das Paket:

```
make
```

Diesmal müssen Sie nicht das *bootstrap*-Target verwenden, weil Sie bereits einen Compiler benutzen, der

aus exakt den gleichen Quellen gebaut wurde.

Der Kompiliervorgang ist nun abgeschlossen. Wie bereits erwähnt, empfehlen wir, die Testsuite für das temporäre System in diesem Kapitel nicht durchlaufen zu lassen. Falls Sie die Testsuite dennoch laufen lassen möchten, führen Sie dieses Kommando aus:

```
make -k check
```

Der Parameter `-k` lässt die Testsuite bis zum Ende durchlaufen, selbst wenn Fehler auftreten sollten. Die Testsuite von GCC ist sehr umfangreich und es ist beinahe sicher, dass Fehler auftreten. Um eine Zusammenfassung der Ergebnisse zu erhalten, benutzen Sie dieses Kommando:

```
../gcc-3.4.3/contrib/test_summary
```

Wenn Sie nur die Zusammenfassungen sehen möchten, pipen Sie die Ausgabe durch **grep -A7 Summ.**

Sie können die Ergebnisse mit denen unter <http://www.linuxfromscratch.org/lfs/build-logs/6.1.1/> vergleichen.

Ein paar unerwartete Fehler lassen sich oftmals nicht vermeiden. Die Entwickler von GCC kennen diese üblicherweise bereits, hatten aber noch keine Zeit, diese Fehler zu beheben. Kurz gesagt, solange Ihre Testergebnisse nicht grob von denen unter der obigen URL abweichen, können Sie beruhigt fortfahren.

Installieren Sie das Paket:

```
make install
```



Anmerkung

An diesem Punkt wird dringend empfohlen, die Gesamtprüfung, die Sie früher in diesem Kapitel gemacht haben, noch einmal zu wiederholen. Schlagen Sie im Abschnitt 5.7, „Anpassen der Toolchain,“ nach und wiederholen Sie die Prüfung. Wenn die Ergebnisse nicht in Ordnung sind, haben Sie höchstwahrscheinlich vergessen, den oben erwähnten GCC Specs-Patch einzuspielen.

Details zu diesem Paket finden Sie in Abschnitt 6.14.2, „Inhalt von GCC“

5.12. Binutils-2.15.94.0.2.2 - Durchlauf 2

Binutils ist eine Sammlung von Software-Entwicklungswerkzeugen. Dazu gehören zum Beispiel Linker, Assembler und weitere Programme für die Arbeit mit Objektdateien.

Geschätzte Kompilierzeit: 1.5 SBU

Ungefähr benötigter Festplattenplatz: 114 MB

Die Installation ist abhängig von: Bash, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed und Texinfo

5.12.1. Neuinstallation von Binutils

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch *-march* und *-mcpu*.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. *CFLAGS* und *CXXFLAGS*) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.15.94.0.2.2/configure --prefix=/tools \
  --disable-nls --enable-shared --with-lib-path=/tools/lib
```

Die Bedeutung der neuen Parameter zu *configure*:

```
--with-lib-path=/tools/lib
```

Dies teilt dem *configure*-Skript mit, den Standard Bibliotheksuchpfad des Linkers als */tools/lib* vorzugeben. Wir möchten im Standard Bibliotheksuchpfad keine Ordner unseres Host-Systems haben, daher geben Sie den gewünschten Pfad vor.

Kompilieren Sie das Paket:

```
make
```

Der Kompilervorgang ist nun abgeschlossen. Wie bereits erwähnt, wird empfohlen, die Testsuite für das temporäre System in diesem Kapitel nicht durchlaufen zu lassen. Falls Sie die Testsuite dennoch laufen lassen möchten, führen Sie dieses Kommando aus:

```
make check
```

Installieren Sie das Paket:

```
make install
```

Nun bereiten Sie Binutils auf das erneute Anpassen der Toolchain im nächsten Kapitel vor:

```
make -C ld clean  
make -C ld LIB_PATH=/usr/lib:/lib
```



Warnung

Entfernen Sie die Binutils Quell- und Kompilierordner jetzt noch nicht. Sie brauchen sie im jetzigen Zustand noch im nächsten Kapitel.

Details zu diesem Paket finden Sie in Abschnitt 6.13.2, „Inhalt von Binutils“

5.13. Gawk-3.1.4

Gawk ist eine Implementierung von awk und wird zur Textmanipulation verwendet.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 16.4 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

5.13.1. Installation von Gawk

Bereiten Sie Gawk zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.20.2, „Inhalt von Gawk“

5.14. Coreutils-5.2.1

Das Paket Coreutils enthält viele Shell-Werkzeuge zum Einstellen der grundlegenden Systemeigenschaften.

Geschätzte Kompilierzeit: 0.9 SBU

Ungefähr benötigter Festplattenplatz: 53.3 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl und Sed

5.14.1. Installation von Coreutils

Bereiten Sie Coreutils zum Kompilieren vor:

```
DEFAULT_POSIX2_VERSION=199209 ./configure --prefix=/tools
```

Dieses Paket hat ein Problem, wenn es mit neueren Glibc-Versionen als 2.3.2 kompiliert wird. Einige der Coreutils-Werkzeuge (wie z. B. **head**, **tail**, und **sort**) lehnen ihre traditionelle Syntax ab; eine Syntax, die allerdings bereits seit ca. 30 Jahren verwendet wird. Die alte Syntax ist so eingebürgert, dass hier die Kompatibilität bewahrt werden sollte, bis die neue Syntax überall übernommen wurde. Rückwärtskompatibilität kann durch das Setzen der Umgebungsvariable `DEFAULT_POSIX2_VERSION` auf „199209“ erreicht werden. Wenn Sie keine Rückwärtskompatibilität wünschen, lassen Sie die Variable einfach weg. Dann müssen Sie allerdings mit den Konsequenzen leben: Viele Pakete müssen gepatcht werden, damit sie mit der neuen Syntax klar kommen. Wir empfehlen, die oben angegebenen Anweisungen so zu übernehmen.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **make RUN_EXPENSIVE_TESTS=yes check**. Der Parameter `RUN_EXPENSIVE_TESTS=yes` teilt der Testsuite mit, noch zusätzliche Tests zu durchlaufen, die auf einigen Plattformen sehr zeitintensiv sein können. Normalerweise ist das unter Linux aber kein Problem.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.15.2, „Inhalt von Coreutils“

5.15. Bzip2-1.0.3

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. **Bzip2** erreicht vor allem bei Textdateien eine wesentlich bessere Kompressionsrate als das traditionelle **gzip**.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 3.5 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc und Make

5.15.1. Installation von Bzip2

Das Paket Bzip2 enthält kein **configure**-Skript. Kompilieren Sie es einfach:

```
make
```

Installieren Sie das Paket:

```
make PREFIX=/tools install
```

Details zu diesem Paket finden Sie in Abschnitt 6.40.2, „Inhalt von Bzip2“

5.16. Gzip-1.3.5

Das Paket Gzip enthält Programme zum Komprimieren und Dekomprimieren von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.2 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

5.16.1. Installation von Gzip

Bereiten Sie Gzip zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.46.2, „Inhalt von Gzip“

5.17. Diffutils-2.8.1

Die Programme dieses Pakets können Unterschiede zwischen Dateien oder Ordnern anzeigen.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 5.6 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

5.17.1. Installation von Diffutils

Bereiten Sie Diffutils zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.41.2, „Inhalt von Diffutils“

5.18. Findutils-4.2.23

Das Paket Findutils enthält Programme zum Auffinden von Dateien durch rekursive Suche in einer Ordnerstruktur oder über den Zugriff auf eine Datenbank. Die Suche über eine Datenbank ist normalerweise schneller, aber es besteht natürlich die Gefahr, dass die Datenbank zum Zeitpunkt der Suche veraltet ist.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 8.9 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

5.18.1. Installation von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.19.2, „Inhalt von Findutils“

5.19. Make-3.80

Das Paket Make enthält Werkzeuge zum Kompilieren von Software.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 7.1 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep und Sed

5.19.1. Installation von Make

Bereiten Sie Make zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.49.2, „Inhalt von Make“

5.20. Grep-2.5.1a

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 4.5 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed und Texinfo

5.20.1. Installation von Grep

Bereiten Sie Grep zum Kompilieren vor:

```
./configure --prefix=/tools \
--disable-perl-regexp
```

Die Bedeutung der configure-Parameter:

--disable-perl-regexp

Dies stellt sicher, dass **grep** nicht gegen die PCRE-Bibliothek verlinkt wird. Diese Bibliothek könnte auf dem Host-System installiert sein, ist aber später in der **chroot**-Umgebung nicht mehr verfügbar.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.44.2, „Inhalt von Grep“

5.21. Sed-4.1.4

Das Paket Sed enthält einen Stream-Editor.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 8.4 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Texinfo

5.21.1. Installation von Sed

Bereiten Sie Sed zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.28.2, „Inhalt von Sed“

5.22. Gettext-0.14.3

Gettext wird zur Übersetzung und Lokalisierung verwendet. Programme können mit Unterstützung für NLS (Native Language Support, Unterstützung für die lokale Sprache) kompiliert werden. Dadurch können Texte und Meldungen in der Sprache des Anwenders ausgegeben werden.

Geschätzte Kompilierzeit: 0.5 SBU

Ungefähr benötigter Festplattenplatz: 63.0 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

5.22.1. Installation von Gettext

Bereiten Sie Gettext zum Kompilieren vor:

```
./configure --prefix=/tools --disable-libasprintf \
--without-csharp
```

Die Bedeutung der configure-Parameter:

--disable-libasprintf

Dieser Parameter sorgt dafür, dass Gettext die Bibliothek `asprintf` nicht erzeugt. Weil nichts in diesem Kapitel diese Bibliothek benötigt, und sie nur unnötig Zeit und Platz verschwendet, überspringen wir sie hier. Diese Bibliothek wird zusammen mit Gettext später noch einmal installiert.

--without-csharp

Dies stellt sicher, dass Ncurses ohne Unterstützung für C#-Compiler erzeugt wird. Auf dem Host-System könnte Unterstützung für C# installiert sein. Sie wäre dann aber später in der **chroot**-Umgebung nicht mehr verfügbar.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **make check**. Die Testsuite von Gettext braucht sehr viel Zeit (ca. 7 SBU). Es ist bekannt, dass die Testsuite von Gettext in diesem Kapitel unter verschiedenen Bedingungen fehlschlägt — zum Beispiel, wenn Sie einen Java-Compiler auf dem Host-System findet. Ein experimenteller Patch zum Deaktivieren von Java ist aus dem LFS-Patches-Projekt unter <http://www.linuxfromscratch.org/patches/> verfügbar.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.30.2, „Inhalt von Gettext“

5.23. Ncurses-5.4

Das Paket Ncurses enthält Bibliotheken für den Terminal-unabhängigen Zugriff auf Textbildschirme.

Geschätzte Kompilierzeit: 0.7 SBU

Ungefähr benötigter Festplattenplatz: 27.5 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

5.23.1. Installation von Ncurses

Bereiten Sie Ncurses zum Kompilieren vor:

```
./configure --prefix=/tools --with-shared \
--without-debug --without-ada --enable-overwrite
```

Die Bedeutung der configure-Parameter:

--without-ada

Dies stellt sicher, dass Ncurses ohne Unterstützung für Ada-Compiler erzeugt wird. Auf dem Host-System könnte Unterstützung für Ada installiert sein. Sie wäre dann aber später in der **chroot**-Umgebung nicht mehr verfügbar.

--enable-overwrite

Dadurch werden die Header-Dateien von Ncurses in `/tools/include` anstelle von `/tools/include/ncurses` installiert. Das stellt sicher, dass andere Pakete die Header-Dateien problemlos finden können.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.21.2, „Inhalt von Ncurses“

5.24. Patch-2.5.4

Das Paket Patch enthält ein Programm zum Erzeugen oder Modifizieren von Dateien indem eine sogenannte „Patch“-Datei angewendet wird. Einen „Patch“ erzeugt man üblicherweise mit **diff** und er beschreibt in maschinenlesbarer Form die Unterschiede zwischen zwei Versionen einer Datei.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 1.5 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

5.24.1. Installation von Patch

Bereiten Sie Patch zum Kompilieren vor:

```
CPPFLAGS=-D_GNU_SOURCE ./configure --prefix=/tools
```

Die Präprozessor-Option `-D_GNU_SOURCE` wird nur auf der PowerPC-Plattform benötigt. Auf anderen Architekturen können Sie sie weglassen.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.51.2, „Inhalt von Patch“

5.25. Tar-1.15.1

Das Paket Tar enthält ein Archivprogramm.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 12.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

5.25.1. Installation von Tar

Bereiten Sie Tar zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.57.2, „Inhalt von Tar“

5.26. Texinfo-4.8

Das Paket Texinfo enthält Programme zum Lesen, Schreiben und Konvertieren von Info-Seiten (Systemdokumentation).

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 14.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses und Sed

5.26.1. Installation von Texinfo

Bereiten Sie Texinfo zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.34.2, „Inhalt von Texinfo“

5.27. Bash-3.0

Das Paket Bash enthält die Bourne-Again-Shell.

Geschätzte Kompilierzeit: 1.2 SBU

Ungefähr benötigter Festplattenplatz: 20.7 MB

Die Installation ist abhängig von: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses und Sed.

5.27.1. Installation von Bash

Bash hat einen Fehler, der mit neueren Versionen von Glibc einen Hänger verursacht. Dieser Patch behebt das Problem:

```
patch -Np1 -i ../bash-3.0-avoid_WCONTINUED-1.patch
```

Bereiten Sie Bash zum Kompilieren vor:

```
./configure --prefix=/tools --without-bash-malloc
```

Die Bedeutung der configure-Parameter:

--without-bash-malloc

Dieser Parameter schaltet Bash's memory allocation (malloc) Funktion ab; sie ist dafür bekannt, Speicherzugriffsfehler zu verursachen. Durch das Abschalten der Funktion, wird Bash die stabilere malloc-Funktion von Glibc benutzen.

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **make tests**.

Installieren Sie das Paket:

```
make install
```

Und erstellen Sie einen Link für die Programme, die **sh** als Shell benutzen:

```
ln -vs bash /tools/bin/sh
```

Details zu diesem Paket finden Sie in Abschnitt 6.37.2, „Inhalt von Bash“

5.28. M4-1.4.3

M4 enthält einen Makroprozessor.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.8 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl und Sed

5.28.1. Installation von M4

Bereiten Sie M4 zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.24.2, „Inhalt von M4“

5.29. Bison-2.0

Mit Bison lassen sich Programme generieren, die die Struktur einer Textdatei analysieren.

Geschätzte Kompilierzeit: 0.6 SBU

Ungefähr benötigter Festplattenplatz: 10.0 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make und Sed

5.29.1. Installation von Bison

Bereiten Sie Bison zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.25.2, „Inhalt von Bison“

5.30. Flex-2.5.31

Mit Flex kann man Programme zum Erkennen von Textmustern erzeugen.

Geschätzte Kompilierzeit: 0.6 SBU

Ungefähr benötigter Festplattenplatz: 22.5 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make und Sed

5.30.1. Installation von Flex

Flex enthält einige bekannte Fehler. Beheben Sie diese mit dem folgenden Patch:

```
patch -Np1 -i ../flex-2.5.31-debian_fixes-3.patch
```

Die GNU autotools werden feststellen, dass die Quellen zu Flex durch den vorigen Patch manipuliert wurden und versuchen, die Hilfeseiten entsprechend zu aktualisieren. Das schlägt jedoch auf vielen Systemen fehl, und die voreingestellte Hilfeseite ist vollkommen in Ordnung. Daher stellen Sie sicher, dass die Manpage nicht neu erzeugt wird:

```
touch doc/flex.1
```

Bereiten Sie Flex nun zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.29.2, „Inhalt von Flex“

5.31. Util-linux-2.12q

Das Paket Util-linux enthält verschiedene Werkzeuge. Darunter befinden sich Programme zum Umgang mit Dateisystemen, Konsolen, Partitionen und (System-)Meldungen.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 8.9 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed und Zlib

5.31.1. Installation von Util-linux

Util-linux verwendet die gerade frisch installierten Header und Bibliotheken im Ordner `/tools` nicht automatisch. Korrigieren Sie dieses Problem indem Sie das `configure`-Skript anpassen:

```
sed -i 's@/usr/include@/tools/include@g' configure
```

Bereiten Sie Util-linux zum Kompilieren vor:

```
./configure
```

Kompilieren Sie einige unterstützende Routinen:

```
make -C lib
```

Aus diesem Paket müssen nur wenige Programme kompiliert werden:

```
make -C mount mount umount
make -C text-utils more
```

Dieses Paket enthält keine Testsuite.

Nun kopieren Sie diese Programme in unseren temporären Ordner `tools`:

```
cp mount/{,u}mount text-utils/more /tools/bin
```

Details zu diesem Paket finden Sie in Abschnitt 6.59.3, „Inhalt von Util-linux“

5.32. Perl-5.8.7

Das Paket Perl enthält die Skriptsprache Perl (Practical Extraction and Report Language).

Geschätzte Kompilierzeit: 0.8 SBU

Ungefähr benötigter Festplattenplatz: 81.6 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

5.32.1. Installation von Perl

Zuerst müssen Sie mit dem folgenden Patch ein paar festeingestellte Pfade zur C-Bibliothek anpassen:

```
patch -Np1 -i ../perl-5.8.7-libc-1.patch
```

Bereiten Sie Perl nun zum Kompilieren vor (passen Sie auf, dass Sie 'IO Fcntl POSIX' richtig schreiben—es sind alles Buchstaben):

```
./configure.gnu --prefix=/tools -Dstatic_ext='IO Fcntl POSIX'
```

Die Bedeutung der configure-Parameter:

```
-Dstatic_ext='IO Fcntl POSIX'
```

Damit wird Perl angewiesen, die notwendigsten statischen Erweiterungen zu installieren, die im nächsten Kapitel für die Coreutils benötigt werden.

Aus diesem Paket müssen nur wenige Programme kompiliert werden:

```
make perl utilities
```

Auch wenn Perl eine Testsuite enthält, sollte sie zum jetzigen Zeitpunkt noch nicht ausgeführt werden. Es wurden nur Teile von Perl installiert und das Ausführen von **make test** würde bewirken, dass nun der Rest von Perl kompiliert werden würden. Das ist zu diesem Zeitpunkt völlig unnötig, die Testsuite kann im nächsten Kapitel ausgeführt werden.

Installieren Sie diese Werkzeuge und ihre Bibliotheken an die richtige Stelle:

```
cp -v perl pod/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.8.7
cp -Rv lib/* /tools/lib/perl5/5.8.7
```

Details zu diesem Paket finden Sie in Abschnitt 6.33.2, „Inhalt von Perl“

5.33. Stripping

Die Schritte in diesem Abschnitt sind optional. Wenn Ihre LFS-Partition sehr klein ist, werden Sie froh sein, ein paar unnötige Dinge loswerden zu können. Die bisher erstellten ausführbaren Dateien und Bibliotheken enthalten ungefähr 130 MB nicht benötigter Debugging-Symbole. So entfernen Sie diese Symbole:

```
strip --strip-debug /tools/lib/*
strip --strip-unnneeded /tools/{,s}bin/*
```

Das erste der obigen Kommandos überspringt rund 20 Dateien mit der Meldung, dass der Dateityp nicht erkannt wurde. Die meisten dieser Dateien sind Skripte und keine Binärdateien.

Passen Sie auf, dass Sie *--strip-unnneeded* nicht auf Bibliotheken anwenden — sie würden zerstört werden und dann müssten Sie die Toolchain neu kompilieren.

Um weitere 30 MB Platz zu sparen, können Sie die Dokumentation entfernen:

```
rm -rf /tools/{info,man}
```

Sie werden nun zum Installieren der Glibc mindestens 850 MB freien Platz auf Ihrem LFS-Dateisystem benötigen. Wenn Sie Glibc kompilieren und installieren können, werden Sie mit den restlichen Paketen keine Probleme haben.

Teil III. Installation des LFS-Systems

Kapitel 6. Installieren der grundlegenden System-Software

6.1. Einführung

In diesem Kapitel begeben Sie sich an den eigentlichen Ort des Geschehens und beginnen mit dem Bau des endgültigen LFS-Systems. Im einzelnen chroot'en Sie in Ihr temporäres Mini-Linux, erzeugen einige Hilfsmittel und beginnen dann, alle Pakete der Reihe nach zu installieren.

Die Installation der Software ist sehr gradlinig. Auch wenn die Installationsanweisungen an einigen Stellen sicherlich kürzer hätten ausfallen können, haben wir uns für die ausführliche Variante entschieden. Wenn Sie lernen möchten wie Linux intern funktioniert, dann sollten Sie wissen, wofür die jeweiligen Pakete benutzt werden und warum ein Benutzer oder das System auf sie angewiesen sind. Deshalb finden Sie zu jedem Paket eine Zusammenfassung seines Inhalts und eine kurze Beschreibung zu den installierten Programmen und Bibliotheken.

Falls Sie in diesem Kapitel Compiler-Optimierungen verwenden möchten, lesen Sie bitte die Anleitung unter <http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt>. Compiler-Optimierungen können ein Programm etwas schneller ablaufen lassen, aber sie können auch zu Schwierigkeiten beim Kompilieren oder Ausführen von Programmen führen. Wenn sich ein Paket nicht kompilieren lässt, versuchen Sie es erstmal ohne Optimierungen und schauen Sie, ob das Problem dann behoben ist. Selbst wenn das Paket mit Compiler-Optimierungen kompilierbar ist, besteht die Gefahr, dass es fehlerhaft kompiliert wurde (z. B. wegen des komplexen Zusammenspiels zwischen Code und den Compilerwerkzeugen). Kurz gesagt, der potentielle Geschwindigkeitsvorteil wird durch das hohe Risiko aufgehoben. Wenn Sie das erste mal ein LFS erstellen, sollten Sie keine Compiler-Optimierungen benutzen. Ihr System wird trotzdem sehr schnell sein und gleichzeitig auch noch stabil.

Die Installationsreihenfolge in diesem Kapitel muss auf jeden Fall eingehalten werden, sonst könnten einige Programme eventuell feste Referenzen auf `/tools` erhalten. *Kompilieren Sie aus diesem Grund auch nicht mehrere Pakete gleichzeitig.* Gleichzeitiges Kompilieren kann Ihnen eine Zeitersparnis bringen, besonders auf Mehrprozessormaschinen, aber es kann zu Programmen führen, die Referenzen auf `/tools` enthalten und nicht mehr funktionieren sobald dieser Ordner entfernt wird.

Auf jeder Informationsseite finden Sie als erstes ein paar allgemeine Informationen zum jeweiligen Paket: Eine kurze Beschreibung des Inhalts, eine Abschätzung der benötigten Kompilierzeit, des benötigten Festplattenspeichers beim Kompilieren, und welche anderen Pakete zum erfolgreichen Kompilieren benötigt werden. Nach den Installationsanweisungen folgt eine Liste der Programme und Bibliotheken (inklusive einer kurzen Beschreibung), die das Paket installiert.

Wenn Sie im Auge behalten möchten, welches Paket welche Dateien installiert, sollten Sie einen Paketmanager verwenden. Eine allgemeine Übersicht zu Paketmanagern finden Sie unter <http://www.linuxfromscratch.org/blfs/view/svn/introduction/important.html>. Eine Paketmanagement-Methode speziell für LFS finden Sie unter http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt.



Anmerkung

Für den Rest des Buches sollten Sie als Benutzer `root` arbeiten, und nicht als `lfs`. An dieser Stelle sollten Sie außerdem nochmals überprüfen, ob `$LFS` gesetzt ist.

6.2. Einhängen der virtuellen Kernel-Dateisysteme

Verschiedene vom Kernel exportierte Dateisysteme werden für die Kommunikation zwischen dem Kernel selbst und dem sog. Userspace verwendet. Dies sind virtuelle Dateisysteme in Hinsicht darauf, dass sie keinen Speicherplatz auf der Festplatte verbrauchen. Der Inhalt der Dateisysteme liegt vollständig im Arbeitsspeicher.

Erstellen Sie die Ordner, in die dann die virtuellen Dateisysteme eingehängt werden:

```
mkdir -pv $LFS/{proc,sys}
```

Und hängen Sie sie ein:

```
mount -vt proc proc $LFS/proc  
mount -vt sysfs sysfs $LFS/sys
```

Denken Sie daran: wenn Sie aus irgendeinem Grund die Arbeit an LFS beenden und später wieder einsteigen, müssen Sie diese Dateisysteme erneut einhängen, bevor Sie in die chroot-Umgebung wechseln.

Schon bald werden aus der chroot-Umgebung heraus weitere Dateisysteme eingebunden. Um das Host-System auf dem neuesten Stand zu halten, sollte für jedes dieser Dateisysteme ein „fake mount“ ausgeführt werden:

```
mount -vft tmpfs tmpfs $LFS/dev  
mount -vft tmpfs tmpfs $LFS/dev/shm  
mount -vft devpts -o gid=4,mode=620 devpts $LFS/dev/pts
```

6.3. Betreten der chroot-Umgebung

Es ist nun an der Zeit, die chroot-Umgebung zu betreten und mit der Installation der benötigten Pakete zu beginnen. Immer noch als *root* führen Sie das folgende Kommando aus. Damit betreten Sie die neue kleine Welt, die zur Zeit nur mit temporären Werkzeugen ausgestattet ist:

```
chroot "$LFS" /tools/bin/env -i \
  HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
  PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
  /tools/bin/bash --login +h
```

Die an `env` übergebene Option `-i` löscht alle Variablen in der chroot-Umgebung. Danach werden nur die Variablen `HOME`, `TERM`, `PS1` und `PATH` wieder gesetzt. `TERM=$TERM` setzt die Variable `TERM` in der chroot-Umgebung auf den gleichen Wert wie außerhalb von chroot, diese Variable wird für das korrekte Funktionieren von Programmen wie `vim` und `less` benötigt. Wenn Sie weitere Variablen wie `CFLAGS` oder `CXXFLAGS` benötigen, ist dies ein guter Platz, um sie erneut zu setzen.

Von nun an brauchen Sie die Variable `LFS` nicht mehr, denn alle weiteren Befehle sind auf Ihr `LFS` beschränkt. Das was die laufende Shell für den Ordner `/` hält, ist in Wirklichkeit der Wert von `$LFS`, den Sie `chroot` oben als Parameter übergeben haben.

Beachten Sie, dass `/tools/bin` am Ende der Variable `PATH` steht. Das bewirkt, dass ein temporäres Werkzeug nicht mehr benutzt wird, sobald seine endgültige Version installiert ist. Zumindest, wenn die Shell sich nicht die Standorte von ausführbaren Dateien merkt—aus diesem Grund wird die Hash-Funktion der `bash` mit der Option `+h` abgeschaltet.

Sie müssen alle Kommandos in den folgenden Kapiteln in der chroot-Umgebung ausführen. Wenn Sie die chroot-Umgebung aus irgendeinem Grund verlassen (zum Beispiel wegen einem Neustart), dann denken Sie daran, die Dateisysteme `proc` und `devpts` einzuhängen (das wurde bereits im vorigen Abschnitt behandelt) *und* die chroot-Umgebung zu betreten, bevor Sie mit der Installation fortfahren.

Die Eingabeaufforderung der `Bash` wird `I have no name!` ausgeben. Das ist normal und liegt daran, dass die Datei `/etc/passwd` derzeit noch fehlt. Mit Hilfe dieser Datei findet nämlich auch die Zuordnung von Benutzer-IDs zu Benutzernamen statt.

6.4. Ändern des Besitzers

Im Augenblick gehört der Ordner `/tools`. Dieser existiert aber nur auf dem Host-System. Auch wenn Sie den Ordner `/tools` nach der fertigen Installation von LFS löschen möchten, ändern Sie vielleicht Ihre Meinung und heben ihn doch auf — z. B. um ein weiteres LFS-System zu installieren. Doch wenn Sie den Ordner `/tools` in seinem jetzigen Zustand behalten, gehören die Dateien einer Benutzer-ID zu der es kein Benutzerkonto gibt. Das ist gefährlich, denn ein später erstelltes Konto könnte genau diese ID erhalten und wäre damit der Besitzer von `/tools` und aller darin enthaltenen Dateien. Dieser Benutzer könnte alle Dateien unbemerkt manipulieren.

Um dieses Problem zu vermeiden, können Sie Ihrem LFS-System den Benutzer `lfs` später beim Erzeugen der `/etc/passwd` hinzufügen und ihm die gleiche Benutzer-ID und Gruppen-ID wie auf Ihrem Host-System geben. Alternativ können Sie `root` zum Besitzer des Ordners machen. Benutzen Sie dazu dieses Kommando:

```
chown -R 0:0 /tools
```

Das Kommando benutzt `0:0` anstelle von `root:root`, weil **chown** den Namen „root“ noch nicht auflösen kann, solange die Passwortdatei noch nicht erzeugt wurde. Wir gehen im weiteren Verlauf davon aus, dass Sie **chown** ausführen anstatt den Benutzer „lfs“ zu erstellen.

6.5. Erstellen der Ordnerstruktur

Nun bringen Sie ein wenig Struktur in das LFS-Dateisystem. Erzeugen Sie mit dem folgenden Kommando eine standardkonforme Ordnerstruktur:

```
install -dv /{bin,boot,dev,etc,opt,home,lib,mnt}
install -dv /{sbin,srv,usr/local,var,opt}
install -dv /root -m 0750
install -dv /tmp /var/tmp -m 1777
install -dv /media/{floppy,cdrom}
install -dv /usr/{bin,include,lib,sbin,share,src}
ln -sv share/{man,doc,info} /usr
install -dv /usr/share/{doc,info,locale,man}
install -dv /usr/share/{misc,terminfo,zoneinfo}
install -dv /usr/share/man/man{1,2,3,4,5,6,7,8}
install -dv /usr/local/{bin,etc,include,lib,sbin,share,src}
ln -sv share/{man,doc,info} /usr/local
install -dv /usr/local/share/{doc,info,locale,man}
install -dv /usr/local/share/{misc,terminfo,zoneinfo}
install -dv /usr/local/share/man/man{1,2,3,4,5,6,7,8}
install -dv /var/{lock,log,mail,run,spool}
install -dv /var/{opt,cache,lib/{misc,locate},local}
install -dv /opt/{bin,doc,include,info}
install -dv /opt/{lib,man/man{1,2,3,4,5,6,7,8}}
```

Normalerweise werden Ordner in der Voreinstellung mit den Rechten 755 erzeugt, aber das ist nicht bei allen Ordnern erwünscht. Nehmen Sie bitte zwei Änderungen vor: eine für den Persönlichen Ordner von *root* und eine weitere an den Ordnern für temporäre Dateien.

Die erste Rechteänderung bewirkt, dass nicht jeder den Ordner */root* betreten darf—das gleiche würde ein normaler Benutzer mit seinem Persönlichen Ordner auch tun. Die zweite Änderung sorgt dafür, dass jeder Benutzer in die Ordner */tmp* und */var/tmp* schreiben, aber nicht die Dateien anderer Benutzer löschen kann. Letzteres wird durch das „sticky bit“ bewirkt—dem höchsten Bit (1) in der Bit-Maske 1777.

6.5.1. Anmerkung zur FHS-Konformität

Unsere Ordnerstruktur basiert auf dem FHS-Standard (siehe <http://www.pathname.com/fhs/>). Zusätzlich zu den oben erstellten Ordnern sieht dieser Standard auch die Existenz von */usr/local/games* und */usr/share/games* vor, aber diese möchten wir in einem Basis-System eigentlich nicht haben. Wenn Sie möchten, können Sie Ihr System natürlich vollständig FHS-konform machen. Zur Struktur in */usr/local/share* macht FHS keine präzisen Angaben, daher haben wir die Ordner erstellt, die wir für nötig halten.

6.6. Erstellen notwendiger symbolischer Links

Einige Programme haben fest eingestellte Pfade zu Programmen, die zum jetzigen Zeitpunkt aber noch nicht existieren. Deshalb erstellen Sie eine Reihe symbolischer Links, die im weiteren Verlauf des Kapitels beim Installieren der restlichen Software durch echte Dateien ersetzt werden.

```
ln -sv /tools/bin/{bash,cat,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv bash /bin/sh
```

6.7. Erstellen der Dateien `passwd`, `group` und der Logdateien

Damit `root` sich am System anmelden kann und damit der Name „`root`“ der richtigen Benutzer-ID zugeordnet werden kann, müssen die entsprechenden Einträge in `/etc/passwd` und `/etc/group` vorhanden sein.

Erzeugen Sie `/etc/passwd` mit dem folgenden Kommando:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
EOF
```

Das tatsächliche Passwort für `root` (Das „`x`“ ist hier nur Platzhalter) wird erst später gesetzt.

Erstellen Sie `/etc/group` mit dem folgenden Kommando:

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

Die erzeugten Gruppen sind nicht Teil irgendeines Standards—es sind die Gruppen, die Udev in diesem Kapitel benutzt. Neben der Gruppe „`root`“ mit der GID 0 schlägt die LSB (*Linux Standard Base*) nur die Gruppe „`bin`“ mit der GID 1 vor. Alle anderen Gruppennamen und GIDs können durch den Anwender frei gewählt werden, weil gut geschriebene Pakete sich nicht auf GID-Nummern verlassen sollten, sondern den Gruppennamen verwenden.

Um die Meldung „I have no name!“ loszuwerden, starten Sie eine neue Shell. Die Auflösung von Benutzer- und Gruppennamen funktioniert sofort nach dem Erstellen von `/etc/passwd` und `/etc/group`, weil Sie in Kapitel 5 eine vollständige Glibc installiert haben.

```
exec /tools/bin/bash --login +h
```

Beachten Sie die Option `+h`. Durch sie wird das interne Pfad-Hashing der **Bash** abgeschaltet. Ohne diese Anweisung würde sich **bash** die Pfade zu ausführbaren Dateien merken und wiederverwenden. Weil die frisch installierten Programme aber sofort nach deren Installation an ihrem neuen Ort genutzt werden sollen, schalten Sie die Funktion für dieses Kapitel aus.

Die Programme **login**, **agetty**, und **init** (und einige weitere) verwenden Logdateien zum Protokollieren von Informationen. Dazu gehört z. B. wer sich zu welcher Zeit an das System angemeldet hat. Diese Programme protokollieren aber nur, wenn die entsprechenden Logdateien bereits existieren. Daher müssen Sie die Logdateien nun anlegen und die richtigen Rechte vergeben:

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}  
chgrp -v utmp /var/run/utmp /var/log/lastlog  
chmod -v 664 /var/run/utmp /var/log/lastlog
```

Die Logdateien haben folgenden Zweck: `/var/run/utmp` protokolliert zur Zeit angemeldete Benutzer. `/var/log/wtmp` protokolliert alle An- und Abmeldungen. `/var/log/lastlog` protokolliert die letzte Anmeldung für jeden Benutzer. `/var/log/btmp` protokolliert fehlgeschlagene Anmeldeversuche.

6.8. Bestücken von /dev mit Gerätedateien

6.8.1. Erzeugen der wichtigsten Gerätedateien

Zum Booten des Kernel müssen nur wenige Gerätedateien vorhanden sein, im einzelnen `console` und `null`. Die Gerätedateien werden auf der Festplatte erzeugt, damit sie vor dem Start von `udev` und im Einbenutzer-Modus verfügbar sind (deshalb die restriktiven Berechtigungen von `console`). Erstellen Sie die Gerätedateien mit diesen Kommandos:

```
mknod -m 600 /dev/console c 5 1
mknod -m 666 /dev/null c 1 3
```

6.8.2. Einhängen von tmpfs und Bestücken von /dev

Der empfohlene Weg zum Bestücken von `/dev` mit Geräten ist, in `/dev` ein virtuelles Dateisystem wie z. B. `tmpfs` einzuhängen und die Geräte dynamisch zu erzeugen sobald sie erkannt oder verwendet werden. Die meisten Geräte werden beim Booten erkannt und erzeugt. Weil das neue System aber bislang noch nicht gebootet wurde, müssen Sie die Arbeit der Bootskripte erstmal selbst erledigen. Hängen Sie nun `/dev` ein:

```
mount -nvt tmpfs none /dev
```

Die Geräte in `/dev` werden normalerweise dynamisch von Udev erzeugt. Weil Udev aber erst später installiert wird, müssen Sie die wichtigsten Geräte an dieser Stelle von Hand einrichten:

```
mknod -m 622 /dev/console c 5 1
mknod -m 666 /dev/null c 1 3
mknod -m 666 /dev/zero c 1 5
mknod -m 666 /dev/ptmx c 5 2
mknod -m 666 /dev/tty c 5 0
mknod -m 444 /dev/random c 1 8
mknod -m 444 /dev/urandom c 1 9
chown -v root:tty /dev/{console,ptmx,tty}
```

Die LFS-Bootskripte erzeugen während dem Bootvorgang einige Ordner und Links. Derzeit arbeiten Sie aber in einer chroot-Umgebung und die Bootskripte wurden noch nie gestartet, daher müssen diese Ordner und Links von Hand erstellt werden:

```
ln -sv /proc/self/fd /dev/fd
ln -sv /proc/self/fd/0 /dev/stdin
ln -sv /proc/self/fd/1 /dev/stdout
ln -sv /proc/self/fd/2 /dev/stderr
ln -sv /proc/kcore /dev/core
mkdir -v /dev/pts
mkdir -v /dev/shm
```

Schließlich hängen Sie das korrekte virtuelle (Kernel-) Dateisystem in die frisch erzeugten Ordner ein:

```
mount -vt devpts -o gid=4,mode=620 none /dev/pts
mount -vt tmpfs none /dev/shm
```

Das obige **mount**-Kommando könnte diese Warnmeldung ausgeben:

```
can't open /etc/fstab: No such file or directory.
```

Diese Datei—`/etc/fstab`—wurde bisher noch nicht erstellt, wird aber zum korrekten Einbinden der Dateisysteme auch nicht benötigt. Daher können Sie die Warnung problemlos ignorieren.

6.9. Linux-Libc-Header-2.6.11.2

Das Paket Linux-Libc-Header enthält die „bereinigten“ Header-Dateien des Linux-Kernels

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 26.9 MB

Die Installation ist abhängig von: Coreutils

6.9.1. Installation von Linux-Libc-Header

Über Jahre hinweg war es gängige Praxis, die „rohen“ Kernel-Header (direkt aus dem Kernel-Archiv) in `/usr/include` zu benutzen. Aber in den letzten Jahren haben die Kernel-Entwickler die Haltung eingenommen, dass man dies nicht tun sollte. Daraus entstand das Projekt Linux-Libc-Header. Es wurde entworfen um eine konsistente Version der Kernel-Header Programmierschnittstelle (API) zu bewahren.

Installieren Sie die Header-Dateien:

```
cp -Rv include/asm-i386 /usr/include/asm
cp -Rv include/linux /usr/include
```

Stellen Sie sicher, dass die Header im Besitz von root sind:

```
chown -Rv root:root /usr/include/{asm,linux}
```

Stellen Sie sicher, dass normale Benutzer Leserechte auf die Header haben:

```
find /usr/include/{asm,linux} -type d -exec chmod -v 755 {} \;
find /usr/include/{asm,linux} -type f -exec chmod -v 644 {} \;
```

6.9.2. Inhalt von Linux-Libc-Header

Installierte Header: `/usr/include/{asm,linux}/*.h`

Kurze Beschreibungen

`/usr/include/{asm,linux}/*.h` Diese Dateien bilden die Linux Header-API.

6.10. Man-pages-2.01

Das Paket Man-pages enthält über 1.200 Hilfeseiten.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 25.8 MB

Die Installation ist abhängig von: Bash, Coreutils und Make

6.10.1. Installation der Man-pages

Installieren Sie die Man-pages durch Ausführen von:

```
make install
```

6.10.2. Inhalt von Man-pages

Installierte Dateien: verschiedene Hilfeseiten (Man-pages)

Kurze Beschreibungen

Hilfeseiten Sie beschreiben z. B. C-/C++-Funktionen und wichtige Geräte- und Konfigurationsdateien.

6.11. Glibc-2.3.4

Glibc enthält die C-Bibliothek. Sie stellt Systemaufrufe und grundlegende Funktionen zur Verfügung (z. B. das Zuweisen von Speicher, Durchsuchen von Ordnern, Öffnen und Schließen sowie Schreiben von Dateien, Zeichenkettenverarbeitung, Mustererkennung, Arithmetik etc.)

Geschätzte Kompilierzeit: 12.3 SBU

Ungefähr benötigter Festplattenplatz: 476 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed und Texinfo

6.11.1. Installation von Glibc



Anmerkung

Einige Pakete außerhalb von LFS empfehlen, die GNU-Software `libiconv` zu installieren, um Daten von einer Kodierung in eine andere umzuwandeln. Auf der Webseite des Projektes unter <http://www.gnu.org/software/libiconv/> wird gesagt: „This library provides an `iconv()` implementation, for use on systems which don't have one, or whose implementation cannot convert from/to Unicode.“ Glibc enthält eine `iconv()`-Funktion und kann auch von/nach Unicode konvertieren, deshalb wird `libiconv` auf einem LFS-System nicht benötigt.

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch `-march` und `-mcpu`.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. `CFLAGS` und `CXXFLAGS`) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Das Installationssystem der Glibc ist sehr unabhängig und lässt sich perfekt installieren, selbst wenn die `specs`-Datei unseres Compilers und der Linker immer noch auf `/tools` verweisen. Sie können die `specs`-Datei und den Linker nicht vor der Installation von Glibc modifizieren, weil die Autoconf-Tests von Glibc dann falsche Resultate ergeben würden.

Das Tar-Archiv von Linxthreads enthält unter anderem auch die Man-Pages für die von Glibc installierte Threading-Bibliothek. Entpacken Sie das Tar-Archiv innerhalb des GLibc Quellordners:

```
tar -xjvf ../glibc-linuxthreads-2.3.4.tar.bz2
```

In bestimmten seltenen Fällen kann die Glibc abstürzen, wenn keine Standardsuchpfade existieren. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../glibc-2.3.4-rtld_search_dirs-1.patch
```

Glibc enthält zwei Testsuites die fehlschlagen, wenn der laufende Kernel 2.6.11.x ist. Das Problem sind hier die Tests selbst, nicht die libc oder der Kernel. Wenn Sie die Testsuite durchlaufen lassen möchten, wenden Sie diesen Patch an:

```
patch -Np1 -i ../glibc-2.3.4-fix_test-1.patch
```

Wenden Sie den folgenden Patch an. Er behebt einen Fehler in der Glibc, der einige Programme am korrekten

Start hindert (darunter z. B. OpenOffice.org):

```
patch -Np1 -i ../glibc-2.3.4-tls_assert-1.patch
```

Die Dokumentation von Glibc empfiehlt, zum Kompilieren einen gesonderten Ordner zu verwenden:

```
mkdir -v ../glibc-build
cd ../glibc-build
```

Bereiten Sie Glibc zum Kompilieren vor:

```
../glibc-2.3.4/configure --prefix=/usr \
  --disable-profile --enable-add-ons \
  --enable-kernel=2.6.0 --libexecdir=/usr/lib/glibc
```

Die Bedeutung der neuen Parameter zu configure:

```
--libexecdir=/usr/lib/glibc
  Dadurch wird das Programm pt_chown in /usr/lib/glibc anstelle von /usr/libexec installiert.
```

Kompilieren Sie das Paket:

```
make
```



Wichtig

In diesem Abschnitt wird die Testsuite von Glibc als *absolut kritisch* betrachtet. Sie sollten diesen Schritt *unter keinen Umständen überspringen*.

Testen Sie das Ergebnis:

```
make -k check >glibc-check-log 2>&1
grep Error glibc-check-log
```

Die Testsuite von Glibc ist sehr stark von einigen Funktionen Ihres Host-Systems abhängig, insbesondere dem Kernel. Grundsätzlich wird erwartet, dass die Testsuite fehlerfrei durchläuft. Nichtsdestotrotz können Fehler unter bestimmten Umständen manchmal nicht vermieden werden. Hier ist eine Liste der uns allgemein bekannten Probleme:

- Der *math*-Test schlägt manchmal fehl, wenn Sie ein System mit einer älteren Intel- oder AMD-CPU verwenden. Bestimmte Optimierungseinstellungen haben hier ebenfalls einen gewissen Einfluss.
- Der *gettext*-Test schlägt manchmal aufgrund von Host-bedingten Problemen fehl. Die genauen Ursachen sind bislang nicht ganz geklärt.
- Falls Sie die LFS-Partition mit der Option *noatime* eingehängt haben, wird der *atime*-Test fehlschlagen. Wie schon unter Abschnitt 2.4, „Einhängen (mounten) der neuen Partition“ erwähnt wurde, sollten Sie den Parameter *noatime* beim Bau von LFS nicht verwenden.
- Auf alter oder langsamer Hardware können einige Tests aufgrund von Zeitüberschreitungen fehlschlagen.

Auch wenn es nur eine harmlose Meldung ist, die Installationsroutine von Glibc wird sich über die fehlende Datei `/etc/ld.so.conf` beschweren. Beheben Sie diese störende Warnung mit:

```
touch /etc/ld.so.conf
```

Installieren Sie das Paket:

```
make install
```

Die Locales wurden durch das obige Kommando nicht mitinstalliert. Holen Sie das nun nach:

```
make localedata/install-locales
```

To save time, an alternative to running the previous command (which generates and installs every locale listed in the `glibc-2.3.4/localedata/SUPPORTED` file) is to install only those locales that are wanted and needed. This can be achieved by using the **localedef** command. Information on this command is located in the `INSTALL` file in the Glibc source. However, there are a number of locales that are essential in order for the tests of future packages to pass, in particular, the *libstdc++* tests from GCC. The following instructions, instead of the *install-locales* target used above, will install the minimum set of locales necessary for the tests to run successfully:

```
mkdir -pv /usr/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Some locales installed by the **make localedata/install-locales** command above are not properly supported by some applications that are in the LFS and BLFS books. Because of the various problems that arise due to application programmers making assumptions that break in such locales, LFS should not be used in locales that utilize multibyte character sets (including UTF-8) or right-to-left writing order. Numerous unofficial and unstable patches are required to fix these problems, and it has been decided by the LFS developers not to support such complex locales at this time. This applies to the `ja_JP` and `fa_IR` locales as well—they have been installed only for GCC and Gettext tests to pass, and the **watch** program (part of the Procps package) does not work properly in them. Various attempts to circumvent these restrictions are documented in internationalization-related hints.

Erzeugen Sie nun die Man-pages von `linuxthreads`, sie sind eine großartige Referenz zur Threading-Programmierschnittstelle (auch auf NPTL anwendbar):

```
make -C ../glibc-2.3.4/linuxthreads/man
```

Installieren Sie diese:

```
make -C ../glibc-2.3.4/linuxthreads/man install
```

6.11.2. Einrichten von Glibc

Sie müssen die Datei `/etc/nsswitch.conf` erstellen. Glibc gibt zwar Standardwerte vor wenn die Datei fehlt oder beschädigt ist, aber diese funktionieren nicht besonders gut mit Netzwerken. Außerdem müssen Sie die Zeitzone korrekt einstellen.

Erstellen Sie nun die Datei `/etc/nsswitch.conf`:

```
cat > /etc/nsswitch.conf << "EOF"
# Begin /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

protocols: files
services: files
ethers: files
rpc: files

# End /etc/nsswitch.conf
EOF
```

Mit diesem Skript finden Sie heraus, in welcher Zeitzone Sie sich befinden:

```
tzselect
```

Nachdem Sie ein paar Fragen zu Ihrem Standort beantwortet haben, wird das Skript den Namen Ihrer Zeitzone ausgeben. Die Ausgabe könnte z. B. *EST5EDT* oder *Canada/Eastern* lauten. Erstellen Sie dann die Datei `/etc/localtime`, indem Sie folgendes ausführen:

```
cp -v --remove-destination /usr/share/zoneinfo/[xxx] \
  /etc/localtime
```

Anstelle von `[xxx]` müssen Sie natürlich den Namen der Zeitzone einsetzen, der Ihnen von **tzselect** ausgegeben wurde (z. B. *Canada/Eastern*).

Die Bedeutung der Option zu `cp`:

```
--remove-destination
```

Dadurch wird das Entfernen des bereits vorhandenen symbolischen Links erzwungen. Sie ersetzen den Link durch eine Kopie der echten Datei, weil wir den Fall abdecken wollen, das `/usr` auf einer separaten Partition liegen könnte. Das würde z. B. dann problematisch werden, wenn der Single-User-Modus gebootet wird.

6.11.3. Einrichten des dynamischen Laders

Per Voreinstellung sucht der dynamische Lader (`/lib/ld-linux.so.2`) in `/lib` und `/usr/lib` nach den dynamischen Bibliotheken, die zur Laufzeit von ausführbaren Programmen benötigt werden. Wenn die benötigten Bibliotheken allerdings außerhalb von `/lib` und `/usr/lib` liegen, müssen Sie diese Ordner in `/etc/ld.so.conf` eintragen, damit der dynamische Lader sie finden kann. Zwei Ordner sind dafür bekannt, weitere Bibliotheken zu enthalten: `/usr/local/lib` und `/opt/lib`. Diese Ordner fügen Sie gleich mit in den Suchpfad ein.

Erstellen Sie die neue Datei `/etc/ld.so.conf` mit dem folgenden Kommando:

```
cat > /etc/ld.so.conf << "EOF"
# Begin /etc/ld.so.conf

/usr/local/lib
/opt/lib

# End /etc/ld.so.conf
EOF
```

6.11.4. Inhalt von Glibc

Installierte Programme: `catchsegv`, `gencat`, `getconf`, `getent`, `iconv`, `iconvconfig`, `ldconfig`, `ldd`, `lddlibc4`, `locale`, `localedef`, `mtrace`, `nscd`, `nscd_nischeck`, `pcprofiledump`, `pt_chown`, `rpcgen`, `rpcinfo`, `sln`, `sprof`, `tzselect`, `xtrace`, `zdump` und `zic`

Installierte Bibliotheken: `ld.so`, `libBrokenLocale.[a,so]`, `libSegFault.so`, `libanl.[a,so]`, `libbsd-compat.a`, `libc.[a,so]`, `libcrypt.[a,so]`, `libdl.[a,so]`, `libg.a`, `libieee.a`, `libm.[a,so]`, `libmcheck.a`, `libmemusage.so`, `libnsl.a`, `libnss_compat.so`, `libnss_dns.so`, `libnss_files.so`, `libnss_hesiod.so`, `libnss_nis.so`, `libnss_nisplus.so`, `libpcprofile.so`, `libpthread.[a,so]`, `libresolv.[a,so]`, `librpcsvc.a`, `librt.[a,so]`, `libthread_db.so` und `libutil.[a,so]`

Kurze Beschreibungen

catchsegv	Kann zum Erzeugen eines Stacktrace benutzt werden (falls ein Programm mit einem Speicherzugriffsfehler abstürzt).
gencat	Erzeugt Nachrichtenkataloge.
getconf	Zeigt System-Konfigurationswerte für dateisystemspezifische Variablen an.
getent	Liest Einträge aus einer administrativen Datenbank.
iconv	Führt Zeichensatzkonvertierungen durch.
iconvconfig	Erzeugt schnellladende iconv Konfigurationsdateien.
ldconfig	Richtet die Laufzeitbindungen des dynamischen Linkers ein.
ldd	Gibt aus, welche gemeinsamen Bibliotheken von einem Programm oder einer Bibliothek benötigt werden.
lddlibc4	Unterstützt ldd bei der Arbeit mit Objektdateien.
locale	Ein Perl-Programm, das im Compiler die Verwendung von POSIX-Locales für

	eingebaute Operationen ein- bzw. ausschaltet.
localedef	Erzeugt Locale-Spezifikationen.
mtrace	Liest und interpretiert eine Speicher-Rückverfolgungsdatei und gibt eine normal lesbare Zusammenfassung aus.
nscd	Der „name service cache daemon“; er stellt einen Zwischenspeicher für die meisten namensbasierten Anfragen zur Verfügung.
nscd_nischeck	Prüft, ob für NIS+-Anfragen der sichere Modus benötigt wird.
pcprofiledump	Gibt Informationen aus, die durch PC-Profiling erzeugt wurden.
pt_chown	Ist ein Hilfsprogramm zu grantpt . Es setzt Besitzer, Gruppe und Zugriffsberechtigungen von Slave-Pseudo-Terminals.
rpcgen	Erzeugt C-Code zum Implementieren des RPC-Protokolls.
rpcinfo	Generiert eine RPC-Anfrage an einen RPC-Server.
sln	Dies ist die statisch gelinkte Variante von ln .
sprof	Liest Profiling-Daten zu Shared-Objects und zeigt sie an.
tzselect	Stellt dem Anwender einige Fragen zu seinem Standort und erzeugt aus den Antworten eine passende Zeitzonenbeschreibung.
xtrace	Verfolgt den Durchlauf eines Programmes, indem es die jeweils ausgeführte Funktion ausgibt.
zdump	Gibt Zeitzonen aus.
zic	Ist ein Compiler für Zeitzonen.
<code>ld.so</code>	Ist ein Hilfsprogramm für ausführbare gemeinsame Bibliotheken.
<code>libBrokenLocale</code>	Wird von Programmen wie z. B. Mozilla verwendet, um Probleme mit defekten Locales zu beheben.
<code>libSegFault</code>	Kümmert sich um die Verarbeitung von Speicherzugriffsfehlern.
<code>libanl</code>	Eine Bibliothek zum asynchronen Nachschlagen von Namen.
<code>libbsd-compat</code>	Mit Hilfe dieser Bibliothek können einige BSD-Programme unter Linux lauffähig gemacht werden.
<code>libc</code>	Dies ist die C-Bibliothek.
<code>libcrypt</code>	Dies ist die Kryptographie-Bibliothek.
<code>libdl</code>	Eine Schnittstellenbibliothek zum dynamischen Linker.
<code>libg</code>	Eine Laufzeitbibliothek zu g++ .
<code>libieee</code>	Die Fließkomma-Bibliothek des Institute of Electrical and Electronic Engineers (IEEE).
<code>libm</code>	Die mathematische Bibliothek.

<code>libmcheck</code>	Enthält Code, der beim Booten ausgeführt wird.
<code>libmemusage</code>	Wird von memusage verwendet und hilft beim Sammeln von Informationen über die Speichernutzung eines Programms.
<code>libnsl</code>	Dies ist die Bibliothek für Netzwerkdienste.
<code>libnss</code>	Die Name Service Switch-Bibliotheken. Sie enthalten Funktionen zum Auflösen von Hostnamen, Benutzernamen, Gruppennamen, Aliasen, Diensten, Protokollen und so weiter.
<code>libpcprofile</code>	Enthält Profiling-Funktionen, die zum Verfolgen der CPU-Benutzung einzelner Quelltextzeilen verwendet werden können.
<code>libpthread</code>	Die POSIX-Threads-Bibliothek.
<code>libresolv</code>	Enthält Funktionen zum Erzeugen, Senden und Auswerten von Paketen an Internet Domain Name Server (DNS).
<code>librpcsvc</code>	Enthält Funktionen, die verschiedene RPC-Dienste zur Verfügung stellen.
<code>librt</code>	Diese Bibliothek enthält Funktionen mit Schnittstellen für die meisten POSIX.1b Echtzeiterweiterungen.
<code>libthread_db</code>	Enthält Funktionen, die zum Erzeugen von Debuggern für Multi-Thread-Programme nützlich sind.
<code>libutil</code>	Enthält Code für „Standard“-Funktionen, die in vielen verschiedenen Unix-Werkzeugen genutzt werden.

6.12. Erneutes Anpassen der Toolchain

Nachdem die neue C-Bibliothek nun installiert ist, muss die Toolchain erneut angepasst werden. Modifizieren Sie sie so, dass alle weiteren kompilierten Programme gegen die neue C-Bibliothek gelinkt werden. Im Grunde ist das das gleiche, was Sie im vorigen Kapitel beim Anpassen der Glibc schonmal gemacht haben, auch wenn es aussieht, als wäre es genau umgekehrt: Im vorigen Kapitel haben Sie die Toolchain von `{,usr}/lib` auf dem Host in den neuen Ordner `/tools/lib` umgelenkt. Nun lenken Sie die Toolchain von diesem Ordner `/tools/lib` um auf unsere LFS-Ordner `{,usr}/lib`.

Beginnen Sie mit dem Anpassen des Linkers. Dies ist der Grund dafür, dass wir die Quell- und Kompilierordner aus dem zweiten Durchlauf von Binutils nicht gelöscht haben. Wechseln Sie in den Ordner `binutils-build` und installieren Sie von dort den angepassten Linker:

```
make -C ld INSTALL=/tools/bin/install install
```



Anmerkung

Falls Sie aus irgendeinem Grund die Warnung übersehen haben, den Binutils-Ordner zu behalten, oder ihn vielleicht versehentlich gelöscht haben, ist noch nichts verloren. Ignorieren Sie einfach das obige Kommando. Das bewirkt, dass das nächste Paket, Binutils, gegen die Glibc-Bibliotheken in `/tools` anstelle von `{,usr}/lib` gelinkt wird. Das ist zwar nicht ideal, aber unsere Tests haben gezeigt, dass die resultierenden Programme identisch zu sein scheinen.

Von nun an wird jedes kompilierte Programm *nur* noch gegen die Bibliotheken in `/usr/lib` und `/lib` gelinkt. Das zusätzliche `INSTALL=/tools/bin/install` ist nötig, weil das Makefile aus dem zweiten Durchlauf immer noch die Referenz auf `/usr/bin/install` enthält, welches wir aber noch nicht installiert haben. Einige Distributionen enthalten einen symbolischen Link `ginstall`, der Vorrang im Makefile hat und hier Probleme verursachen kann. Das obige Kommando kümmert sich auch darum.

Sie können nun die Binutils Quell- und Kompilierordner löschen.

Als nächstes müssen Sie GCCs `specs`-Datei so bearbeiten, dass sie den neuen dynamischen Linker referenziert. Diese Aufgabe wird von einem einfachen `perl`-Kommando erledigt:

```
perl -pi -e 's@ /tools/lib/ld-linux.so.2@ /lib/ld-linux.so.2@g;' \
-e 's@*\startfile_prefix_spec:\n@$_/usr/lib/ @g;' \
`gcc --print-file specs`
```

Danach sollten Sie die `specs`-Datei überprüfen und sicherstellen, dass alle gewünschten Änderungen wirklich durchgeführt wurden.



Wichtig

Wenn Sie mit einer Plattform arbeiten, bei der der Name des Linkers nicht `ld-linux.so.2` ist, *müssen* Sie in den obigen Kommandos „`ld-linux.so.2`“ durch den Namen des Linkers für Ihre Plattform ersetzen. Wenn nötig, schlagen Sie nochmal im Abschnitt Abschnitt 5.2, „Technische Anmerkungen zur Toolchain,“ nach.



Achtung

An dieser Stelle ist es zwingend nötig, die grundlegenden Funktionen (Kompilieren und Linken) der angepassten Toolchain zu überprüfen. Aus diesem Grund führen Sie bitte den folgenden Test durch:

```
echo 'main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /lib'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos ist:

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

Beachten Sie, dass nun `/lib` der Prefix zum dynamischen Linker ist.

Wenn Sie eine andere oder überhaupt keine Ausgabe erhalten, ist etwas ernsthaft schiefgelaufen. Sie müssen das überprüfen und alle bisherigen Schritte noch einmal nachvollziehen, um das Problem zu finden und zu beheben. Machen Sie nicht weiter, solange das Problem nicht behoben ist. Am wahrscheinlichsten ist, dass etwas beim Anpassen der specs-Datei weiter oben nicht funktioniert hat.

Wenn Sie mit dem Ergebnis zufrieden sind, löschen Sie die Testdateien:

```
rm -v dummy.c a.out
```

6.13. Binutils-2.15.94.0.2.2

Binutils ist eine Sammlung von Software-Entwicklungswerkzeugen. Dazu gehören zum Beispiel Linker, Assembler und weitere Programme für die Arbeit mit Objektdateien.

Geschätzte Kompilierzeit: 1.3 SBU

Ungefähr benötigter Festplattenplatz: 158 MB

Die Installation ist abhängig von: Bash, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed und Texinfo

6.13.1. Installation von Binutils

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch *-march* und *-mcpu*.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. *CFLAGS* und *CXXFLAGS*) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Jetzt ist ein guter Zeitpunkt um sicherzustellen, dass die Pseudo-Terminals (PTYs) in Ihrer chroot-Umgebung funktionieren. Mit dem folgenden schnellen Test sehen Sie, ob alles korrekt eingerichtet ist:

```
expect -c "spawn ls"
```

Falls die folgende Meldung erscheint, ist Ihre chroot-Umgebung nicht für PTYs vorbereitet:

```
The system has no more ptys.
Ask your system administrator to create more.
```

Das Problem muss behoben werden, bevor Sie die Testsuites von Binutils und GCC laufen lassen.

Die Dokumentation zu Binutils empfiehlt, Binutils außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.15.94.0.2.2/configure --prefix=/usr \
--enable-shared
```

Kompilieren Sie das Paket:

```
make tooldir=/usr
```

Normalerweise ist *tooldir* (der Ordner, in den die ausführbaren Dateien endgültig installiert werden) auf $\$(exec_prefix)/\$(target_alias)$ eingestellt. Ein i686-Computer löst dies zum Beispiel zu */usr/i686-pc-linux-gnu* auf. Da wir aber nur für unser eigenes System installieren, brauchen wir diesen speziellen Ordner in */usr* nicht. Diese Konfiguration fände z. B. dann Verwendung, wenn das System zum Querkompilieren genutzt würde (zum Beispiel, um auf einer Intel-Maschine Code zu generieren, der auf einem PowerPC ausgeführt werden kann).

**Wichtig**

In diesem Abschnitt wird die Testsuite von Binutils als *kritisch* eingestuft. Wir raten Ihnen, die Tests unter keinen Umständen zu überspringen.

Testen Sie das Ergebnis:

```
make check
```

Installieren Sie das Paket:

```
make tooldir=/usr install
```

Installieren Sie die Header-Datei `libiberty`, sie wird von einigen Paketen benötigt:

```
cp -v ../binutils-2.15.94.0.2.2/include/libiberty.h /usr/include
```

6.13.2. Inhalt von Binutils

Installierte Programme: `addr2line`, `ar`, `as`, `c++filt`, `gprof`, `ld`, `nm`, `objcopy`, `objdump`, `ranlib`, `readelf`, `size`, `strings` und `strip`

Installierte Bibliotheken: `libiberty.a`, `libbfd.[a,so]` und `libopcodes.[a,so]`

Kurze Beschreibungen

addr2line	Konvertiert Programmadressen zu Dateinamen und Zeilennummern. Mit Hilfe des Programmnamens und einer Speicheradresse benutzt das Programm Debugging-Informationen in der ausführbaren Datei, um herauszufinden, welche Quelldatei und Zeilennummer mit der Adresse assoziiert ist.
ar	Wird zum Erzeugen und Extrahieren von Dateien aus einem Archiv verwendet.
as	Ein Assembler. Er assembliert die Ausgabe von gcc zu Objektdateien.
c++filt	Wird vom dynamischen Linker benutzt, um C++- und Java-Symbole aufzuschlüsseln, damit überladene Funktionen nicht in Konflikt geraten.
gprof	Zeigt call graph-Profilings-Daten an.
ld	Ein Linker. Er verbindet mehrere Objektdateien und Archivdateien zu einer einzigen Datei, replaziert ihre Daten und verbindet ihre Symbolreferenzen.
nm	Listet alle in einer Objektdatei vorkommenden Symbole auf.
objcopy	Wird zum Konvertieren eines bestimmten Objektdateityps in einen anderen verwendet.
objdump	Zeigt ausgewählte Informationen über eine Objektdatei an. Diese Informationen sind hauptsächlich für Programmierer sinnvoll, die an den Kompilierwerkzeugen arbeiten.
ranlib	Erzeugt einen Index des Archivinhalts und speichert ihn im Archiv. Der Index listet alle reallokierbaren Symbole auf, die von im Archiv enthaltenen Objektdateien definiert werden.
readelf	Zeigt Informationen über Binärdateien vom Typ ELF an.

size	Listet die Abschnitts- und Gesamtgröße für eine Objektdatei auf.
strings	Gibt für jede angegebene Datei die druckbaren Zeichenketten aus, die eine festgelegte Mindestgröße haben (Voreinstellung ist 4). Bei Objektdateien gibt es in der Voreinstellung nur die Zeichenketten aus den Initialisierungs- und Ladeabschnitten aus. Bei anderen Dateitypen durchsucht es die gesamte Datei.
strip	Entfernt bestimmte Symbole aus Objektdateien (z. B. Debugging-Symbole).
libiberty	Enthält Routinen, die von verschiedenen GNU-Programmen genutzt werden, inklusive getopt , obstack , strerror , strtol und strtoul .
libbfd	Die Bibliothek für Binärdateibezeichner.
libopcodes	Eine Bibliothek zur Behandlung von Obcodes. Sie wird zum Erzeugen von Werkzeugen wie z. B. objdump benutzt. Obcodes sind die „lesbaren“ Versionen der Prozessorinstruktionen.

6.14. GCC-3.4.3

Das Paket GCC enthält die GNU-Compiler-Sammlung. Darin sind die C- und C++-Compiler enthalten.

Geschätzte Kompilierzeit: 11.7 SBU

Ungefähr benötigter Festplattenplatz: 451 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed und Texinfo

6.14.1. Installation von GCC

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch `-march` und `-mcpu`.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. `CFLAGS` und `CXXFLAGS`) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Vorerst installieren Sie nur den No-Fixincludes-Patch (*nicht* den Specs-Patch!), den Sie auch im vorigen Kapitel benutzt haben:

```
patch -Np1 -i ../gcc-3.4.3-no_fixincludes-1.patch
```

In Kombination mit der neuen Version von Binutils hat GCC Probleme beim Kompilieren einiger Programme, die nicht zur LFS-Installation gehören (z. B. Mozilla und kdegraphics). Mit dem folgenden Patch wird das Problem behoben:

```
patch -Np1 -i ../gcc-3.4.3-linkonce-1.patch
```

Wenden Sie nun einen **Sed**-Befehl an um die Installation von `libiberty.a` zu verhindern. Wir möchten die von Binutils bereitgestellte Version von `libiberty.a` verwenden:

```
sed -i 's/install_to_${INSTALL_DEST} //' libiberty/Makefile.in
```

Die Dokumentation zu GCC empfiehlt, GCC außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-3.4.3/configure --prefix=/usr \
  --libexecdir=/usr/lib --enable-shared \
  --enable-threads=posix --enable-__cxa_atexit \
  --enable-clocale=gnu --enable-languages=c,c++
```

Kompilieren Sie das Paket:

```
make
```

**Wichtig**

In diesem Abschnitt wird die Testsuite als *absolut kritisch* betrachtet. Sie sollten diesen Schritt unter keinen Umständen überspringen.

Testen Sie die Ergebnisse, aber halten Sie bei Fehlern nicht an:

```
make -k check
```

Einige Fehler sind bekannt und wurden schon im vorigen Kapitel bemerkt. Die Anmerkungen aus Abschnitt 5.11, „GCC-3.4.3 - Durchlauf 2,“ sind hier immer noch gültig. Blättern Sie zurück, wenn nötig.

Installieren Sie das Paket:

```
make install
```

Einige Pakete erwarten, dass der C-Präprozessor unter `/lib` installiert ist. Erstellen Sie daher diesen symbolischen Link:

```
ln -sv ../usr/bin/cpp /lib
```

Viele Pakete benutzen den Namen `cc`, um den C-Compiler aufzurufen. Um auch diesen Paketen Rechnung zu tragen, erzeugen Sie einen weiteren symbolischen Link:

```
ln -sv gcc /usr/bin/cc
```

**Anmerkung**

An dieser Stelle ist es wichtig, den „Gesundheitscheck“, den Sie schon früher durchgeführt haben, erneut laufen zu lassen. Schlagen Sie im Abschnitt 6.12, „Erneutes Anpassen der Toolchain,“ nach und wiederholen Sie den Test. Wenn das Ergebnis negativ ist, haben Sie möglicherweise versehentlich den GCC-Specs-Patch aus Kapitel 5 angewendet.

6.14.2. Inhalt von GCC

Installierte Programme: `c++`, `cc` (Link auf `gcc`), `cpp`, `g++`, `gcc`, `gccbug` und `gcov`

Installierte Bibliotheken: `libgcc.a`, `libgcc_eh.a`, `libgcc_s.so`, `libstdc++.a`, `libstdc++.so` und `libsupc++.a`

Kurze Beschreibungen

<code>cc</code>	Dies ist der C-Compiler.
<code>cpp</code>	Ein C-Präprozessor. Er wird vom Compiler benutzt, um <code>#include</code> , <code>#define</code> und ähnliche Anweisungen im Quellcode durch ihren endgültigen Code zu ersetzen.
<code>c++</code>	Dies ist der C++-Compiler.
<code>g++</code>	Dies ist der C++-Compiler.
<code>gcc</code>	Dies ist der C-Compiler.

gccbug Ein Shellskript, mit dem man nützliche Fehlerberichte erzeugen kann.

- gcov** Ein Werkzeug zum Testen des Deckungsgrades. Es wird zum Analysieren von Programmen benutzt, um herauszufinden, wo Optimierungen den größten Effekt zeigen.
- libgcc** Enthält Laufzeitunterstützung für **gcc**.
- libstdc++** Die Standard-C++-Bibliothek.
- libsupc++** Stellt Unterstützungsroutinen für die Programmiersprache C++ zur Verfügung.

6.15. Coreutils-5.2.1

Das Paket Coreutils enthält viele Shell-Werkzeuge zum Einstellen der grundlegenden Systemeigenschaften.

Geschätzte Kompilierzeit: 0.9 SBU

Ungefähr benötigter Festplattenplatz: 52.8 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl und Sed

6.15.1. Installation von Coreutils

Die Funktion von **uname** ist bekannterweise ein wenig fehlerhaft, weil der Parameter `-p` immer `unknown` ausgibt. Der folgende Patch behebt das Problem auf Intel-Architekturen:

```
patch -Np1 -i ../coreutils-5.2.1-uname-2.patch
```

Normalerweise würde Coreutils einige Programme installieren, die später von anderen Paketen bereitgestellt werden sollen. Verhindern Sie die Installation dieser Programme mit diesem Patch:

```
patch -Np1 -i ../coreutils-5.2.1-suppress_uptime_kill_su-1.patch
```

Bereiten Sie Coreutils zum Kompilieren vor:

```
DEFAULT_POSIX2_VERSION=199209 ./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Diese Testsuite benötigt einige System-Benutzer und -Gruppen um korrekt zu funktionieren. Wenn Sie die Testsuite laufen lassen möchten, müssen Sie vorher die folgenden Befehle ausführen. Falls Sie die Testsuite nicht ausführen möchten überspringen Sie die Befehle und machen einfach bei „Installieren Sie das Paket“ fort.

Erstellen Sie zwei Dummy-Gruppen und einen Dummy-Benutzer:

```
echo "dummy1:x:1000:" >> /etc/group
echo "dummy2:x:1001:dummy" >> /etc/group
echo "dummy:x:1000:1000:::/bin/bash" >> /etc/passwd
```

Sie können die Testsuite nun durchlaufen lassen. Als erstes starten Sie einige Tests, die als *root* laufen müssen:

```
make NON_ROOT_USERNAME=dummy check-root
```

Die verbleibenden Tests werden als Benutzer *dummy* ausgeführt:

```
src/su dummy -c "make RUN_EXPENSIVE_TESTS=yes check"
```

Danach entfernen Sie die dummy Gruppen und Benutzer:

```
sed -i '/dummy/d' /etc/passwd /etc/group
```

Installieren Sie das Paket:

```
make install
```

Und verschieben Sie einige Programme an die von FHS vorgegebene Stelle:

```
mv -v /usr/bin/{cat,chgrp,chmod,chmod,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,hostname,ln,ls,mkdir,mknod,mv,pwd,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
```

Einige der LFS-Bootskripte sind abhängig von den Kommandos **head** und **sleep**. Da `/usr` in den früheren Phasen des Bootvorgangs noch nicht eingehängt sein könnte, müssen sich diese Programme auf der `root`-Partition befinden:

```
mv -v /usr/bin/{head,sleep} /bin
```

6.15.2. Inhalt von Coreutils

Installierte Programme: `basename`, `cat`, `chgrp`, `chmod`, `chown`, `chroot`, `cksum`, `comm`, `cp`, `csplit`, `cut`, `date`, `dd`, `df`, `dir`, `dircolors`, `dirname`, `du`, `echo`, `env`, `expand`, `expr`, `factor`, `false`, `fmt`, `fold`, `groups`, `head`, `hostid`, `hostname`, `id`, `install`, `join`, `link`, `ln`, `logname`, `ls`, `md5sum`, `mkdir`, `mkfifo`, `mknod`, `mv`, `nice`, `nl`, `nohup`, `od`, `paste`, `pathchk`, `pinky`, `pr`, `printenv`, `printf`, `ptx`, `pwd`, `readlink`, `rm`, `rmdir`, `seq`, `sha1sum`, `shred`, `sleep`, `sort`, `split`, `stat`, `stty`, `sum`, `sync`, `tac`, `tail`, `tee`, `test`, `touch`, `tr`, `true`, `tsort`, `tty`, `uname`, `unexpand`, `uniq`, `unlink`, `users`, `vdir`, `wc`, `who`, `whoami` und `yes`

Kurze Beschreibungen

basename	Entfernt den Pfad und Suffix von einem angegebenen Dateinamen.
cat	Gibt Dateien an der Standardausgabe aus bzw. fügt sie zusammen.
chgrp	Ändert die Gruppenzugehörigkeit von Dateien und Ordnern.
chmod	Ändert die Zugriffsrechte der angegebenen Dateien. Der Modus kann entweder symbolisch (in Form der durchzuführenden Änderungen) oder als Oktalzahl angegeben werden (repräsentiert die absoluten neuen Rechte).
chown	Ändert Besitzer und/oder Gruppenzugehörigkeit der angegebenen Dateien und Ordner.
chroot	Macht den angegebenen Ordner temporär zum neuen Basisordner („/“) für den übergebenen Befehl (z. B. bash). Der Befehl wird dann in diesem „Gefängnis“ ausgeführt.
cksum	Gibt die CRC-Prüfsumme (Cyclic Redundancy Check) und die Anzahl der Bytes einer angegebenen Datei aus.
comm	Vergleicht zwei sortierte Dateien und gibt in drei Spalten die Zeilen aus, die jeweils einzigartig bzw. gleich sind.
cp	Kopiert Dateien.
csplit	Teilt eine Datei in mehrere neue Dateien. Dazu wird ein bestimmtes Muster oder Zeilennummern verwendet. Außerdem gibt csplit die Anzahl Bytes der neuen Dateien aus.

- cut** Gibt Ausschnitte von Zeilen aus. Die Ausschnitte werden nach Feldern oder Positionsangaben gewählt.
- date** Gibt die aktuelle Zeit im angegebenen Format aus oder stellt die Systemzeit ein.
- dd** Kopiert eine Datei mit der angegebenen Blockgröße und -anzahl. Optional kann währenddessen eine Konvertierung durchgeführt werden.

df	Berichtet über den verfügbaren (und verwendeten) Festplattenspeicher auf allen eingehängten Dateisystemen oder den Dateisystemen, die die angegebenen Dateien enthalten.
dir	Listet den Inhalt eines Ordners auf (das Gleiche wie ls).
dircolors	Gibt Kommandos zum Setzen der Umgebungsvariable <code>LS_COLOR</code> aus, um damit das Farbschema von ls zu ändern.
dirname	Entfernt den nicht-ordnerspezifischen Teil eines Dateinamens.
du	Gibt aus, wieviel Festplattenspeicher der aktuelle Ordner, die Unterordner und Dateien oder eine einzelne Datei verbraucht.
echo	Gibt eine angegebene Zeichenkette aus.
env	Führt ein Kommando in einer modifizierten Arbeitsumgebung aus.
expand	Konvertiert Tabulatoren zu Leerzeichen.
expr	Wertet einen Ausdruck aus.
factor	Gibt den Primfaktor aller angegebenen Ganzzahlen aus.
false	Tut gar nichts, ist immer erfolglos. Es beendet sich immer mit einem Abschlusscode, der auf einen Fehler hinweist.
fmt	Formatiert die Absätze in der übergebenen Datei neu.
fold	Fügt Zeilenumbrüche in den angegebenen Dateien ein.
groups	Gibt die Gruppenzugehörigkeit eines Benutzers aus.
head	Gibt die ersten zehn (oder die angegebene Anzahl) von Zeilen einer Datei aus.
hostid	Gibt die numerische ID (hexadezimal) des Systems aus.
hostname	Setzt den Hostnamen bzw. zeigt ihn an.
id	Gibt die effektive Benutzer-ID, Gruppen-ID, und Gruppenzugehörigkeit des aktuellen Benutzers oder eines angegebenen Benutzers aus.
install	Kopiert Dateien und setzt deren Zugriffsrechte und, falls möglich, Besitzer und Gruppe.
join	Fügt aus zwei Dateien die Zeilen mit identischen join-Feldern zusammen.
link	Erzeugt einen harten Link von der angegebenen Datei zu einer Datei.
ln	Erzeugt einen harten oder symbolischen Link zwischen Dateien.
logname	Gibt den Login-Namen des aktuellen Benutzers aus.
ls	Listet den Inhalt des angegebenen Ordners auf.
md5sum	Erzeugt eine MD5-Prüfsumme (Message Digest 5) bzw. zeigt sie an.
mkdir	Erzeugt Ordner mit den angegebenen Namen.
mkfifo	Erzeugt FIFO's (First-In, First-Out, eine sogenannte "named Pipe" im UNIX Sprachgebrauch) mit dem angegebenen Namen.

mknod	Erzeugt eine Gerätedatei mit dem angegebenen Namen. Eine Gerätedatei ist eine spezielle zeichen- oder blockorientierte Datei oder ein FIFO.
mv	Verschiebt Dateien und Ordner oder benennt sie um.
nice	Führt ein Programm mit geänderter Priorität aus.
nl	Nummeriert die Zeilen der angegebenen Dateien.
nohup	Führt ein Programm aus, so dass es immun gegen „hangup“s ist. Die Ausgaben des Programms werden in eine Protokolldatei umgeleitet.
od	Gibt eine Datei oktal oder in anderen Formaten aus.
paste	Fügt angegebene Dateien zusammen. Sequenziell zusammengehörende Zeilen werden Seite an Seite durch Tabulatoren getrennt zusammengefügt.
pathchk	Prüft, ob Dateinamen gültig und portierbar sind.
pinky	Eine abgespeckte Version von finger. Es gibt ein paar Informationen über den angegebenen Benutzer aus.
pr	Bereitet Dateien seiten- oder spaltenweise für den Ausdruck vor.
printenv	Gibt die Umgebungsvariablen aus.
printf	Gibt die angegebenen Argumente in einem bestimmten Format aus — dies ist der C-Funktion printf sehr ähnlich.
ptx	Erzeugt aus dem Inhalt von Dateien einen vertauschten Index, mit jedem Stichwort im Kontext.
pwd	Gibt den Namen des aktuellen Arbeits-Ordners aus.
readlink	Gibt das Ziel eines symbolischen Links aus.
rm	Löscht Dateien oder Ordner.
rmdir	Löscht leere Ordner.
seq	Gibt eine Zahlenreihe in einem bestimmten Wertebereich und mit einem bestimmten Inkrement aus.
sha1sum	Prüft 160-Bit SHA1-Prüfsummen oder gibt sie aus.
shred	Überschreibt eine Datei mehrfach mit zufälligen Mustern, um das Wiederherstellen der Daten zu erschweren.
sleep	Pausiert für die angegebene Zeit.
sort	Sortiert die Zeilen einer Datei.
split	Teilt eine Datei in Stücke, nach Größe oder nach Zeilennummern.
stat	Zeigt den Datei- oder Dateisystemstatus an.
stty	Setzt Terminal-Einstellungen oder zeigt sie an.
sum	Gibt Prüfsumme und Anzahl der Blöcke einer Datei aus.
sync	Schreibt den Dateisystempuffer. Geänderte Blöcke werden auf die Festplatte geschrieben und

der Superblock wird aktualisiert.

tac	Fügt Dateien rückwärts zusammen.
tail	Gibt die letzten zehn (oder die angegebene Anzahl) von Zeilen einer Datei aus.
tee	Liest von der Standardeingabe während gleichzeitig auf die Standardausgabe und in eine Datei geschrieben wird.
test	Vergleicht Werte und prüft Dateitypen.
touch	Ändert Zeitstempel von Dateien, setzt Zugriffs- und Änderungszeit einer Datei auf die aktuelle Zeit. Dateien, die noch nicht existieren, werden mit der Länge 0 angelegt.
tr	Übersetzt, quetscht oder entfernt Zeichen von der Standardeingabe.
true	Macht nichts, ist immer erfolgreich. Beendet immer mit einem Statuscode, der Erfolg bedeutet.
tsort	Sortiert topologisch. Schreibt eine vollständig sortierte Liste entsprechend der teilweisen Sortierung in einer Datei.
tty	Gibt den Dateinamen des Terminals aus, das mit der Standardeingabe verbunden ist.
uname	Gibt Systeminformationen aus.
unexpand	Konvertiert Leerzeichen zu Tabulatoren.
uniq	Entfernt alle identischen Zeilen bis auf eine.
unlink	Entfernt eine Datei.
users	Gibt die Namen der eingeloggten Benutzer aus.
vdir	Macht das Gleiche wie ls -l .
wc	Gibt die Anzahl Zeilen, Wörter und Bytes einer Datei aus. Und eine Summe, falls mehrere Dateien angegeben wurden.
who	Zeigt an, wer gerade eingeloggt ist.
whoami	Gibt den Benutzernamen aus, der mit der aktuell effektiven Benutzer-ID verknüpft ist.
yes	Gibt „y“ oder eine andere Zeichenkette solange aus, bis es beendet wird.

6.16. Zlib-1.2.3

Die in Zlib enthaltenen Routinen werden von vielen Programmen zum Komprimieren und Dekomprimieren genutzt.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 3.1 MB

Die Installation ist abhängig von: Binutils, Coreutils, GCC, Glibc, Make und Sed

6.16.1. Installation von Zlib



Anmerkung

Vorsicht: Zlib baut seine gemeinsamen Bibliotheken falsch, wenn die Umgebungsvariable `CFLAGS` gesetzt ist. Falls Sie die Umgebungsvariable `CFLAGS` verwenden, fügen Sie ihr für den Durchlauf von `configure` den Wert `-fPIC` an und entfernen Sie ihn später wieder.

Bereiten Sie Zlib zum Kompilieren vor:

```
./configure --prefix=/usr --shared --libdir=/lib
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie `make check` aus.

Installieren Sie die gemeinsamen Bibliotheken:

```
make install
```

Das vorige Kommando hat eine `.so`-Datei im Ordner `/lib` installiert. Entfernen Sie sie wieder und erstellen Sie stattdessen einen Link in `/usr/lib`:

```
rm -v /lib/libz.so
ln -sfv ../../lib/libz.so.1.2.3 /usr/lib/libz.so
```

Kompilieren Sie nun die statische Bibliothek:

```
make clean
./configure --prefix=/usr
make
```

Um das Ergebnis zu prüfen, führen Sie `make check` aus.

Installieren Sie die statische Bibliothek:

```
make install
```

Und korrigieren Sie die Zugriffsrechte auf die statische Bibliothek:

```
chmod -v 644 /usr/lib/libz.a
```

6.16.2. Inhalt von Zlib

Installierte Bibliotheken: libz.[a,so]

Kurze Beschreibungen

`libz` Enthält Funktionen zum Komprimieren und Dekomprimieren, die von vielen Programmen genutzt werden.

6.17. Mktmp-1.5

Das Paket Mktmp enthält Programme zum sicheren Anlegen temporärer Dateien aus Shell-Skripten heraus.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 436 KB

Die Installation ist abhängig von: Coreutils, Make und Patch

6.17.1. Installation von Mktmp

Viele Skripte verwenden leider immer noch das missbilligte Programm **tempfile**, das die gleich Funktionalität hat wie **mktemp**. Patchen Sie mktemp, damit es auch einen Wrapper für **tempfile** installiert:

```
patch -Np1 -i ../mktmp-1.5-add_tempfile-2.patch
```

Bereiten Sie Mktmp zum Kompilieren vor:

```
./configure --prefix=/usr --with-libc
```

Die Bedeutung der configure-Parameter:

--with-libc

Dadurch benutzt **mktemp** die Funktionen *mkstemp* und *mkdtemp* aus der C-Bibliothek.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
make install-tempfile
```

6.17.2. Inhalt von Mktmp

Installierte Programme: mktemp und tempfile

Kurze Beschreibungen

mktemp Erzeugt temporäre Dateien auf sichere Weise. Es wird in Skripten verwendet.

tempfile Erzeugt temporäre Dateien auf weniger sichere Weise als **mktemp**. Es wird aus Gründen der Rückwärtskompatibilität installiert.

6.18. Iana-Etc-1.04

Das Paket Iana-Etc enthält Daten zu Netzwerkdiensten und Protokollen.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 1.9 MB

Die Installation ist abhängig von: Make

6.18.1. Installation von Iana-Etc

Das folgende Kommando konvertiert die von IANA bereitgestellten RAW-Daten in das korrekte Format für `/etc/protocols` und `/etc/services`:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.18.2. Inhalt von Iana-Etc:

Installierte Dateien: `/etc/protocols` und `/etc/services`

Kurze Beschreibungen

- | | |
|-----------------------------|---|
| <code>/etc/protocols</code> | Beschreibt verschiedene im TCP/IP-Subsystem verfügbare DARPA Internet-Protokolle. |
| <code>/etc/services</code> | Ermöglicht eine Zuordnung von leicht zu lesenden Namen für Internetdienste und den zugehörigen Port-Nummern und Protokolltypen. |

6.19. Findutils-4.2.23

Das Paket Findutils enthält Programme zum Auffinden von Dateien durch rekursive Suche in einer Ordnerstruktur oder über den Zugriff auf eine Datenbank. Die Suche über eine Datenbank ist normalerweise schneller, aber es besteht natürlich die Gefahr, dass die Datenbank zum Zeitpunkt der Suche veraltet ist.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 9.4 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

6.19.1. Installation von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

```
./configure --prefix=/usr --libexecdir=/usr/lib/locate \
--localstatedir=/var/lib/locate
```

Die Bedeutung der configure-Parameter:

--localstatedir

Der obige Parameter ändert den Standort der **locate**-Datenbank wie vom FHS-Standard verlangt nach `/var/lib/locate`.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.19.2. Inhalt von Findutils

Installierte Programme: bigram, code, find, frcode, locate, updatedb und xargs

Kurze Beschreibungen

bigram	Wurde früher zum Anlegen von locate -Datenbanken benutzt.
code	Wurde früher zum Anlegen von locate -Datenbanken benutzt. Es ist der Vorgänger von frcode .
find	Durchsucht eine Ordnerstruktur nach Dateien, die einem bestimmten Kriterium entsprechen.
frcode	Wird von updatedb aufgerufen, um die Liste der Dateinamen zu komprimieren. Durch die sog. front-Komprimierung wird die Datenbankgröße um den Faktor 4 bis 5 verkleinert.
locate	Durchsucht die locate -Datenbank mit Dateinamen und gibt die Dateien aus, die eine bestimmte

Zeichenkette enthalten oder auf ein bestimmtes Suchmuster passen.

updatedb Aktualisiert die **locate**-Datenbank. Es durchsucht das gesamte Dateisystem (inklusive anderer eingehängter Dateisysteme, wenn nicht anders angegeben) und trägt jeden gefundenen Dateinamen in die Datenbank ein.

xargs Kann benutzt werden, um ein bestimmtes Kommando auf eine Liste von Dateien anzuwenden.

6.20. Gawk-3.1.4

Gawk ist eine Implementierung von awk und wird zur Textmanipulation verwendet.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 16.4 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

6.20.1. Installation von Gawk

Bereiten Sie Gawk zum Kompilieren vor:

```
./configure --prefix=/usr --libexecdir=/usr/lib
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.20.2. Inhalt von Gawk

Installierte Programme: awk (Link auf gawk), gawk, gawk-3.1.4, grcat, igawk, pgawk, pgawk-3.1.4 und pwcacat

Kurze Beschreibungen

awk	Ein Link auf gawk .
gawk	Ein Programm zur Manipulation von Textdateien. Es ist die GNU-Implementierung von awk .
gawk-3.1.4	Ein harter Link auf gawk .
grcat	Zeigt die Gruppendatenbank <code>/etc/group</code> an.
igawk	Ermöglicht gawk das Einbinden von Dateien.
pgawk	Die Profiling-Version von gawk .
pgawk-3.1.4	Ein harter Link auf pgawk .
pwcacat	Zeigt die Passwortdatenbank <code>/etc/passwd</code> an.

6.21. Ncurses-5.4

Das Paket Ncurses enthält Bibliotheken für den Terminal-unabhängigen Zugriff auf Textbildschirme.

Geschätzte Kompilierzeit: 0.6 SBU

Ungefähr benötigter Festplattenplatz: 18.6 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

6.21.1. Installation von Ncurses

Bereiten Sie Ncurses zum Kompilieren vor:

```
./configure --prefix=/usr --with-shared --without-debug
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Vergeben Sie Ausführungsrechte für die Ncurses-Bibliothek:

```
chmod -v 755 /usr/lib/*.5.4
```

Korrigieren Sie eine Bibliothek, die nicht ausführbar sein sollte:

```
chmod -v 644 /usr/lib/libncurses++.a
```

Verschieben Sie die Bibliotheken in den Ordner `/lib`, denn es wird erwartet, dass sie sich dort befinden:

```
mv -v /usr/lib/libncurses.so.5* /lib
```

Da die Bibliotheken gerade verschoben wurden, zeigen ein paar symbolische Links nun ins Leere. Erstellen Sie diese symbolischen Links neu:

```
ln -sfv ../../lib/libncurses.so.5 /usr/lib/libncurses.so
ln -sfv libncurses.so /usr/lib/libcurses.so
```

6.21.2. Inhalt von Ncurses

Installierte Programme: captinfo (Link auf tic), clear, infocmp, infotocap (Link auf tic), reset (Link auf tset), tack, tic, toe, tput und tset

Installierte Bibliotheken: libcurses.[a,so] (Link auf libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so] und libpanel.[a,so]

Kurze Beschreibungen

captinfo	Konvertiert termcap-Beschreibungen zu terminfo-Beschreibungen.
clear	Löscht den Bildschirminhalt (wenn möglich).
infocmp	Vergleicht terminfo-Beschreibungen oder gibt sie aus.
infotocap	Konvertiert terminfo-Beschreibungen zu termcap-Beschreibungen.
reset	Setzt ein Terminal auf seine Voreinstellungen zurück.
tack	Wird benutzt, um die Korrektheit eines Eintrages in der terminfo-Datenbank zu überprüfen.
tic	Der Compiler für Beschreibungen zu terminfo-Einträgen. Er übersetzt terminfo-Dateien aus dem Quellformat in das binäre Format, das von den ncurses-Bibliotheksroutinen benötigt wird. Eine terminfo-Datei enthält Informationen über die Fähigkeiten eines bestimmten Terminals.
toe	Listet alle verfügbaren Terminaltypen auf und gibt zu jedem den Namen und die Beschreibung aus.
tput	Macht der Shell die Werte von Terminal-abhängigen Fähigkeiten zugänglich. Es kann auch zum Zurücksetzen oder Initialisieren eines Terminals oder zum Anzeigen seines vollständigen Namens verwendet werden.
tset	Kann zum Initialisieren eines Terminals verwendet werden.
libcurses	Ein Link auf libncurses.
libncurses	Enthält Funktionen zum Anzeigen von Text auf einem Terminal in vielen komplizierten Variationen. Ein gutes Beispiel ist das angezeigte Menü von make menuconfig des Kernels.
libform	Enthält Funktionen zum Implementieren von Formularen.
libmenu	Enthält Funktionen zum Implementieren von Menüs.
libpanel	Enthält Funktionen zum Implementieren von Schaltflächen.

6.22. Readline-5.0

Das Paket Readline enthält Bibliotheken die Unterstützung für einen Verlauf und das Bearbeiten von Kommandozeilen bereitstellen.

Geschätzte Kompilierzeit: 0.11 SBU

Ungefähr benötigter Festplattenplatz: 9.1 MB

Die Installation ist abhängig von: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses und Sed

6.22.1. Installation von Readline

Der folgende Patch behebt ein Problem, bei dem Readline manchmal nur 33 Zeichen einer Zeile anzeigt und dann zur nächsten Zeile springt. Zusätzlich sind noch weitere vom Readline-Autor empfohlene Fehlerbereinigungen enthalten.

```
patch -Np1 -i ../readline-5.0-fixes-1.patch
```

Bereiten Sie Readline zum Kompilieren vor:

```
./configure --prefix=/usr --libdir=/lib
```

Kompilieren Sie das Paket:

```
make SHLIB_XLDFLAGS=-lncurses
```

Die Bedeutung der make-Option:

```
SHLIB_XLDFLAGS=-lncurses
```

Dieser Parameter zwingt Readline, gegen die Bibliothek `libncurses` zu linken.

Installieren Sie das Paket:

```
make install
```

Vergeben Sie Readline's dynamischen Bibliotheken passendere Zugriffsrechte:

```
chmod -v 755 /lib/lib{readline,history}.so*
```

Nun verschieben Sie die statischen Bibliotheken an eine passendere Stelle:

```
mv -v /lib/lib{readline,history}.a /usr/lib
```

Als nächstes werden die `.so`-Dateien im Ordner `/lib` gelöscht und nach `/usr/lib` verlinkt:

```
rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.5 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.5 /usr/lib/libhistory.so
```

6.22.2. Inhalt von Readline

Installierte Bibliotheken: libhistory.[a,so] und libreadline.[a,so]

Kurze Beschreibungen

- | | |
|-------------|--|
| libhistory | Stellt eine konsistente Schnittstelle zum Wiederaufrufen von Zeilen aus dem Verlauf zur Verfügung. |
| libreadline | Kümmert sich um die Konsistenz der Benutzerschnittstelle bei Programmen, die eine Kommandozeilenoberfläche bereitstellen müssen. |

6.23. Vim-6.3

Das Paket Vim enthält einen sehr mächtigen Texteditor.

Geschätzte Kompilierzeit: 0.4 SBU

Ungefähr benötigter Festplattenplatz: 38.0 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses und Sed



Alternativen zu Vim

Wenn Sie einen anderen Editor bevorzugen—zum Beispiel Emacs, Joe oder Nano—dann schauen Sie unter <http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html>, dort finden Sie einige Installationshinweise.

6.23.1. Installation von Vim

Entpacken Sie zuerst beide Archivdateien—`vim-6.3.tar.bz2` und (optional) `vim-6.3-lang.tar.gz`—in den gleichen Ordner. Dann ändern Sie den voreingestellten Pfad zu der Konfigurationsdatei `vimrc` nach `/etc`:

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

Vim hat zwei Sicherheitsmängel, die dem Autor auch bereits mitgeteilt wurden. Der folgende Patch behebt die Probleme:

```
patch -Np1 -i ../vim-6.3-security_fix-2.patch
```

Bereiten Sie Vim zum Kompilieren vor:

```
./configure --prefix=/usr --enable-multibyte
```

Die Bedeutung der `configure`-Parameter:

`--enable-multibyte`

Dieser optionale—aber dringend empfohlene—Parameter schaltet die Unterstützung zum Editieren von Dateien mit Multibyte-Zeichenkodierung ein. Das wird benötigt, wenn Sie ein Locale mit Multibyte-Zeichensatz verwenden. Dieser Parameter ist auch hilfreich, wenn Sie Dateien bearbeiten möchten, die mit Distributionen wie z. B. Fedora Core erzeugt wurden. Diese Distributionen benutzen UTF-8 als voreingestellten Zeichensatz.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie das Ergebnis testen möchten, können Sie dazu `make test` verwenden. Die Testsuite gibt jedoch eine Menge sinnlose Zeichen auf dem Bildschirm aus und könnte die Einstellungen Ihres Terminals durcheinander bringen. Sie können die Ausgabe in eine Datei umleiten, um dieses Problem zu umgehen.

Installieren Sie das Paket:

```
make install
```

Viele Benutzer sind es gewöhnt, **vi** anstelle von **vim** zu starten. Damit **vim** gestartet wird, obwohl **vi** eingegeben wurde, erzeugen Sie einen symbolischen Link:

```
ln -sv vim /usr/bin/vi
```

Falls Sie später ein X Window-System auf Ihrem LFS installieren möchten, sollten Sie nach der Installation von X Ihren Vim erneut installieren. Vim bringt eine grafische Oberfläche mit, die allerdings X und ein paar weitere Bibliotheken voraussetzt. Weitere Informationen finden Sie in der Dokumentation zu Vim und im BLFS-Buch unter <http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim>.

6.23.2. Einrichten von Vim

In der Voreinstellung läuft **vim** im vi-inkompatiblen Modus. Das ist wahrscheinlich neu für Leute, die in der Vergangenheit andere Editoren verwendet haben. Die Einstellung „*nocompatible*“ ist dennoch unten aufgeführt, um daran zu erinnern, dass das neue Verhalten benutzt wird. Wenn Sie zum vi-kompatiblen Modus wechseln möchten, sollte „*compatible*“ im Kopfbereich der Datei stehen. Das ist nötig, weil diese Option viele Voreinstellungen für Parameter vornimmt. Ihre eigenen Änderungen an diesen Parametern müssen danach erfolgen, weil sie sonst von „*compatible*“ zurückgesetzt würden. Erzeugen Sie eine Standard vim-Konfigurationsdatei mit diesem Kommando:

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc

set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
    set background=dark
endif

" End /etc/vimrc
EOF
```

Der Parameter *set nocompatible* versetzt **vim** in einen nützlicheren Betriebsmodus (Voreinstellung) als den vi-kompatiblen Modus. Entfernen Sie das „no“ falls Sie das alte **vi**-Verhalten nutzen möchten. *set backspace=2* erlaubt das sogenannte Backspacing über Zeilenumbrüche hinweg, automatisches Einrücken und das Starten von Einrückungen. *syntax on* aktiviert **vims** Hervorheben von Syntax. Schließlich stellt die *if*-Verzweigung sicher, dass mittels *set background=dark* die Hintergrundfarbe von bestimmten Terminals besser eingestellt ist. Dadurch wird hervorgehobene Syntax in diesen Terminal-Emulatoren besser lesbar.

Die Dokumentation zu weiteren möglichen Optionen erhalten Sie mit diesem Kommando:

```
vim -c ':options'
```

6.23.3. Inhalt von Vim

Installierte Programme: efm_filter.pl, efm_perl.pl, ex (Link auf vim), less.sh, mve.awk, pltags.pl, ref, rview (Link auf vim), rvim (Link auf vim), shtags.pl, tcltags, vi (Link auf vim), view (Link auf vim), vim, vim132, vim2html.pl, vimdiff (Link auf vim), vimm, vimspell.sh, vimtutor und xxd

Kurze Beschreibungen

efm_filter.pl	Ein Filter zum Erzeugen einer Fehlerdatei, die von vim gelesen werden kann.
efm_perl.pl	Reformatiert Fehlermeldungen von Perl, um sie mit dem Quickfix-Modus von „vim“ benutzen zu können.
ex	Startet vim im ex-Modus.
less.sh	Ein Skript, welches vim mit less.vim startet.
mve.awk	Verarbeitet vim -Fehler.
pltags.pl	Erzeugt eine Markup-Datei für Perl-Code, die mit vim benutzt werden kann.
ref	Prüft die Schreibweise von Argumenten.
rview	Eine eingeschränkte Version von view : es gibt keine Shell-Kommandos und view kann nicht angehalten werden.
rvim	Eine eingeschränkte Version von vim : es gibt keine Shell-Kommandos und vim kann nicht angehalten werden.
shtags.pl	Erzeugt eine Markup-Datei für Perl-Skripte.
tcltags	Erzeugt eine Markup-Datei für TCL-Code.
view	Startet vim im Nur-lesen-Modus.
vi	Dies ist der Editor.
vim	Dies ist der Editor.
vim132	Startet vim in einem Terminal mit 132-Spalten-Modus.
vim2html.pl	Konvertiert Vim-Dokumentation zu HyperText Markup Language (HTML).
vimdiff	Editiert zwei oder drei Versionen einer Datei mit vim und zeigt die Unterschiede an.
vimm	Aktiviert das DEC Locator-Eingabemodell auf einem entfernten Terminal.
vimspell.sh	Untersucht eine Datei und erzeugt die nötigen Syntax-Regeln um das Hervorheben der Syntax in vim zu ermöglichen. Dieses Skript benötigt das alte Unix-Kommando spell , welches allerdings weder von LFS, noch von BLFS bereitgestellt wird.
vimtutor	Bringt Ihnen die wichtigsten Tastenbelegungen und Kommandos von vim bei.
xxd	Erzeugt eine Hex-Ausgabe einer Datei. Das geht auch umgekehrt und kann zum Patchen von Binärdateien benutzt werden.

6.24. M4-1.4.3

M4 enthält einen Makroprozessor.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.8 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl und Sed

6.24.1. Installation von M4

Bereiten Sie M4 zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.24.2. Inhalt von M4

Installiertes Programm: m4

Kurze Beschreibungen

m4 Kopiert die Eingabe zur Ausgabe und führt dabei Makros aus. Die Makros können entweder vordefiniert oder selbstgeschrieben sein und beliebige Argumente übernehmen. Neben der Fähigkeit, Makros auszuführen, besitzt **m4** eingebaute Funktionen zum Einfügen benannter Dateien, zum Ausführen von Unix-Befehlen und Integer-Berechnungen, zur Manipulation von Text und zur Behandlung von Rekursionen usw. **m4** kann entweder als Frontend zu einem Compiler oder als eigenständiger Makroprozessor genutzt werden.

6.25. Bison-2.0

Mit Bison lassen sich Programme generieren, die die Struktur einer Textdatei analysieren.

Geschätzte Kompilierzeit: 0.6 SBU

Ungefähr benötigter Festplattenplatz: 9.9 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make und Sed

6.25.1. Installation von Bison

Bereiten Sie Bison zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.25.2. Inhalt von Bison

Installierte Programme: bison und yacc

Installierte Bibliothek: liby.a

Kurze Beschreibungen

- bison** Erzeugt aus einer Reihe von Regeln ein Programm zum Analysieren der Struktur von Textdateien. Bison ist ein Ersatz für yacc (Yet Another Compiler Compiler).
- yacc** Ein Wrapper zu **bison**. Er wird benutzt, weil immer noch viele Programm **yacc** anstelle von **bison** aufrufen. **Bison** wird dann mit der Option `-y` aufgerufen.
- `liby.a` Die Yacc-Bibliothek, die die Implementierung von yacc-kompatiblen `yyerror`- und `main`-Funktionen enthält. Diese Bibliothek ist normalerweise nicht sehr nützlich, aber sie wird von POSIX vorausgesetzt.

6.26. Less-382

Less ist ein Textanzeigeprogramm.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.3 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses und Sed

6.26.1. Installation von Less

Bereiten Sie Less zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin --sysconfdir=/etc
```

Die Bedeutung der configure-Parameter:

```
--sysconfdir=/etc
```

Dieser Parameter bewirkt, dass die in diesem Paket installierten Programme ihre Konfigurationsdateien in `/etc` suchen.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.26.2. Inhalt von Less

Installierte Programme: less, lessecho und lesskey

Kurze Beschreibungen

- | | |
|-----------------|---|
| less | Ein Dateibetrachter. Er zeigt den Inhalt einer Datei an und ermöglicht, darin zu blättern, nach Zeichenketten zu suchen und zu Markierungen springen. |
| lessecho | Wird zum Expandieren von Metazeichen in Unix-Dateinamen benötigt (z. B. * und ?). |
| lesskey | Wird zum Festlegen der Tastenbelegung für less verwendet. |

6.27. Groff-1.19.1

Groff enthält verschiedene Programme zur Verarbeitung und Formatierung von Text.

Geschätzte Kompilierzeit: 0.5 SBU

Ungefähr benötigter Festplattenplatz: 38.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

6.27.1. Installation von Groff

Groff erwartet, dass die Umgebungsvariable `PAGE` die Standardpapiergröße enthält. Für Anwender in den Vereinigten Staaten ist `PAGE=letter` korrekt. Wenn Ihr Standort woanders ist, ersetzen Sie bitte `PAGE=letter` durch `PAGE=A4`.

Bereiten Sie Groff zum Kompilieren vor:

```
PAGE=[papier_größe] ./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Einige Dokumentationsprogramme wie zum Beispiel **xman** funktionieren ohne diese symbolischen Links nicht:

```
ln -sv soelim /usr/bin/zsoelim
ln -sv eqn /usr/bin/geqn
ln -sv tbl /usr/bin/gtbl
```

6.27.2. Inhalt von Groff

Installierte Programme: addftinfo, afmtodit, eqn, eqn2graph, geqn (Link auf eqn), grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (Link auf tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pfbtops, pic, pic2graph, post-grohtml, pre-grohtml, refer, soelim, tbl, tfmtodit, troff und zsoelim (Link auf soelim)

Kurze Beschreibungen

addftinfo	Liest eine troff-Schriftdatei und fügt einige schriftmetrische Informationen hinzu, die vom groff -System benutzt werden.
afmtodit	Erzeugt eine Schrift-Datei zur Verwendung mit groff und grops .
eqn	Kompiliert in troff-Eingabedateien enthaltene Beschreibungen von Gleichungen zu Kommandos, die troff versteht.

eqn2graph	Konvertiert eine EQN-Gleichung zu einem beschnittenen Bild.
geqn	Ein Link auf gawk .
grn	Ein groff -Präprozessor für gremlin-Dateien.
grodvi	Ein Treiber für groff , der das TeX dvi-Format erzeugt.
groff	Eine Benutzerschnittstelle für das groff-Dokumentenformatierungssystem. Normalerweise führt es das Programm troff und einen für das Ausgabegerät passenden Postprozessor aus.
groffer	Zeigt groff-Dateien und Man-pages unter X und im tty an.
grog	Liest Dateien ein und schätzt, welche der groff -Optionen <i>-e</i> , <i>-man</i> , <i>-me</i> , <i>-mm</i> , <i>-ms</i> , <i>-p</i> , <i>-s</i> und <i>-t</i> zum Drucken benötigt werden, und gibt das nötige groff -Kommando aus.
grolbp	Ein groff -Treiber für Canon CAPSL-Drucker (Laserdrucker der Serie LBP-4 und LBP-8).
grolj4	Ein Treiber für groff , der Ausgaben im PCL5-Format, passend für HP LaserJet 4-Drucker erzeugt.
grops	Übersetzt die Ausgabe von GNU troff zu PostScript.
grotty	Übersetzt die Ausgabe von GNU troff in eine passende Form für schreibmaschinenähnliche Geräte.
gtbl	Ein Link auf tbl .
hptodit	Erzeugt aus einer HP-markierten Schriftmetrik-Datei eine Schriftdatei zur Verwendung mit groff -Tlj4 .
indxbib	Erzeugt mit einer angegebenen Datei einen invertierten Index für die bibliographischen Datenbanken zur Verwendung mit refer , lookbib und lkbib .
lkbib	Durchsucht bibliographische Datenbanken nach Referenzen, die bestimmte Schlüssel enthalten, und gibt die gefundenen Referenzen aus.
lookbib	Gibt einen Prompt auf die standard-Fehlerausgabe (solange die Standardeingabe kein Terminal ist), liest eine Zeile mit Stichwörtern von der Standardeingabe, durchsucht eine bibliographische Datenbank nach Referenzen zu diesen Stichwörtern, gibt die gefundenen Referenzen aus und wiederholt das so lange bis keine weitere Eingabe mehr vorhanden ist.
mmroff	Ein einfacher Präprozessor für groff .
neqn	Formatiert Gleichungen für die ASCII-Ausgabe (American Standard Code for Information Interchange).
nroff	Ein Skript, das nroff -Kommandos mit groff emuliert.
pfbtops	Übersetzt eine Postscript-Schrift im <code>.pfb</code> -Format zu ASCII.
pic	Kompiliert in groff- oder TeX-Eingabedateien enthaltene Beschreibungen von Bildern zu Kommandos, die von TeX oder troff verwendet werden können.
pic2graph	Konvertiert ein PIC-Diagramm zu einem beschnittenen Bild.
post-grohtml	Übersetzt die Ausgabe von GNU troff zu HTML.

pre-grohtml	Übersetzt die Ausgabe von GNU troff zu HTML.
refer	Kopiert den Inhalt einer Datei zur Standardausgabe, außer das Zeilen zwischen <i>./</i> und <i>./</i> als Zitat interpretiert werden und Zeilen zwischen <i>.R1</i> und <i>.R2</i> als Kommandos behandelt werden, die angeben, wie mit Zitaten umgegangen werden soll.
soelim	Liest Dateien und ersetzt Zeilen der Form <i>.so <Datei> ></i> mit dem tatsächlichen Inhalt von <i><Datei></i> .
tbl	Kompiliert in troff-Eingabedateien eingebettete Beschreibungen von Tabellen zu Kommandos, die von troff unterstützt werden.
tfmtoedit	Erzeugt Schriftdateien zur Verwendung mit groff -Tdvi .
troff	Ist hochkompatibel mit Unix troff . Üblicherweise wird es mit dem Kommando groff aufgerufen, welches auch Präprozessoren und Postprozessoren in der richtigen Reihenfolge und mit den richtigen Optionen aufruft.
zsoelim	Ein Link auf soelim .

6.28. Sed-4.1.4

Das Paket Sed enthält einen Stream-Editor.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 8.4 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Texinfo

6.28.1. Installation von Sed

In der Voreinstellung wird die HTML-Dokumentation von Sed in `/usr/share/doc` installiert. Ändern Sie diesen Pfad zu `/usr/share/doc/sed-4.1.4`. Das folgende `sed`-Kommando ist Ihnen dabei behilflich:

```
sed -i 's@/doc@&/sed-4.1.4@' doc/Makefile.in
```

Bereiten Sie Sed zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie `make check` aus.

Installieren Sie das Paket:

```
make install
```

6.28.2. Inhalt von Sed

Installiertes Programm: sed

Kurze Beschreibungen

sed Wird zum Filtern und Transformieren von Dateien in einem einzigen Durchlauf verwendet.

6.29. Flex-2.5.31

Mit Flex kann man Programme zum Erkennen von Textmustern erzeugen.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 22.5 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make und Sed

6.29.1. Installation von Flex

Flex enthält einige bekannte Fehler. Beheben Sie diese mit dem folgenden Patch:

```
patch -Np1 -i ../flex-2.5.31-debian_fixes-3.patch
```

Die GNU autotools erkennen, dass der Quellcode von Flex durch den vorhergehenden Patch verändert wurde und versuchen, die Man-page entsprechend anzupassen. Das funktioniert aber auf vielen Systemen nicht korrekt und die Standard-Man-Page ist völlig in Ordnung, daher stellen Sie sicher, dass sie nicht neu erzeugt wird:

```
touch doc/flex.1
```

Bereiten Sie Flex zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Einige Programme erwarten die `lex`-Bibliothek in `/usr/lib`. Erstellen Sie daher einen entsprechenden symbolischen Link:

```
ln -sv libfl.a /usr/lib/libl.a
```

Einige wenige Programme kennen **flex** noch nicht und versuchen den Vorgänger **lex** aufzurufen. Um diesen Programmen dennoch gerecht zu werden, erzeugen Sie ein kleines Shell-Skript mit dem Namen `lex`, welches `flex` im `lex`-Emulationsmodus aufruft:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex

exec /usr/bin/flex -l "$@"

# End /usr/bin/lex
```

```
EOF  
chmod -v 755 /usr/bin/lex
```

6.29.2. Inhalt von Flex

Installierte Programme: flex und lex

Installierte Bibliothek: libfl.a

Kurze Beschreibungen

flex Ein Werkzeug zum Erzeugen von Programmen, die Muster in Text erkennen können. Mustererkennung ist in vielen Programmen nützlich. Flex erzeugt aus einem Satz an Suchregeln ein Programm, das nach diesen Mustern sucht.

lex Ein Skript, welches **flex** im **lex**-Emulationsmodus startet.

libfl.a Die flex-Bibliothek.

6.30. Gettext-0.14.3

Gettext wird zur Übersetzung und Lokalisierung verwendet. Programme können mit Unterstützung für NLS (Native Language Support, Unterstützung für die lokale Sprache) kompiliert werden. Dadurch können Texte und Meldungen in der Sprache des Anwenders ausgegeben werden.

Geschätzte Kompilierzeit: 1.2 SBU

Ungefähr benötigter Festplattenplatz: 65.1 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

6.30.1. Installation von Gettext

Bereiten Sie Gettext zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Zum Durchlaufen der Testsuite können Sie dieses Kommando benutzen: **make check**. Leider benötigt diese Testsuite viel Zeit: etwa 7 SBU.

Installieren Sie das Paket:

```
make install
```

6.30.2. Inhalt von Gettext

Installierte Programme: autopoint, config.charset, config.rpath, envsubst, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, uand xgettext

Installierte Bibliotheken: libasprintf.[a,so], libgettextlib.so, libgettextpo.[a,so] und libgettextsrc.so

Kurze Beschreibungen

autopoint	Kopiert die Dateien einer typischen Gettext-Infrastruktur in ein Quellpaket.
config.charset	Gibt eine systemabhängige Tabelle von zeichenkodierenden Aliasen aus.
config.rpath	Gibt einen systemabhängigen Satz von Variablen aus, die beschreiben, wie der Laufzeit-Suchpfad von gemeinsamen Bibliotheken in einer ausführbaren Datei gesetzt wird.
envsubst	Erweitert Umgebungsvariablen in Shell-Format-Zeichenketten.
gettext	Übersetzt Nachrichten in natürlicher Sprache in die Muttersprache des Anwenders. Dafür benutzt es einen Übersetzungsnachrichten-Katalog.

gettextize	Kopiert alle standard-Gettext-Dateien in den Basisordner eines Pakets um so die ersten Schritte der Internationalisierung zu erleichtern.
hostname	Zeigt den Netzwerk-Hostnamen in verschiedenen Formen an.

msgattrib	Filtert Nachrichten in einem Übersetzungskatalog nach ihren Attributen und manipuliert diese Attribute.
msgcat	Fügt die angegebenen .po-Dateien aneinander und verschmelzt sie.
msgcmp	Vergleicht zwei .po-Dateien, um sicherzustellen, dass beide den gleichen Satz an msgid-Zeichenketten enthalten.
msgcomm	Findet die Nachrichten, die die angegebenen .po-Dateien gemeinsam haben.
msgconv	Konvertiert den Übersetzungskatalog in einen anderen Zeichensatz.
msgen	Erzeugt einen englischen Übersetzungskatalog.
msgexec	Führt ein Kommando auf allen Übersetzungen in einem Katalog aus.
msgfilter	Wendet einen Filter auf alle Übersetzungen in einem Katalog an.
msgfmt	Erzeugt aus einem Übersetzungskatalog einen binären Katalog.
msggrep	Extrahiert alle Nachrichten aus einem Katalog, die auf ein bestimmtes Muster passen oder zu einer bestimmten Quelldatei gehören.
msginit	Erzeugt eine neue .po-Datei und initialisiert die Meta-Informationen mit Werten aus der Arbeitsumgebung des Benutzers.
msgmerge	Kombiniert zwei Übersetzungen in eine einzige Datei.
msgunfmt	Erzeugt aus einem binären Katalog einen Nachrichtenkatalog in Textform.
msguniq	Vereinheitlicht doppelte Übersetzungen in einem Nachrichtenkatalog.
ngettext	Zeigt die Übersetzung einer Textnachricht an, deren Grammatik von einer Zahl abhängt.
xgettext	Extrahiert alle übersetzbaren Nachrichten aus den angegebenen Quelldateien, um daraus eine erste Nachrichtenkatalogvorlage zu erstellen.
<code>libasprintf</code>	Definiert die <i>autosprintf</i> -Klasse; sie macht C-formatierte Routinen in C++ Programmen verfügbar, vor allem zur Verwendung mit <code><string></code> Strings und den <code><iostream></code> Streams.
<code>libgettextlib</code>	Eine private Bibliothek, die die allgemeinen Routinen der verschiedenen gettext-Programme enthält. Sie sind nicht zur normalen Verwendung gedacht.
<code>libgettextpo</code>	Wird zum Schreiben von spezialisierten Programmen verwendet, die .po-Dateien verarbeiten sollen. Diese Bibliothek wird benutzt, wenn die mitgelieferten Standardprogramme von gettext nicht ausreichen (so wie msgattrib und msgen).
<code>libgettextsrc</code>	Eine private Bibliothek, die die allgemeinen Routinen der verschiedenen gettext-Programme enthält. Sie sind nicht zur normalen Verwendung gedacht.

6.31. Inetutils-1.4.2

Inetutils enthält verschiedene Programme zur grundlegenden Netzwerkunterstützung.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 8.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses und Sed

6.31.1. Installation von Inetutils

Inetutils hat Probleme mit der 2.6er Kernelserie. Beheben Sie diese Probleme mit dem folgenden Patch:

```
patch -Np1 -i ../inetutils-1.4.2-kernel_headers-1.patch
```

Sie werden nicht alle Programme aus diesem Paket installieren. Dennoch würde Inetutils die Man-pages zu diesen Programmen installieren. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../inetutils-1.4.2-no_server_man_pages-1.patch
```

Bereiten Sie Inetutils zum Kompilieren vor:

```
./configure --prefix=/usr --libexecdir=/usr/sbin \
  --sysconfdir=/etc --localstatedir=/var \
  --disable-logger --disable-syslogd \
  --disable-whois --disable-servers
```

Die Bedeutung der configure-Parameter:

--disable-logger

Das verhindert die Installation des Programmes **logger**, welches Nachrichten an den System-Log-Daemon übergibt. Logger wird hier ausgelassen, weil etwas später durch Util-Linux eine bessere Version installiert wird.

--disable-syslogd

Dieser Parameter verhindert die Installation des System-Log-Daemon, weil Sie später einen anderen mit dem Paket Syslogd installieren werden.

--disable-whois

Dies verhindert die Installation des **whois**-Clients, welcher leider elendig veraltet ist. Im BLFS-Buch finden Sie eine Installations-Anleitung für einen besseren **whois**-Client.

--disable-servers

Das verhindert die Installation verschiedener Server-Dienste die zu Inetutils gehören. Diese Dienste sind in einem Basis-System wie LFS nicht angebracht. Einige sind von Natur aus unsicher und nur in vertrauenswürdigen Netzen ohne Risiko einsetzbar. Mehr Informationen finden Sie unter <http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html>. Bitte beachten Sie auch, dass es für fast alle dieser Dienste einen besseren Ersatz gibt.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Und verschieben Sie das Programm **ping** an die richtige Stelle:

```
mv -v /usr/bin/ping /bin
```

6.31.2. Inhalt von Inetutils

Installierte Programme: ftp, ping, rcp, rlogin, rsh, talk, telnet und tftp

Kurze Beschreibungen

ftp	Das Programm für FTP (File Transfer Protocol).
ping	Sendet echo-request-Pakete und berichtet, wie lange die Antwort braucht.
rcp	Kopiert Dateien auf entfernten Systemen.
rlogin	Führt eine entfernte Anmeldung durch.
rsh	Führt eine entfernte Shell aus.
talk	Wird zum Unterhalten mit anderen Benutzern verwendet.
telnet	Dies ist ein Telnet-Client.
tftp	Das Programm zu TFTP (Trivial File Transfer Protocol).

6.32. IPRoute2-2.6.11-050330

Das Paket IPRoute2 enthält verschiedene Programme zur grundlegenden Unterstützung von IPv4-basierten Netzwerken.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 4.3 MB

Die Installation ist abhängig von: GCC, Glibc, Make, Linux-Headers und Sed

6.32.1. Installation von IPRoute2

Das Programm **arpd** aus diesem Paket ist von Berkeley DB abhängig. Weil **arpd** in einem Basis-System eigentlich nicht benötigt wird, entfernen Sie die Abhängigkeit zu Berkeley DB mit dem folgenden **sed**-Kommando. Falls Sie **arpd** benötigen, finden Sie eine Anleitung zur Installation von Berkeley DB im BLFS-Buch unter <http://www.linuxfromscratch.org/blfs/view/svn/server/databases.html#db>.

```
sed -i '/^TARGETS/s@arpd@g' misc/Makefile
```

Bereiten Sie IPRoute2 zum Kompilieren vor:

```
./configure
```

Kompilieren Sie das Paket:

```
make SBINDIR=/sbin
```

Die Bedeutung der make-Option:

```
SBINDIR=/sbin
```

Dies stellt sicher, dass die Binärdateien von IPRoute2 nach `/sbin` installiert werden. Lt. FHS ist dies der korrekte Ort, weil einige der Programme aus IPRoute2 in Bootskripten verwendung finden.

Installieren Sie das Paket:

```
make SBINDIR=/sbin install
```

6.32.2. Inhalt von IPRoute2

Installierte Programme: `ctstat` (Link auf `lnstat`), `ifcfg`, `ifstat`, `ip`, `lnstat`, `nstat`, `routef`, `routel`, `rtacct`, `rtmon`, `rtrpr`, `rtstat` (Link auf `lnstat`), `ss` und `tc`.

Kurze Beschreibungen

ctstat	Ein Werkzeug für den Verbindungsstatus.
ifcfg	Ein Shell-Skript Wrapper für ip .
ifstat	Zeigt Schnittstellenstatistiken an, inklusive der Menge der gesendeten und empfangenen Pakete pro Schnittstelle.

ip	<p>Dies ist die eigentliche ausführbare Datei. Sie hat viele Funktionen:</p> <p>ip link [Gerät] zeigt den Gerätestatus an und ermöglicht Änderungen an den Einstellungen.</p> <p>ip addr zeigt Adressen und ihre Eigenschaften an, fügt neue Adressen hinzu und löscht alte.</p> <p>ip neighbor zeigt Bindungen und Eigenschaften von benachbarten Geräten an, fügt neue Nachbargerätebindungen hinzu und löscht alte.</p> <p>ip rule zeigt Routingregeln an und bearbeitet sie.</p> <p>ip route ermöglicht das Anzeigen und Ändern von Routingtabellen.</p> <p>ip tunnel zeigt IP-Tunnel und die Eigenschaften an und ermöglicht Änderungen daran.</p> <p>ip maddr zeigt Multicast-Adressen und ihre Eigenschaften an und ermöglicht Änderungen.</p> <p>ip mroute setzt, ändert oder löscht Multicast-Routen.</p> <p>ip monitor ermöglicht, dauerhaft den Status von Netzwerkgeräten, Adressen und Routen zu überwachen.</p>
lnstat	Bietet Netzwerkstatistiken unter Linux. Dies ist ein allgemeinerer und vollständigerer Ersatz für das alte Programm rtstat .
nstat	Zeigt Netzwerkstatistiken an.
routef	Eine Komponente von ip route . Sie wird zum Leeren der Routingtabellen genutzt.
routel	Eine Komponente von ip route . Sie wird zum Auflisten der Routingtabellen genutzt.
rtacct	Zeigt den Inhalt von <code>/proc/net/rt_acct</code> an.
rtmon	Ein Werkzeug zum Überwachen des Routing.
rtpr	Konvertiert die Ausgabe von ip -o zurück in eine lesbare Form.
rtstat	Ein Werkzeug für den Routingstatus.
ss	Ähnlich wie das Kommando netstat . Zeigt aktive Verbindungen an.
tc	<p>Programm zur Kontrolle des Netzwerkverkehrs (Traffic Controlling). Implementiert Quality of Service (QOS) und Class Of Service (COS):</p> <p>tc qdisc ermöglicht das Einstellen der Warteschlangen-Regeln.</p> <p>tc class ermöglicht das Einrichten von Klassen, basierend auf einer Warteschlangen-Regelung.</p> <p>tc estimator ermöglicht das Schätzen des Netzwerk-Flusses in ein Netzwerk.</p>

tc filter ermöglicht das Erstellen von QOS/COS Paketfiltern.

tc policy ermöglicht das Erstellen von QOS/COS Regelwerken.

6.33. Perl-5.8.7

Das Paket Perl enthält die Skriptsprache Perl (Practical Extraction and Report Language).

Geschätzte Kompilierzeit: 4.1 SBU

Ungefähr benötigter Festplattenplatz: 140 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

6.33.1. Installation von Perl

Wenn Sie die vollständige Kontrolle darüber haben möchten, wie Perl sich selbst zum Installieren einrichtet, dann können Sie stattdessen das interaktive **Configure**-Skript benutzen. Wenn Sie mit den (normalerweise sinnvollen) von Perl automatisch erkannten Voreinstellungen zufrieden sind, benutzen Sie einfach das folgende Kommando:

```
./configure.gnu --prefix=/usr -Dpager="/bin/less -isR"
```

Die Bedeutung der configure-Parameter:

```
-Dpager="/bin/less -isR"
```

Dies korrigiert einen Fehler in der Art und Weise, wie **perldoc**, das Programm **less** aufruft.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite ausführen möchten, müssen Sie erst eine Basisversion der Datei `/etc/hosts` erstellen. Sie wird von einigen Tests zum Auflösen des Hostnamens „localhost“ benötigt:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Wenn Sie möchten, können Sie nun die Testsuite starten:

```
make test
```

Installieren Sie das Paket:

```
make install
```

6.33.2. Inhalt von Perl

Installierte Programme: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.7 (Link auf perl), perlbug, perlcc, perldoc, perlvp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (Link auf s2p), pstruct (Link auf c2ph), s2p, splain und xsubpp

Installierte Bibliotheken: Mehrere hundert, die hier nicht alle aufgelistet werden können

Kurze Beschreibungen

a2p	Übersetzt <code>awk</code> zu Perl.
c2ph	Gibt C-Strukturen aus, die von <code>cc -g -S</code> erzeugt wurden.
dprofpp	Zeigt Perl-Profilng-Daten an.
en2cxs	Erzeugt aus Unicode-Zeichenzuordnungen oder Tcl-Encoding-Dateien eine Perl-Erweiterung für das Encode-Modul.
find2perl	Übersetzt find -Kommandos zu Perl.
h2ph	Konvertiert <code>.h</code> C Header-Dateien zu <code>.ph</code> Perl Header-Dateien.
h2xs	Konvertiert <code>.h</code> C Header-Dateien zu Perl-Erweiterungen.
libnetcfg	Kann zum Einrichten von <code>libnet</code> benutzt werden.
perl	Kombiniert viele der besten Eigenschaften von C, sed , awk und sh in einer einzigen universell einsetzbaren Sprache. Perl wird auch als das Schweizer Taschenmesser für Programmier bezeichnet.
perl5.8.7	Ein harter Link auf perl .
perlbug	Wird zum Erzeugen und Emailen von Fehlerberichten zu Perl oder seinen Modulen verwendet.
perlcc	Erzeugt ausführbare Dateien aus Perl-Programmen.
perldoc	Zeigt Teile einer Dokumentation im pod-Format an.
perlivp	Die Perl Installations-Prüfprozedur. Damit wird geprüft, ob Perl und seine Bibliotheken korrekt installiert wurden.
piconv	Die Perl-Version des Zeichensatz-Konverters iconv .
pl2pm	Ein Werkzeug zum groben Umwandeln von Perl4 <code>.pl</code> -Dateien in Perl5 <code>.pm</code> -Module.
pod2html	Konvertiert pod-Dateien in das HTML-Format.
pod2latex	Konvertiert pod-Dateien zu LaTeX.
pod2man	Konvertiert pod-Daten zu formatierter <code>*roff</code> -Eingabe.
pod2text	Konvertiert pod-Daten in formatierten ASCII-Text.
pod2usage	Gibt Benutzungshinweise aus eingebetteten pod-Dokumenten in Dateien aus.
podchecker	Prüft die Syntax von pod-Dokumentationsdateien.
podselect	Zeigt ausgewählte Abschnitte einer pod-Dokumentation an.
psed	Die Perl-Version des Stream-Editors sed .
pstruct	Gibt C-Strukturen aus, die von <code>cc -g -S</code> erzeugt wurden.
s2p	Konvertiert sed -Skripte zu perl.
splain	Erzwingt die ausführliche Analyse von Warnungen in Perl.
xsubpp	Konvertiert Perl XS-Code zu C-Code.

6.34. Texinfo-4.8

Das Paket Texinfo enthält Programme zum Lesen, Schreiben und Konvertieren von Info-Seiten (Systemdokumentation).

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 14.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses und Sed

6.34.1. Installation von Texinfo

Texinfo ermöglicht es lokalen Benutzern durch eine sog. Symlink-Attacke bestimmte Dateien zu überschreiben. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../texinfo-4.8-tempfile_fix-1.patch
```

Bereiten Sie Texinfo zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Optional können Sie auch die zu einer typischen TeX-Installation gehörenden Pakete installieren:

```
make TEXMF=/usr/share/texmf install-tex
```

Die Bedeutung des make-Parameters:

```
TEXMF=/usr/share/texmf
```

Die Makefile-Variante TEXMF enthält den Pfad zu Ihrem TeX Basisordner, falls später TeX installiert wird.

Das Info-Dokumentationssystem speichert die Liste der Menüeinträge in einer einfachen Textdatei. Die Datei liegt in `/usr/share/info/dir`. Unglücklicherweise können die Einträge in dieser Datei durch Probleme mit Makefile-Dateien einzelner Pakete durcheinander geraten. Falls Sie diese Datei jemals neu erzeugen müssen, ist Ihnen das folgende Kommando dabei behilflich:

```
cd /usr/share/info
rm dir
for f in *
do install-info $f dir 2>/dev/null
done
```

6.34.2. Inhalt von Texinfo

Installierte Programme: info, infokey, install-info, makeinfo, texi2dvi, texi2pdf und texindex

Kurze Beschreibungen

info	Wird zum Lesen von Info-Dokumenten benutzt. Info-Dokumente sind Man-pages sehr ähnlich, gehen aber oft tiefer in die Materie als einfach nur die möglichen Parameter zu beschreiben. Vergleichen Sie beispielsweise man bison und info bison .
infokey	Kompiliert eine Quelldatei mit Info-Anpassungen in ein binäres Format.
install-info	Wird zum Installieren von Info-Dateien benutzt. Es aktualisiert die Einträge in der info -Indexdatei.
makeinfo	Übersetzt Texinfo Quelldokumente in verschiedene andere Formate: Info-Dateien, reiner Text, oder HTML.
texi2dvi	Wird zum Formatieren von Texinfo-Dokumenten in ein Geräteunabhängiges Format zum Drucken benutzt.
texi2pdf	Wird zum Konvertieren von Texinfo-Dokumenten in das portable Document Format (PDF) verwendet.
texindex	Sortiert Texinfo-Indexdateien.

6.35. Autoconf-2.59

Autoconf erstellt Shell-Skripte, mit denen man Software-Pakete automatisch zum Kompilieren einrichten kann.

Geschätzte Kompilierzeit: 0.5 SBU

Ungefähr benötigter Festplattenplatz: 8.5 MB

Die Installation ist abhängig von: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl und Sed

6.35.1. Installation von Autoconf

Bereiten Sie Autoconf zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse können Sie **make check** benutzen. Dies dauert lange; etwa 2 SBU.

Installieren Sie das Paket:

```
make install
```

6.35.2. Inhalt von Autoconf

Installierte Programme: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate und ifnames

Kurze Beschreibungen

autoconf	Ein Werkzeug zum Erzeugen von Shell-Skripten, die Quellcode-Pakete automatisch einrichten und sie an unterschiedliche Unix-System anpassen. Die resultierenden configure-Skripte sind eigenständig—sie können auch dann ausgeführt werden, wenn autoconf nicht installiert ist.
autoheader	Ein Werkzeug zum Erzeugen von Vorlagedateien für C <i>#define</i> -Anweisungen, die configure benutzen soll.
autom4te	Ein Wrapper zu dem Makroprozessor M4.
autoreconf	Führt automatisch autoconf , autoheader , aclocal , automake , gettextize und libtoolize in der richtigen Reihenfolge aus. Das spart Zeit, wenn Änderungen an autoconf und automake Vorlagedateien gemacht wurden.
autoscan	Kann beim Erzeugen einer <code>configure.in</code> -Datei für ein Softwarepaket behilflich sein. Es untersucht die Quelldateien in einem Ordner und sucht nach üblichen Portabilitätsproblemen und erzeugt eine <code>configure.scan</code> -Datei, die als Basis für eine <code>configure.in</code> -Datei zu dem Softwarepaket dienen kann.
autoupdate	Verändert eine <code>configure.in</code> -Datei so, dass sie nicht mehr die alten Namen der autoconf

Makros aufruft, sondern die neuen.

ifnames

Kann beim Schreiben einer `configure.in`-Datei für ein Paket hilfreich sein. Es gibt die Bezeichner aus, die ein Paket in Präprozessor-Konditionen benutzt. Wenn ein Paket bereits für Portabilität eingerichtet ist, kann dieses kleine Werkzeug zum Auffinden der nötigen **configure**-Tests hilfreich sein. Es kann einige Lücken in autoscan-generierten `configure.in`-Dateien füllen.

6.36. Automake-1.9.5

Automake generiert Makefile-Dateien zur Verwendung mit Autoconf.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 8.8 MB

Die Installation ist abhängig von: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl und Sed

6.36.1. Installation von Automake

Bereiten Sie Automake zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse können Sie **make check** benutzen. Dies dauert recht lange; etwa 5 SBUs.

Installieren Sie das Paket:

```
make install
```

6.36.2. Inhalt von Automake

Installierte Programme: acinstall, aclocal, aclocal-1.9.5, automake, automake-1.9.5, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree und ylwrap

Kurze Beschreibungen

acinstall	Ein Skript, das M4-Dateien im aclocal-Stil installiert.
aclocal	Erzeugt auf dem Inhalt von <code>configure.in</code> -Dateien basierend, entsprechende <code>aclocal.m4</code> -Dateien.
aclocal-1.9.5	Ein harter Link auf aclocal .
automake	Ein Werkzeug zum automatischen Erzeugen von <code>Makefile.in</code> 's aus sog. <code>Makefile.am</code> -Dateien. Um alle <code>Makefile.in</code> -Dateien eines Pakets zu erzeugen, lassen Sie dieses Programm im Basisordner des Pakets laufen. Durch das Scannen von <code>configure.in</code> findet es automatisch jede nötige <code>Makefile.am</code> -Datei und erzeugt die entsprechende <code>Makefile.in</code> -Datei.
automake-1.9.5	Ein harter Link auf automake .
compile	Ein Wrapper für verschiedene Compiler.
config.guess	Ein Skript. Es versucht, kanonische Tripplets für das Build, den Host oder die Zielarchitektur zu erraten.

config.sub

Ein Unter-Skript zum Validieren der Konfiguration.

depcomp	Ein Skript zum Kompilieren eines Programmes, so dass nicht nur das gewünschte Ergebnis erzeugt wird, sondern auch Abhängigkeitsinformationen generiert werden.
elisp-comp	Byte-kompiliert Emacs Lisp-Code.
install-sh	Ein Skript, welches ein Programm, ein Skript oder eine Datendatei installiert.
mdate-sh	Ein Skript, welches den Änderungszeitstempel einer Datei oder eines Ordners ausgibt.
missing	Ein Skript, welches fehlende GNU-Programme während der Installation ersetzt.
minstalldirs	Ein Skript zum Erzeugen einer Ordnerstruktur.
py-compile	Kompiliert ein Python-Programm.
symlink-tree	Ein Skript zum Erzeugen einer Symlink-Version einer Ordnerstruktur.
ylwrap	Ein Wrapper für lex und yacc .

6.37. Bash-3.0

Das Paket Bash enthält die Bourne-Again-SHell.

Geschätzte Kompilierzeit: 1.2 SBU

Ungefähr benötigter Festplattenplatz: 20.6 MB

Die Installation ist abhängig von: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses und Sed.

6.37.1. Installation von Bash

Wenn Sie die Bash-Dokumentation heruntergeladen haben und die HTML-Dokumentation installieren möchten, dann führen Sie bitte die folgenden Kommandos aus:

```
tar -xvf ../bash-doc-3.0.tar.gz &&
sed -i "s|html|dir = @html|dir = /usr/share/doc/bash-3.0|" \
    Makefile.in
```

Der folgende Patch behebt mehrere Probleme, unter anderem eines, bei dem Readline manchmal nur 33 Zeichen einer Zeile anzeigt und dann zur nächsten Zeile springt:

```
patch -Np1 -i ../bash-3.0-fixes-3.patch
```

Es gibt auch ein Problem, wenn Bash mit recht neuen Versionen von Glibc kompiliert wird. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../bash-3.0-avoid_WCONTINUED-1.patch
```

Bereiten Sie Bash zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin \
    --without-bash-malloc --with-installed-readline
```

Die Bedeutung der configure-Parameter:

--with-installed-readline

Dieser Parameter lässt Bash die von uns installierte `readline`-Bibliothek anstelle der Bash-eigenen Version benutzen.

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **make tests**.

Installieren Sie das Paket:

```
make install
```

Starten Sie die frisch installierte **bash** (ersetzt die gerade laufende Version):

```
exec /bin/bash --login +h
```



Anmerkung

Die verwendeten Parameter machen **bash** zu einer interaktiven Login-Shell. Hashing bleibt weiterhin abgeschaltet, so dass frisch installierte Programme sofort verfügbar sind.

6.37.2. Inhalt von Bash

Installierte Programme: bash, bashbug und sh (Link auf bash)

Kurze Beschreibungen

- bash** Ein weit verbreiteter Befehlsinterpreter. Er führt alle möglichen Arten von Erweiterungen und Ersetzungen an einer Kommandozeile durch, bevor diese dann ausgeführt wird. Das macht diesen Befehlsinterpreter zu einem mächtigen Werkzeug.
- bashbug** Ein Shell-Skript, welches dem Benutzer helfen soll, einen Fehlerbericht zur **bash** in einem standardisierten Format zu erstellen und per E-Mail zu versenden.
- sh** Ein symbolischer Link auf das Programm **bash**. Wenn die **bash** als **sh** aufgerufen wird, versucht sie, das Verhalten der historischen Versionen von **sh** so gut wie möglich nachzuahmen und bleibt dabei trotzdem POSIX-Konform.

6.38. File-4.13

File ist ein kleines Werkzeug mit dem man den Dateityp einer oder mehrerer Dateien feststellen kann.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 6.2 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed und Zlib

6.38.1. Installation von File

Bereiten Sie File zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.38.2. Inhalt von File

Installierte Programme: file

Installierte Bibliothek: libmagic.[a,so]

Kurze Beschreibungen

- | | |
|-----------------|---|
| file | Versucht, Dateien zu klassifizieren. Dazu führt es verschiedene Tests durch—Dateisystem-Tests, Tests mit „magischen“ Nummern, und Sprachtests. Der erste erfolgreiche Test entscheidet über das Ergebnis. |
| libmagic | Enthält Routinen zur Erkennung von „magischen“ Nummern; wird vom Programm file verwendet. |

6.39. Libtool-1.5.14

Das Libtool-Skript enthält die Unterstützung für Bibliotheken. Libtool versteckt die Komplexität von gemeinsam benutzten Bibliotheken hinter einer konsistenten und portablen Schnittstelle.

Geschätzte Kompilierzeit: 1.5 SBU

Ungefähr benötigter Festplattenplatz: 19.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

6.39.1. Installation von Libtool

Bereiten Sie Libtool zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.39.2. Inhalt von Libtool

Installierte Programme: libtool und libtoolize

Installierte Bibliotheken: libltdl.[a,so]

Kurze Beschreibungen

libtool	Stellt vereinheitlichte Dienste zum Erstellen von Bibliotheken zur Verfügung.
libtoolize	Stellt einen einheitlichen Weg zur Verfügung um einem Paket libtool -Unterstützung hinzuzufügen.
libltdl	Versteckt die verschiedenen Schwierigkeiten mit Bibliotheken die dlopen verwenden.

6.40. Bzip2-1.0.3

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. **Bzip2** erreicht vor allem bei Textdateien eine wesentlich bessere Kompressionsrate als das traditionelle **gzip**.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 3.9 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc und Make

6.40.1. Installation von Bzip2

Wenden Sie einen Patch an, um auch die Dokumentation zu diesem Paket zu installieren:

```
patch -Np1 -i ../bzip2-1.0.3-install_docs-1.patch
```

Das Programm **bzgrep** filtert die Buchstaben '|' und '&' in übergebenen Dateinamen nicht aus. Dadurch ist es möglich, Kommandos mit den Benutzerrechten des Benutzers von **bzgrep** auszuführen. Wenden Sie zur Korrektur diesen Patch an:

```
patch -Np1 -i ../bzip2-1.0.3-bzgrep_security-1.patch
```

Bereiten Sie Bzip2 zum Kompilieren vor:

```
make -f Makefile-libbz2_so
make clean
```

Der Parameter `-f` veranlasst Bzip2, ein alternatives Makefile (in diesem Fall `Makefile-libbz2_so`) zu verwenden. Dieses erzeugt eine dynamische Bibliothek `libbz2.so` und verlinkt die Bzip2-Werkzeuge damit.

Kompilieren und testen Sie das Paket:

```
make
```

Falls Sie Bzip2 neu installieren müssen, müssen Sie zuerst `rm -vf /usr/bin/bz*` ausführen, ansonsten schlägt `make install` fehl.

Installieren Sie die Programme:

```
make install
```

Installieren Sie die ausführbare Datei **bzip2** nach `/bin`. Dann erzeugen Sie ein paar nötige symbolische Links und räumen auf:

```
cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat
```

6.40.2. Inhalt von Bzip2

Installierte Programme: bunzip2 (Link auf bzip2), bzcata (Link auf bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless und bzmora

Installierte Bibliotheken: libbz2.[a,so]

Kurze Beschreibungen

bunzip2	Dekomprimiert bzip2-Dateien.
bzcata	Dekomprimiert zur Standardausgabe.
bzcmp	Führt cmp auf bzip2-Dateien aus.
bzdiff	Führt diff auf bzip2-Dateien aus.
bzgrep	Führt grep auf bzip2-Dateien aus.
bzegrep	Führt egrep auf bzip2-Dateien aus.
bzfgrep	Führt fgrep auf bzip2-Dateien aus.
bzip2	Komprimiert Dateien mit dem blocksortierenden Burrows-Wheeler Textkompressionsalgorithmus und Huffman-Kodierung. Die Kompressionsrate ist merkbar besser als die von herkömmlichen Kompressoren mit LZ77/LZ78, wie zum Beispiel gzip .
bzip2recover	Versucht, Daten aus beschädigten bzip2-Dateien zu reparieren.
bzless	Führt less auf bzip2-Dateien aus.
bzmora	Führt more auf bzip2-Dateien aus.
<code>libbz2*</code>	Die Bibliothek, die verlustlose blocksortierende Datenkompression mit Hilfe des Burrows-Wheeler-Algorithmus implementiert.

6.41. Diffutils-2.8.1

Die Programme dieses Pakets können Unterschiede zwischen Dateien oder Ordnern anzeigen.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 5.6 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

6.41.1. Installation von Diffutils

Bereiten Sie Diffutils zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.41.2. Inhalt von Diffutils

Installierte Programme: cmp, diff, diff3 und sdiff

Kurze Beschreibungen

- cmp** Vergleicht zwei Dateien und berichtet, ob, und an welchen Bytes sie sich unterscheiden.
- diff** Vergleicht zwei Dateien oder Ordner und berichtet, in welchen Zeilen sie sich unterscheiden.
- diff3** Vergleicht drei Dateien Zeile für Zeile.
- sdiff** Führt interaktiv zwei Dateien zusammen und gibt das Ergebnis aus.

6.42. Kbd-1.12

Kbd enthält die Dateien für das Tastaturlayout und entsprechende Werkzeuge dazu.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 11.8 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make und Sed

6.42.1. Installation von Kbd

Bereiten Sie Kbd zum Kompilieren vor:

```
./configure
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.42.2. Inhalt von Kbd

Installierte Programme: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, getunimap, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (Link auf psfxtable), psfgettable (Link auf psfxtable), psfstrietable (Link auf psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start und unicode_stop

Kurze Beschreibungen

chvt	Ändert das aktive Virtuelle Terminal.
deallocvt	Gibt unbenutzte Virtuelle Terminals wieder frei.
dumpkeys	Gibt Tastaturübersetzungstabellen aus.
fgconsole	Gibt die Nummer des aktiven Virtuellen Terminals aus.
getkeycodes	Gibt die Scancode-zu-Keycode Zuweisungstabelle des Kernels aus.
getunimap	Gibt die derzeitige Unicode-zu-Schrift Zuweisungstabelle des Kernels aus.
kbd_mode	Setzt den Tastaturmodus bzw. zeigt ihn an.
kbdrate	Setzt die Tastenwiederholrate und -pausen oder zeigt sie an.
loadkeys	Lädt Tastaturübersetzungstabellen.
loadunimap	Lädt eine Unicode-zu-Schrift Zuweisungstabelle des Kernels.

mapscrn	Ein veraltetes Programm, das benutzerdefinierte Zeichenausgabe-Zuweisungstabellen in den Konsolentreiber lädt. Dies wird heutzutage durch setfont erledigt.
openvt	Startet ein Programm in einem neuen Virtuellen Terminal (VT).
psfaddtable	Ein Link auf psfxtable .
psfgettable	Ein Link auf psfxtable .
psfstriptime	Ein Link auf psfxtable .
psfxtable	Ein Satz von Werkzeugen zum Umgang mit Unicode-Zeichentabellen für Konsole-Schriften.
resizecons	Ändert die Vorstellung des Kernels über die Ausmaße einer Konsole.
setfont	Ändert EGA- (Enhanced Graphic Adapter) und VGA- (Video Graphics Array) Schriften in der Konsole.
setkeycodes	Lädt Scancode-zu-Keycode Zuweisungstabellen des Kernel. Nützlich, wenn Sie ein paar unübliche Tasten auf Ihrer Tastatur haben.
setleds	Stellt Tastaturoptionen und die LED's ein.
setlogcons	Sendet Kernel-Nachrichten auf die Konsole.
setmetamode	Definiert die Behandlung von Meta-Tasten auf der Tastatur.
setvesablank	Lässt Sie den eingebauten Hardware-Bildschirmschoner anpassen (keine fliegenden Toaster, nur ein einfacher schwarzer Schirm).
showconsolefont	Zeigt die aktuelle EGA/VGA-Konsole-Schrift an.
showkey	Zeigt Scancode, Keycode und ASCII-Code der auf der Tastatur gedrückten Taste an.
unicode_start	Schaltet die Tastatur in den Unicode-Modus. Benutzen Sie dieses Kommando nicht auf einem LFS-System, weil die Anwendungen nicht für Unicode-Unterstützung eingerichtet sind.
unicode_stop	Schaltet den Unicode-Modus von Tastatur und Konsole wieder aus.

6.43. E2fsprogs-1.37

E2fsprogs stellt die Werkzeuge zur Verwendung mit dem ext2-Dateisystem zur Verfügung. Auch ext3 wird unterstützt (ein Journaling Dateisystem).

Geschätzte Kompilierzeit: 0.6 SBU

Ungefähr benötigter Festplattenplatz: 40.0 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed und Texinfo

6.43.1. Installation von E2fsprogs

Beheben Sie einen Kompilierfehler in der Testsuite von E2fsprogs:

```
sed -i -e 's/-DTEST/$(ALL_CFLAGS) &/' lib/e2p/Makefile.in
```

Es wird empfohlen, E2fsprogs außerhalb des Quellordners zu kompilieren:

```
mkdir -v build
cd build
```

Bereiten Sie E2fsprogs zum Kompilieren vor:

```
../configure --prefix=/usr --with-root-prefix="" \
  --enable-elf-shlibs --disable-evms
```

Die Bedeutung der configure-Parameter:

--with-root-prefix=""

Bestimmte Programme (wie z. B. **e2fsck**) sind absolut essentiell. Sie müssen z. B. selbst dann verfügbar sein, wenn `/usr` noch nicht eingehängt ist. Diese Programme gehören in Ordner wie `/lib` und `/sbin`. Ohne diese Option würden die Programme entgegen unserem Willen in `/usr` installiert werden.

--enable-elf-shlibs

Das erzeugt die gemeinsamen Bibliotheken, die einige Programme in diesem Paket verwenden.

--disable-evms

Dies deaktiviert die Installation des Enterprise Volume Management System (EVMS) Plugin. Das Plugin ist nicht auf dem Stand der aktuellen internen EVMS Schnittstellen und außerdem wird EVMS nicht als Teil des LFS Basis-Systems installiert; daher brauchen wir dieses Plugin nicht. Weitere Informationen erhalten Sie auf der Webseite von EMVS unter <http://evms.sourceforge.net/>.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie die Binärdateien und die Dokumentation:

```
make install
```

Installieren Sie die gemeinsamen Bibliotheken:

```
make install-libs
```

6.43.2. Inhalt von E2fsprogs

Installierte Programme: badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs und uuidgen.

Installierte Bibliotheken: libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so] und libuuid.[a,so]

Kurze Beschreibungen

badblocks	Durchsucht ein Gerät (üblicherweise eine Festplatte) nach defekten Blöcken.
blkid	Ein Kommandozeilenprogramm zum Auffinden und Anzeigen der Eigenschaften eines Blockgerätes.
chattr	Ändert Attribute eines ext2-Dateisystems. Auch ext3 wird unterstützt (die Journaling-Version von ext2).
compile_et	Ein Fehlertabellen-Compiler. Er konvertiert eine Tabelle mit Fehlercode-Namen und Meldungen zu einer C-Quelldatei, die dann mit der com_err Bibliothek verwendet werden kann.
debugfs	Ein Dateisystemdebugger. Er kann benutzt werden, um den Status eines ext2-Dateisystems zu untersuchen und zu verändern.
dumpe2fs	Gibt Informationen zum Superblock und zu Blockgruppen des Dateisystems auf einem bestimmten Gerät aus.
e2fsck	Wird zum Prüfen und optional zum Reparieren von ext2- und ext3-Dateisystemen verwendet.
e2image	Wird zum Speichern kritischer ext2-Dateisystemdaten in eine Datei verwendet.
e2label	Zeigt oder verändert das Label eines ext2-Dateisystems auf dem angegebenen Gerät.
findfs	Findet ein Dateisystem mit Hilfe des Label oder einer UUID (Universally Unique Identifier).
fsck	Wird zum Prüfen und (optional) Reparieren eines Dateisystems verwendet.
fsck.ext2	Prüft in der Voreinstellung ext2-Dateisysteme.
fsck.ext3	Prüft in der Voreinstellung ext3-Dateisysteme.
logsave	Speichert die Ausgabe eines Kommandos in eine Logdatei.
lsattr	Listet Dateiattribute eines ext2-Dateisystems auf.

mk_cmds	Konvertiert eine Tabelle mit Kommando-Namen und Hilfmeldungen zu C-Quellcode, der dann mit der <code>libss</code> Subsystem-Bibliothek verwendet werden kann.
mke2fs	Wird zum Erstellen eines <code>ext2</code> - oder <code>ext3</code> -Dateisystems auf einem Gerät verwendet.
mkfs.ext2	Erzeugt in der Voreinstellung ein <code>ext2</code> -Dateisystem.
mkfs.ext3	Erzeugt in der Voreinstellung ein <code>ext3</code> -Dateisystem.
mklost+found	Wird benutzt, um den Ordner <code>lost+found</code> auf einem second extended Dateisystem zu erzeugen. Es führt eine Vorzuweisung von Blöcken zu diesem Ordner durch, um damit e2fsck die Arbeit zu erleichtern.
resize2fs	Kann zum Vergrößern oder Verkleinern eines <code>ext2</code> -Dateisystems verwendet werden.
tune2fs	Wird zum Einstellen von veränderbaren Parametern auf einem <code>ext2</code> -Dateisystem eingesetzt.
uuidgen	Erzeugt neue, universell einzigartige Bezeichner (UUID). Jede UUID kann grundsätzlich als einzigartig betrachtet werden, auf dem lokalen oder auf anderen Systemen, in der Vergangenheit und in der Zukunft.
<code>libblkid</code>	Enthält Routinen zum Identifizieren von Geräten und zum Extrahieren von Token.
<code>libcom_err</code>	Die allgemeine Routine zum Anzeigen von Fehlern.
<code>libe2p</code>	Wird von dumpe2fs , chattr und lsattr benutzt.
<code>libext2fs</code>	Enthält Routinen, die Programme im Benutzerkontext zum Manipulieren eines <code>ext2</code> -Dateisystems verwenden können.
<code>libss</code>	Wird von debugfs benutzt.
<code>libuuid</code>	Enthält Routinen zum Erzeugen von einmaligen Bezeichnern für Objekte, die hinter dem lokalen System verfügbar sein könnten.

6.44. Grep-2.5.1a

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 4.5 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed und Texinfo

6.44.1. Installation von Grep

Bereiten Sie Grep zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.44.2. Inhalt von Grep

Installierte Programme: `egrep` ([Link auf grep](#)), `fgrep` ([Link auf grep](#)) und `grep`

Kurze Beschreibungen

- egrep** Gibt die Zeilen aus, die auf einen bestimmten regulären Ausdruck passen.
- fgrep** Gibt die Zeilen aus, die auf eine Liste von festgelegten Zeichenketten passen.
- grep** Gibt die Zeilen aus, die auf einen bestimmten einfachen regulären Ausdruck passen.

6.45. GRUB-0.96

Das Paket Grub enthält den GRand Unified Bootloader.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 10.0 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses und Sed

6.45.1. Installation von GRUB

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch *-march* und *-mcpu*.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. *CFLAGS* und *CXXFLAGS*) für den Kompilervorgang zurückgesetzt oder entsprechend abgeändert werden.

Bereiten Sie GRUB zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Beachten Sie, dass die Test-Ergebnisse immer den Fehler „ufs2_stage1_5 is too big“ ausgeben. Das liegt an einem Compiler-Problem, kann aber ignoriert werden, solange Sie nicht von einer UFS-Partition booten möchten. UFS-Partitionen werden normalerweise nur von Sun Workstations benutzt.

Installieren Sie das Paket:

```
make install
mkdir -v /boot/grub
cp -v /usr/lib/grub/i386-pc/stage{1,2} /boot/grub
```

Ersetzen Sie *i386-pc* durch den für Ihre Plattform korrekten Ordner.

Der Ordner *i386-pc* enthält auch einige **stage1_5*-Dateien, die jeweils für verschiedene Dateisysteme gedacht sind. Schauen Sie nach, welche zur Verfügung stehen und kopieren Sie die notwendigen nach */boot/grub*. Die meisten Leute werden *e2fs_stage1_5* und/oder *reiserfs_stage1_5* kopieren.

6.45.2. Inhalt von GRUB

Installierte Programme: grub, grub-install, grub-md5-crypt, grub-terminfo und mbchk

Kurze Beschreibungen

grub Die GRand Unified Bootloader Kommando-Shell.

grub-install Installiert GRUB auf dem angegebenen Gerät.

grub-md5-crypt	Verschlüsselt Passwörter im MD5-Format.
grub-terminfo	Erzeugt ein terminfo-Kommando aus dem Namen eines Terminals. Es kann verwendet werden, wenn Sie ein unbekanntes Terminal haben.
mbchk	Prüft das Format eines Multiboot-Kernel.

6.46. Gzip-1.3.5

Das Paket Gzip enthält Programme zum Komprimieren und Dekomprimieren von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.2 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

6.46.1. Installation von Gzip

Gzip hat zwei bekannte Sicherheitsmängel. Der folgende Patch behebt beide Probleme:

```
patch -Np1 -i ../gzip-1.3.5-security_fixes-1.patch
```

Bereiten Sie Gzip zum Kompilieren vor:

```
./configure --prefix=/usr
```

Das Skript **gzexe** hat den Pfad zu **gzip** fest eingebaut. Da diese Datei im nachhinein verschoben wird, müssen Sie mit dem folgenden Kommando sicherstellen, dass der korrekte Pfad in das Skript geschrieben wird:

```
sed -i 's@"BINDIR"@/bin@g' gzexe.in
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Verschieben Sie die Datei **gzip** nach `/bin` und erzeugen Sie ein paar übliche Links dorthin:

```
mv -v /usr/bin/gzip /bin
rm -v /usr/bin/{gunzip,zcat}
ln -sv gzip /bin/gunzip
ln -sv gzip /bin/zcat
ln -sv gzip /bin/compress
ln -sv gunzip /bin/uncompress
```

6.46.2. Inhalt von Gzip

Installierte Programme: `compress` (Link auf `gzip`), `gunzip` (Link auf `gzip`), `gzexe`, `gzip`, `uncompress` (Link auf `gunzip`), `zcat` (Link auf `gzip`), `zcmp`, `zdiff`, `zegrep`, `zfgrep`, `zforce`, `zgrep`, `zless`, `zmore` und `znew`

Kurze Beschreibungen

compress Komprimiert und Dekomprimiert Dateien.

gunzip Dekomprimiert gzip-Dateien.

gzexe	Erzeugt selbstextrahierende ausführbare Dateien.
gzip	Komprimiert Dateien mit dem Lempel-Ziv (LZ77) Algorithmus.
uncompress	Entpackt komprimierte Dateien.
zcat	Dekomprimiert gzip-Dateien zur Standardausgabe.
zcmp	Führt cmp auf gzip-Dateien aus.
zdiff	Führt diff auf gzip-Dateien aus.
zegrep	Führt egrep auf gzip-Dateien aus.
zfgrep	Führt fgrep auf gzip-Dateien aus.
zforce	Erzwingt eine <code>.gz</code> -Erweiterung an die komprimierten Dateien, damit gzip diese Dateien nicht erneut komprimiert. Das kann sinnvoll sein, wenn Dateinamen bei einer Datenübertragung abgeschnitten wurden.
zgrep	Führt grep auf gzip-Dateien aus.
zless	Führt less auf gzip-Dateien aus.
zmore	Führt more auf gzip-Dateien aus.
znew	Konvertiert Dateien im compress -Format in das gzip -Format— <code>.Z</code> zu <code>.gz</code> .

6.47. Hotplug-2004_09_23

Die Hotplug-Skripte reagieren auf Hotplug-Ereignisse des Kernels. Jedes Ereignis korrespondiert mit einer entsprechenden Änderung des Kernel-Status aus dem `sysfs`-Dateisystem. Dazu gehört z. B. das Hinzufügen und Entfernen von Hardware. Dieses Paket findet nicht nur später angeschlossene Geräte, sondern auch schon beim Systemstart vorhandene Hardware. Hotplug lädt die Kernel-Module für gefundene Geräte in den laufenden Kernel.

Geschätzte Kompilierzeit: 0.01 SBU

Ungefähr benötigter Festplattenplatz: 460 KB

Die Installation ist abhängig von: Bash, Coreutils, Find, Gawk und Make

6.47.1. Installation von Hotplug

Installieren Sie das Paket Hotplug:

```
make install
```

Kopieren Sie eine Datei, die vom „install“-Skript ausgelassen wird.

```
cp -v etc/hotplug/pnp.distmap /etc/hotplug
```

Entfernen Sie das von Hotplug installierte Init-Skript, denn später soll das mit den LFS-Bootskripten mitgelieferte Skript genutzt werden:

```
rm -rfv /etc/init.d
```

Die LFS-Bootskripte unterstützen derzeit noch kein Hotplugging für Netzwerkgeräte. Aus diesem Grund entfernen Sie bitte diesen Hotplug-Agent:

```
rm -fv /etc/hotplug/net.agent
```

Erzeugen Sie einen Ordner zum Speichern der Firmware, die mit **hotplug** in bestimmte Geräte geladen werden kann:

```
mkdir -v /lib/firmware
```

6.47.2. Inhalt von Hotplug

Installiertes Programm: hotplug

Installierte Skripte: /etc/hotplug/*.rc, /etc/hotplug/*.agent

Installierte Dateien: /etc/hotplug/hotplug.functions, /etc/hotplug/blacklist, /etc/hotplug/{pci,usb}, /etc/hotplug/usb.usermap, /etc/hotplug.d und /var/log/hotplug/events

Kurze Beschreibungen

hotplug

Dieses Skript wird vom Linux-Kernel aufgerufen, wenn sich etwas am internen Status ändert (z. B. wenn ein neues Gerät hinzugefügt oder entfernt wurde).

/etc/hotplug/*.rc

Diese Skripte werden zum sog. Cold-Plugging verwendet. Als Cold-Plugging bezeichnet man z. B. das Erkennen und die allgemeine Handhabung von Hardware, die bereits beim Systemstart vorhanden ist. Sie werden von dem `hotplug` Initskript aufgerufen, welches mit den LFS-Bootskripten mitgeliefert wird. Die `*.rc`-Skripte versuchen, Hotplug-Ereignisse wiederherzustellen, die während dem Systemstart verloren gingen weil z. B. das Basissdateisystem noch nicht eingebunden war.

/etc/hotplug/*.agent

Diese Skripte werden von **hotplug** aufgerufen, wenn bestimmte Arten Hotplug-Ereignisse vom Kernel erzeugt werden. Ihre Aufgabe ist das Laden bestimmter Kernel-Module und das Ausführen benutzerdefinierter Skripte, sofern vorhanden.

/etc/hotplug/blacklist

Diese Datei enthält eine Liste von Modulen, die von `hotplug` nicht in den Kernel geladen werden sollen.

/etc/hotplug/hotplug.functions

Diese Datei enthält Funktionen, die von vielen Hotplug-Skripten verwendet werden.

/etc/hotplug/{pci,usb}

In diesen Ordnern werden benutzerdefinierte Skripte für Hotplug-Ereignisse abgelegt.

/etc/hotplug/usb.usermap

Diese Datei enthält ein Regelwerk aus dem hervorgeht, welche benutzerdefinierten Skripte für bestimmte USB-Geräte aufgerufen werden sollen. Die Geräte werden anhand von Hersteller, Kennung und anderen Eigenschaften erkannt.

/etc/hotplug.d

Dieser Ordner enthält Programme (oder symbolische Links), die Hotplug-Ereignisse erhalten möchten. Z. B. richtet `Udev` während der Installation hier seine Verknüpfung ein.

/lib/firmware

Dieser Ordner enthält die Firmware für Geräte, die vor dem Verwenden die Firmware hochgeladen bekommen müssen.

/var/log/hotplug/events

Diese Datei enthält alle Ereignisse, die **hotplug** seit dem Systemstart aufgerufen hat.

6.48. Man-1.5p

Man enthält Programme zum Finden und Anzeigen von Hilfeseiten (Man-pages).

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.9 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make und Sed

6.48.1. Installation von Man

Zuerst nehmen Sie zwei Anpassungen an den Quellen von Man vor.

Die erste ist eine `sed`-Ersetzung mit der der Parameter `-R` an die Variable `PAGER` angehängt wird, so dass Less korrekt mit Escape-Sequenzen umgeht:

```
sed -i 's@-is@&R@g' configure
```

Die zweite ist ebenfalls eine `sed`-Ersetzung, mit deren Hilfe die Zeile „`MANPATH /usr/man`“ in `man.conf` auskommentiert wird. Dadurch werden redundante Ergebnisse vermieden, wenn Programme wie z. B. **whatis** verwendet werden:

```
sed -i 's@MANPATH./usr/man@#&@g' src/man.conf.in
```

Bereiten Sie Man zum Kompilieren vor:

```
./configure -confdir=/etc
```

Die Bedeutung der `configure`-Parameter:

`-confdir=/etc`

Durch diesen Parameter sucht **man** seine Konfigurationsdatei `man.conf` im Ordner `/etc`.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```



Anmerkung

Falls Sie an einem Terminal arbeiten, das Textattribute für Farbe oder Fettdruck nicht unterstützt, dann können Sie die Escape-Sequenzen für Select Graphic Rendition (SGR) abschalten. Dafür bearbeiten Sie die Datei `man.conf` und fügen den Parameter `-c` an die Variable `NROFF` an. Wenn Sie an einem Rechner verschiedene Terminalarten verwenden, können Sie auch im laufenden Betrieb die Umgebungsvariable `GROFF_NO_SGR` setzen.

Wenn der Zeichensatz für Ihr Locale 8 Bit verwendet, suchen Sie nach der Zeile „NROFF“ in `/etc/man.conf` und stellen Sie sicher, dass sie so aussieht:

```
NROFF /usr/bin/nroff -Tlatin1 -mandoc
```

Beachten Sie, dass „latin1“ auch dann benutzt werden sollte, wenn das nicht der Zeichensatz des aktuellen Locale ist. Das hat den folgenden Grund: Nach der Spezifikation hat **groff** keine Unterstützung für Zeichensätze außerhalb von ISO-8859-1, ausgenommen einiger weniger Escape-Sequenzen. Beim formatieren von Man-pages geht **groff** grundsätzlich davon aus, dass sie in ISO-8859-1 kodiert sind. Der Parameter `-Tlatin1` weist **groff** an, die gleiche Kodierung auch für die Ausgabe zu verwenden. Da **groff** keine Eingaben umkodiert, ist die formatierte Ausgabe somit in der gleichen Kodierung wie die Eingabe. Daher ist diese Kodierung als Eingabe für den Pager verwendbar.

Das löst nicht das Problem des nicht-funktionierenden **man2dvi**-Kommandos für lokalisierte nicht-ISO-8859-1 Locales. Es funktioniert auch nicht mit Multibyte-Zeichensätzen. Zu dem ersten Problem gibt es derzeit keine Lösung. Und das zweite Problem wird nicht behandelt, weil LFS keine Multibyte-Zeichensätze unterstützt.

Weitere Informationen zur Kompression von Man- und Info-pages erhalten Sie im BLFS-Buch unter <http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/compressdoc.html>.

6.48.2. Inhalt von Man

Installierte Programme: apropos, makewhatis, man, man2dvi, man2html und whatis

Kurze Beschreibungen

apropos	Durchsucht die whatis -Datenbank und gibt kurze Beschreibungen zu den Kommandos aus, die die angegebene Zeichenkette enthalten.
makewhatis	Erstellt die whatis -Datenbank. Das Kommando liest alle Man-pages in den Ordnern von <code>MANPATH</code> ein und schreibt für jedes Paket den Namen und eine kurze Beschreibung in die whatis -Datenbank.
man	Formatiert die angeforderte Online Man-page und zeigt sie an.
man2dvi	Konvertiert eine Hilfeseite in das dvi-Format.
man2html	Konvertiert eine Hilfeseite zu HTML.
whatis	Durchsucht die whatis -Datenbank und zeigt eine kurze Beschreibung zu den Systemkommandos an, die das übergebene Stichwort als separates Wort enthalten.

6.49. Make-3.80

Das Paket Make enthält Werkzeuge zum Kompilieren von Software.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 7.1 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep und Sed

6.49.1. Installation von Make

Bereiten Sie Make zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.49.2. Inhalt von Make

Installiertes Programm: make

Kurze Beschreibungen

make Erkennt automatisch, welche Teile eines großen Programms (neu) kompiliert werden müssen und führt automatisch die notwendigen Kommandos aus.

6.50. Module-Init-Tools-3.1

Das Paket Module-Init-Tools enthält diverse Programme zur Verwaltung von Kernel-Modulen für Kernelversionen $\geq 2.5.47$.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 4.9 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make und Sed

6.50.1. Installation von Module-Init-Tools

Module-Init-Tools versucht, die Manpage von `modprobe.conf` beim Kompilieren neu zu schreiben. Das ist erstens unnötig und setzt zweitens **docbook2man** voraus — welches in LFS aber nicht installiert ist. Um das Neuschreiben der Datei zu verhindern, führen Sie bitte dieses Kommando aus:

```
touch modprobe.conf.5
```

Wenn Sie die Testsuite für Module-Init-Tools ausführen möchten, müssen Sie das zusätzliche Archiv herunterladen. Führen Sie dieses Kommando aus, um die Testsuite zu starten (**make distclean** ist nötig um die Quelltexte aufzuräumen, denn Sie werden während dem Testlauf neu kompiliert):

```
tar -xvf ../module-init-tools-testsuite-3.1.tar.bz2 --strip-components=1 &&
./configure &&
make check &&
make distclean
```

Bereiten Sie Module-Init-Tools zum Kompilieren vor:

```
./configure --prefix="" --enable-zlib
```

Die Bedeutung der configure-Parameter:

```
--enable-zlib
```

Durch diesen Parameter kann Module-Init-Tools mit komprimierten Kernel-Modulen umgehen.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.50.2. Inhalt von Module-Init-Tools

Installierte Programme: `depmod`, `insmod`, `insmod.static`, `lsmod` (Link auf `insmod`), `modinfo`, `modprobe` (Link auf `insmod`) und `rmmod` (Link auf `insmod`)

Kurze Beschreibungen

depmod	Erzeugt, basierend auf den Symbolen in existierenden Modulen, eine Abhängigkeitsdatei. Diese Datei wird von modprobe benutzt, um benötigte Module automatisch nachzuladen.
insmod	Installiert ein ladbares Modul in den laufenden Kernel.
insmod.static	Eine statisch kompilierte Version von insmod .
lsmod	Listet die zur Zeit laufenden Kernelmodule auf.
modinfo	Untersucht eine mit einem Kernelmodul assoziierte Objektdatei und zeigt die darin verfügbaren Informationen an.
modprobe	Benutzt eine von depmod erzeugte Abhängigkeitsdatei, um benötigte Module automatisch nachzuladen.
rmmod	Entlädt ein Modul aus dem laufenden Kernel.

6.51. Patch-2.5.4

Das Paket Patch enthält ein Programm zum Erzeugen oder Modifizieren von Dateien indem eine sogenannte „Patch“-Datei angewendet wird. Einen „Patch“ erzeugt man üblicherweise mit **diff** und er beschreibt in maschinenlesbarer Form die Unterschiede zwischen zwei Versionen einer Datei.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 1.5 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

6.51.1. Installation von Patch

Bereiten Sie Patch zum Kompilieren vor (Die Präprozessor-Option `-D_GNU_SOURCE` wird nur auf der PowerPC-Plattform benötigt. Auf anderen Architekturen können Sie sie weglassen.):

```
CPPFLAGS=-D_GNU_SOURCE ./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.51.2. Inhalt von Patch

Installiertes Programm: patch

Kurze Beschreibungen

patch Verändert Dateien nach den Vorgaben einer patch-Datei. Eine patch-Datei ist üblicherweise eine Auflistung von Unterschieden, die mit dem Programm **diff** erzeugt wurde. Durch Anwenden dieser Unterschiede auf die Originaldateien erstellt **patch** eine gepatchte Version.

6.52. Procps-3.2.5

Procps enthält Programme zur Überwachung und Steuerung von Systemprozessen. Die Informationen zu den Prozessen erhält Procps aus dem Ordner /proc.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 2.3 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Make und Ncurses

6.52.1. Installation von Procps

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.52.2. Inhalt von Procps

Installierte Programme: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w und watch

Installierte Bibliothek: libproc.so

Kurze Beschreibungen

free	Gibt die Menge an freiem und benutzten Arbeitsspeicher aus, sowohl physischem als auch Swap.
kill	Sendet Signale an Prozesse.
pgrep	Findet Prozesse aufgrund ihres Namens und anderer Attribute.
pkill	Signalisiert Prozesse basierend auf ihrem Namen oder anderen Attributen.
pmap	Gibt eine Speicherübersicht des angegebenen Prozesses aus.
ps	Listet zur Zeit laufende Prozesse auf.
skill	Sendet Signale an Prozesse, die den angegebenen Kriterien entsprechen.
snice	Ändert die Priorität von Prozessen, die auf die angegebenen Kriterien passen.
sysctl	Ändert Kernelparamter zur Laufzeit.
tload	Gibt eine Grafik der aktuellen durchschnittlichen Systemlast aus.
top	Zeigt eine Liste der Prozesse an, die am meisten CPU-Last erzeugen. Ermöglicht eine Übersicht über laufende Prozesse in Echtzeit.
uptime	Gibt aus, wie lange ein System bereits läuft, wieviele Benutzer eingeloggt sind und wie hoch die Systemlast ist.

- vmstat** Erzeugt Statistiken zur Ausnutzung des virtuellen Speichers, gibt Informationen zu Prozessen, Speicher, Paging, Block-IO, traps und CPU-Aktivität aus.
- w** Zeigt an, welche Benutzer gerade eingeloggt sind, wo, und seit wann.
- watch** Führt ein Kommando immer wieder aus und gibt eine Bildschirmseite von seiner Ausgabe aus. So können Sie die Ausgabe eines Programms beobachten.
- libproc** Enthält Funktionen, die von den meisten Programmen in diesem Paket benutzt werden.

6.53. Psmisc-21.6

Das Paket Psmisc enthält Programme zum Anzeigen von Prozessinformationen.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 1.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses und Sed

6.53.1. Installation von Psmisc

Bereiten Sie Psmisc zum Kompilieren vor:

```
./configure --prefix=/usr --exec-prefix=""
```

Die Bedeutung der configure-Parameter:

```
--exec-prefix=""
```

Dies stellt sicher, dass die Binärdateien von Psmisc nach `/bin` anstelle von `/usr/bin` installiert werden. Lt. FHS ist dies der korrekte Ort, weil einige der Programme in den LFS-Bootskripten verwendet werden.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

`pstree` und `pstree.x11` müssen nicht in `/bin` liegen. Daher verschieben Sie sie nach `/usr/bin`:

```
mv -v /bin/pstree* /usr/bin
```

Normalerweise wird Psmisc's Programm `pidof` nicht installiert. Das ist meistens kein Problem weil wir später das Paket Sysvinit installieren, welches eine bessere Version von `pidof` installiert. Aber wenn Sie nicht Sysvinit verwenden möchten, können Sie die Installation von Psmisc durch Erstellen dieses Links komplettieren:

```
ln -sv killall /bin/pidof
```

6.53.2. Inhalt von Psmisc

Installierte Programme: `fuser`, `killall`, `pstree` und `pstree.x11` (Link auf `pstree`)

Kurze Beschreibungen

- | | |
|----------------|--|
| fuser | Zeigt die PIDs von Prozessen an, die gerade eine bestimmte Datei oder ein Dateisystem verwenden. |
| killall | Beendet Prozesse aufgrund ihres Namens. Es sendet ein Signal an alle Prozesse, die ein |

bestimmtes Kommando ausführen.

pstree

Zeigt laufende Prozesse als Baumstruktur an.

pstree.x11

Das gleiche wie **pstree**, wartet allerdings vor dem Beenden auf eine Bestätigung.

6.54. Shadow-4.0.9

Das Paket Shadow enthält Programme zur sicheren Verwaltung von Kennwörtern.

Geschätzte Kompilierzeit: 0.4 SBU

Ungefähr benötigter Festplattenplatz: 13.7 MB

Die Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

6.54.1. Installation von Shadow



Anmerkung

Wenn Sie sichere Passwörter erzwingen möchten, sollten Sie vor der Installation von Shadow unter <http://www.linuxfromscratch.org/blfs/view/svn/postlfs/cracklib.html> nachlesen und Cracklib installieren. Fügen Sie dann den Parameter `--with-libcrack` zu dem folgenden **configure**-Kommando hinzu:

Bereiten Sie Shadow zum Kompilieren vor:

```
./configure --libdir=/lib --enable-shared
```

Verhindern Sie die Installation des Programmes **groups** und der zugehörigen Man-page, da Coreutils eine bessere Version enthält:

```
sed -i 's/groups$(EXEEXT) //' src/Makefile
sed -i '/groups/d' man/Makefile
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Shadow benutzt zwei Dateien zur Einrichtung der systemweiten Authentifizierungseinstellungen. Installieren Sie diese beiden Konfigurationsdateien:

```
cp -v etc/{limits,login.access} /etc
```

Sie sollten die voreingestellte *crypt*-Methode auf *MD5* ändern, welche sicherer ist. Außerdem ermöglicht sie Passwörter mit mehr als 8 Zeichen. Desweiteren müssen Sie den alten Standort der Benutzermailboxen von `/var/spool/mail` nach `/var/mail` ändern. Das erledigen Sie einfach, indem Sie die Konfigurationsdatei gleich beim Kopieren an die richtige Stelle ändern (benutzen Sie am besten „Kopieren und Einfügen“ um diesen Befehl auszuführen):



Anmerkung

Wenn Sie Shadow mit Unterstützung für Cracklib installieren, dann fügen Sie das folgende **sed**-Kommando ein:

```
-e 's@DICTPATH.*@DICTPATH\t/lib/cracklib/pw_dict@'
```

```
sed -e 's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
    -e 's@/var/spool/mail@/var/mail@' \
    etc/login.defs.linux > /etc/login.defs
```

Verschieben Sie ein Programm an die korrekte Stelle:

```
mv -v /usr/bin/passwd /bin
```

Verschieben Sie Shadow's Bibliotheken an eine bessere Stelle:

```
mv -v /lib/libshadow.*a /usr/lib
rm -v /lib/libshadow.so
ln -sfv ../../lib/libshadow.so.0 /usr/lib/libshadow.so
```

Die Option `-D` zu `useradd` benötigt zur korrekten Funktion diesen Ordner:

```
mkdir -v /etc/default
```

6.54.2. Einrichten von Shadow

Dieses Paket enthält Werkzeuge zum Bearbeiten, Hinzufügen und Löschen von Benutzerpasswörtern. Wir werden hier nicht erläutern, was genau *password shadowing* bedeutet. Eine vollständige Erklärung finden Sie in der Datei `doc/HOWTO` in der entpackten Shadow-Ordnerstruktur. Eines gilt es allerdings zu beachten: Programme, die Passwörter überprüfen müssen (z. B. `xm`, `ftp` und `pop3` Server), müssen *shadow-konform* sein. Das heißt, sie müssen mit Shadow-Passwörtern umgehen können.

Um Shadow-Passwörter zu aktivieren, benutzen Sie dieses Kommando:

```
pwconv
```

Und um Shadow-Gruppenpasswörter zu aktivieren, benutzen Sie dieses Kommando:

```
grpconv
```

Unter normalen Umständen haben Sie bis hierher noch keine Passwörter erzeugt. Wenn Sie jedoch von woanders hierher zurückgeblättert haben um nachträglich Shadow zu aktivieren, dann sollten Sie alle Benutzerpasswörter mit dem Kommando `passwd` und die Gruppenpasswörter mit dem Kommando `gpasswd` zurücksetzen.

6.54.3. Vergeben des Passworts für root

Wählen Sie ein Kennwort für den Benutzer `root` und setzen Sie es mit dem Kommando:

```
passwd root
```

6.54.4. Inhalt von Shadow

Installierte Programme: `chage`, `chfn`, `chpasswd`, `chsh`, `expiry`, `faillog`, `gpasswd`, `groupadd`, `groupdel`, `groupmod`, `grpck`, `grpconv`, `grpunconv`, `lastlog`, `login`, `logoutd`, `mkpasswd`, `newgrp`, `newusers`, `passwd`, `pwck`, `pwconv`, `pwunconv`, `sg` (Link auf `newgrp`), `useradd`, `userdel`, `usermod`, `vigr` (Link auf `vipw`) und `vipw`

Installierte Bibliotheken: `libshadow.[a,so]`

Kurze Beschreibungen

chage	Ändert die maximale Anzahl von Tagen zwischen zwei nötigen Passwortänderungen.
chfn	Wird zum Ändern des vollständigen Namens und weiterer Informationen eines Benutzers benutzt.
chpasswd	Wird benutzt, um das Passwort mehrerer Benutzer in einem Durchlauf zu ändern.
chsh	Wird benutzt, um die voreingestellte Shell eines Benutzers zu ändern.
expiry	Prüft, ob ein Kennwort abgelaufen ist und setzt eine entsprechende Regelung durch.
faillog	Wird zum Untersuchen der Logdatei nach fehlgeschlagenen Logins, zum Setzen einer maximalen Fehlerzahl vor der Sperrung eines Kontos oder um den Zähler zurückzusetzen verwendet.
gpasswd	Wird zum Hinzufügen und Löschen von Mitgliedern in Gruppen verwendet.
groupadd	Erzeugt eine Gruppe mit dem angegebenen Namen.
groupdel	Löscht eine Gruppe mit dem angegebenen Namen.
groupmod	Ändert den Namen oder die GID einer Gruppe.
grpck	Prüft die Integrität der Gruppen-Dateien <code>/etc/group</code> und <code>/etc/gshadow</code> .
grpconv	Erzeugt oder aktualisiert die group-Datei von Shadow aus der normalen group-Datei.
grpunconv	Aktualisiert <code>/etc/group</code> aus <code>/etc/gshadow</code> und löscht die letztere dann.
lastlog	Berichtet über die letzten Anmeldungen aller oder eines bestimmten Benutzers.
login	Wird vom System benutzt, um einen Benutzer anzumelden.
logoutd	Ein Daemon, der Beschränkungen auf die Login-Zeit und -Ports durchsetzt.
mkpasswd	Erzeugt zufällige Passwörter.
newgrp	Wird zum Ändern der aktuellen GID in einer Login-Sitzung benutzt.
newusers	Wird zum Erzeugen oder Aktualisieren einer Serie von Benutzerkonten in einem Durchlauf verwendet.
passwd	Ändert das Passwort für einen Benutzer oder eine Gruppe.
pwck	Prüft die Integrität der Passwort-Dateien <code>/etc/passwd</code> und <code>/etc/shadow</code> .
pwconv	Erzeugt oder aktualisiert die Shadow-Passwort-Datei aus der normalen Passwort-Datei.
pwunconv	Aktualisiert <code>/etc/passwd</code> aus <code>/etc/shadow</code> und löscht letztere danach.
sg	Führt ein Kommando mit der angegebenen GID aus.
su	Führt eine Shell mit geänderter Benutzer- und Gruppen-ID aus.
useradd	Erzeugt einen neuen Benutzer mit dem angegebenen Namen oder aktualisiert die Vorgaben für neue Benutzer.
userdel	Löscht das angegebene Benutzerkonto.

usermod Ändert Loginname, UID, Shell, Gruppe, Persönlichen Ordner und ähnliches für einen Benutzer.
vim Kann zum Bearbeiten von `/etc/group-` oder `/etc/gshadow-`Dateien benutzt werden.
vipw Kann zum Bearbeiten von `/etc/passwd-` oder `/etc/shadow-`Dateien benutzt werden.
`libshadow` Enthält Funktionen, die von den meisten der Programme in diesem Paket verwendet werden.

6.55. Sysklogd-1.4.1

Die in Sysklogd enthaltenen Programme dienen zum Aufzeichnen von Systemmeldungen, zum Beispiel denen des Kernels wenn ungewöhnliche Ereignisse auftreten.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 704 KB

Die Installation ist abhängig von: Binutils, Coreutils, GCC, Glibc und Make

6.55.1. Installation von Sysklogd

Der folgende Patch behebt mehrere Probleme, unter anderem auch ein Kompilierproblem von Sysklogd mit Kernen der 2.6er Serie:

```
patch -Np1 -i ../sysklogd-1.4.1-fixes-1.patch
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.55.2. Einrichtung von Sysklogd

Erstellen Sie nun die Konfigurationsdatei `/etc/syslog.conf`:

```
cat > /etc/syslog.conf << "EOF"
# Begin /etc/syslog.conf

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# log the bootscript output:
local2.* -/var/log/boot.log

# End /etc/syslog.conf
EOF
```

6.55.3. Inhalt von Sysklogd

Installierte Programme: klogd und syslogd

Kurze Beschreibungen

- klogd** Ein System-Daemon zum Abfangen und Protokollieren von Kernel-Meldungen.
- syslogd** Protokolliert Meldungen, die von Systemprogrammen zum Protokollieren angeboten werden. Jede Meldung enthält zumindest einen Datumsstempel und den Hostnamen, und üblicherweise auch den Namen des Programms. Dies ist aber davon abhängig, wie vertrauensselig der Daemon eingestellt wurde.

6.56. Sysvinit-2.86

Das Sysvinit Paket enthält Programme, mit denen Sie das Starten, Ausführen und Beenden des Systems kontrollieren können.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 1012 KB

Die Installation ist abhängig von: Binutils, Coreutils, GCC, Glibc und Make

6.56.1. Installation von Sysvinit

Wenn Runlevel gewechselt werden (zum Beispiel beim Herunterfahren des Systems), sendet **init** Signale an alle Programme, die es gestartet hat. **Init** gibt „Sending processes the TERM signal“ auf dem Bildschirm aus. Dieser Text suggeriert, dass **init** Signale an *alle* Prozesse sendet. Das ist so aber nicht korrekt, denn es geht hier nur um Prozesse, die von **init** gestartet wurden. Um diese Verwirrung zu vermeiden, können Sie die Quellen so modifizieren, dass es sich besser liest: „Sending processes started by init the TERM signal“:

```
sed -i 's@Sending processes@& started by init@g' \  
src/init.c
```

Kompilieren Sie das Paket:

```
make -C src
```

Installieren Sie das Paket:

```
make -C src install
```

6.56.2. Einrichten von Sysvinit

Erstellen Sie die Datei `/etc/inittab`:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty -I '\033(K' tty1 9600
2:2345:respawn:/sbin/agetty -I '\033(K' tty2 9600
3:2345:respawn:/sbin/agetty -I '\033(K' tty3 9600
4:2345:respawn:/sbin/agetty -I '\033(K' tty4 9600
5:2345:respawn:/sbin/agetty -I '\033(K' tty5 9600
6:2345:respawn:/sbin/agetty -I '\033(K' tty6 9600

# End /etc/inittab
EOF
```

Der Parameter `-I '\033(K'` bewirkt, dass **agetty** als erstes diese Escape-Sequenz an das Terminal sendet. Mit dieser Sequenz wird das Terminal auf einen benutzerdefinierten Zeichensatz umgeschaltet, der dann mit **setfont** eingestellt werden kann. Die Konsole-Initkripte aus den LFS-Bootskripten benutzen **setfont** während dem Start. Das Senden dieser Sequenz ist für Personen wichtig, die nicht-ISO 8859-1 Bildschirmschriften benutzen und hat keinen Effekt für englischsprachige Benutzer.

6.56.3. Inhalt von Sysvinit

Installierte Programme: halt, init, killall5, last, lastb (Link auf last), mesg, pidof (Link auf killall5), poweroff (Link auf halt), reboot (Link auf halt), runlevel, shutdown, sulogin, telinit (Link auf init), utmpdump und wall

Kurze Beschreibungen

halt	Ruft üblicherweise shutdown mit dem Parameter <code>-h</code> auf, außer wenn der aktuelle Runlevel 0 ist, dann teilt es dem Kernel mit, das System anzuhalten. Vorher vermerkt es in <code>/var/log/wtmp</code> , dass das System nun heruntergefahren wird.
init	Der erste gestartete Prozess nachdem der Kernel die Hardware initialisiert hat. Init übernimmt den Bootvorgang und startet alle anstehenden Programme.
killall5	Sendet ein Signal an alle Prozesse, außer denen in der eigenen Sitzung—so beendet es nicht die Programme, die das Skript ausführen welches es aufgerufen hat.
last	Zeigt, welcher Benutzer als letztes eingeloggt und ausgeloggt hat, indem es die Datei <code>/var/log/wtmp</code> durchsucht. Es kann auch Systemstarts und -stopps sowie Wechsel der Runlevel zeigen.
lastb	Zeigt die letzten fehlgeschlagenen Login-Versuche, die in <code>/var/log/btmp</code> protokolliert wurden.
mesg	Kontrolliert, welche anderen Benutzer Nachrichten auf das aktuelle Terminal senden können.
mountpoint	Prüft, ob der Ordner ein Mountpunkt ist.
pidof	Gibt die PIDs eines Programms aus.
poweroff	Weist den Kernel an, das System anzuhalten und den Computer auszuschalten. Siehe auch die Beschreibung zu halt .
reboot	Weist den Kernel an, das System neu zu starten. Siehe auch die Beschreibung zu halt .
runlevel	Zeigt den vorigen und den aktuellen Runlevel an. Die nötigen Informationen werden aus <code>/var/run/utmp</code> gelesen.
shutdown	Fährt das System sicher herunter, sendet entsprechende Signale an alle Prozesse und benachrichtigt alle angemeldeten Benutzer.
sulogin	Erlaubt <i>root</i> , sich einzuloggen. Es wird normalerweise von init gestartet, wenn das System im Einbenutzermodus gestartet wurde.
telinit	Weist init an, in den angegebenen Runlevel zu wechseln.
utmpdump	Zeigt den Inhalt der angegebenen Logindatei in einem benutzerfreundlicheren Format an.
wall	Schreibt eine Nachricht an alle angemeldeten Benutzer.

6.57. Tar-1.15.1

Das Paket Tar enthält ein Archivprogramm.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 12.7 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make und Sed

6.57.1. Installation von Tar

Tar hat bei Dateien größer 4GB ein Problem mit dem Parameter `-S`. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../tar-1.15.1-sparse_fix-1.patch
```

Bereiten Sie Tar zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin --libexecdir=/usr/sbin
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie `make check` aus.

Installieren Sie das Paket:

```
make install
```

6.57.2. Inhalt von Tar

Installierte Programme: rmt und tar

Kurze Beschreibungen

- rmt** Mit diesem Programm kann man ein magnetorientiertes Bandlaufwerk an einem entfernten Rechner steuern. Zur Kommunikation wird Interprozesskommunikation verwendet.
- tar** Wird zum Erzeugen, Auflisten und Extrahieren von Dateien aus einem Archiv verwendet. Diese Archive werden oft auch als „Tarball“ bezeichnet.

6.58. Udev-056

Das Paket Udev enthält Programme zum dynamischen Erzeugen von Gerätedateien.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 6.7 MB

Die Installation ist abhängig von: Coreutils und Make

6.58.1. Installation von Udev

Kompilieren Sie das Paket:

```
make udevdir=/dev
```

udevdir=/dev

Dieser Parameter gibt an, in welchem Ordner **udev** Gerätedateien erzeugen soll.

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **make test**.

Installieren Sie das Paket:

```
make DESTDIR=/ udevdir=/dev install
```

Die Bedeutung der make-Option:

DESTDIR=/

Dies verhindert, dass die Installationsroutine von Udev jegliche Instanzen von **udev** beendet, die möglicherweise auf dem Host-System laufen.

Udev's Konfigurationsdateien sind alles andere als optimal, installieren Sie daher diese Konfigurationsdateien:

```
cp -v ../udev-config-4.rules /etc/udev/rules.d/25-lfs.rules
```

Führen Sie **udevstart** aus, um eine vollständigen Satz an Gerätedateien zu erzeugen.

```
/sbin/udevstart
```

6.58.2. Inhalt von Udev

Installierte Programme: udev, udevd, udevsend, udevstart, udevinfo und udevtest

Installierter Ordner: /etc/udev

Kurze Beschreibungen

udev Erzeugt Gerätedateien in /dev oder benennt als Reaktion auf Hotplug-Ereignisse Netzwerkgeräte um (nicht in LFS).

udev	Ein Daemon, der Hotplug-Ereignisse umsortiert bevor er sie an udev weiterreicht. Damit werden bestimmte sog. Race conditions verhindert.
udevsend	Übermittelt Hotplug-Ereignisse an udev .
udevstart	Erzeugt Gerätedateien zu Gerätetreibern, die fest in den Kernel einkompiliert sind. Dazu simuliert es die Hotplug-Ereignisse, die vom Kernel verworfen wurden, bevor das Programm gestartet wurde (z. B. weil das Basis-Dateisystem noch nicht eingehängt war) und übermittelt diese synthetischen Hotplug-Ereignisse an udev .
udevinfo	Ermöglicht Anwendern, die udev -Datenbank nach Informationen über zur Zeit verfügbare Geräte im System abzufragen. Es stellt außerdem eine Möglichkeit dar, jedes Gerät im <code>sysfs</code> -Dateisystem abzufragen, um beim Erzeugen von udev-Regeln behilflich zu sein.
udevtest	Simuliert einen udev -Durchlauf für das angegebene Gerät und gibt den Namen der Gerätedatei oder des Netzwerkgerätes (nicht in LFS) aus, die ein echter udev -Aufruf für dieses Gerät erzeugt hätte.
<code>/etc/udev</code>	Enthält Konfigurationsdateien, Geräteberechtigungen und Regeln für die Namensvergabe von udev .

6.59. Util-linux-2.12q

Das Paket Util-linux enthält verschiedene Werkzeuge. Darunter befinden sich Programme zum Umgang mit Dateisystemen, Konsolen, Partitionen und (System-)Meldungen.

Geschätzte Kompilierzeit: 0.2 SBU

Ungefähr benötigter Festplattenplatz: 11.6 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed und Zlib

6.59.1. Anmerkung zur FHS-Konformität

FHS empfiehlt, `/var/lib/hwclock` anstelle des eigentlich üblichen Ordners `/etc` als Speicherort für die Datei `adjtime` zu benutzen. Führen Sie das folgende Kommando aus, um das Programm **hwclock** FHS-Konform zu machen:

```
sed -i 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
    hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

6.59.2. Installation von Util-linux

Util-linux lässt sich mit neueren Versionen der Linux-Libc-Header nicht kompilieren. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../util-linux-2.12q-cramfs-1.patch
```

In Util-Linux gibt es ein Sicherheitsproblem, das es einem Benutzer ermöglicht, ein Gerät ohne die Option `nosuid` neu einzuhängen. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../util-linux-2.12q-umount_fix-1.patch
```

Bereiten Sie Util-linux zum Kompilieren vor:

```
./configure
```

Kompilieren Sie das Paket:

```
make HAVE_KILL=yes HAVE_SLN=yes
```

Die Bedeutung der make-Parameter:

HAVE_KILL=yes

Verhindert, dass das Programm **kill** (bereits durch Procps installiert) erneut kompiliert und installiert wird.

HAVE_SLN=yes

Verhindert, dass das Programm **sln** (eine statisch gelinkte Version von **ln**, bereits durch Glibc installiert) erneut kompiliert und installiert wird.

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket und verschieben Sie die Datei **logger** nach `/bin`, denn es wird von den LFS-Bootskripten benötigt:

```
make HAVE_KILL=yes HAVE_SLN=yes install
mv /usr/bin/logger /bin
```

6.59.3. Inhalt von Util-linux

Installierte Programme: agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, pg, pivot_root, ramsize (Link auf rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (Link auf rdev), script, setfdprm, setsid, setterm, sfdisk, swapdev, swapoff (Link auf swapon), swapon, tunelp, ul, umount, vidmode (Link auf rdev), whereis und write

Kurze Beschreibungen

agetty	Öffnet einen tty-Port, fragt nach dem Login-Namen und startet das Programm login .
arch	Gibt die Systemarchitektur aus.
blockdev	Ermöglicht den Aufruf von Blockgeräte-ioctls an der Kommandozeile.
cal	Zeigt einen einfachen Kalender an.
cfdisk	Wird zum Bearbeiten der Partitionstabelle eines Gerätes benutzt.
chkdupexe	Findet Duplikate von ausführbaren Dateien.
col	Filtert Rückwärts-Zeilenvorschübe aus.
colcrt	Filtert nroff -Ausgaben für Terminals, denen bestimmte Fähigkeiten fehlen, wie zum Beispiel durchstreichen oder halbe Zeilen.
colrm	Filtert eine bestimmte Spalte aus.
column	Formatiert eine Datei in mehrere Spalten.
ctrlaltdel	Setzt die Funktion der Tastenkombination Strg-Alt-Entf auf einen Hart- oder Softreset.
cytune	Wurde benutzt, um die Parameter der seriellen Schnittstellen auf Cyclade-Karten zu verändern.
ddate	Gibt das Diskordianische Datum aus, oder konvertiert ein Gregorianisches Datum in ein Diskordianisches.
dmesg	Zeigt die Bootmeldungen des Kernel an.
elvtune	Kann zum Manipulieren der Performance und Interaktivität von Blockgeräten benutzt werden.
fdformat	Formatiert eine Diskette low-level.
fdisk	Wird zum Bearbeiten der Partitionstabelle eines Gerätes benutzt.

fsck.cramfs	Führt eine Konsistenzprüfung auf einem Cramfs-Dateisystem durch.
fsck.minix	Führt eine Konsistenzprüfung auf einem Minix-Dateisystem durch.
getopt	Analysiert die Optionen in der Kommandozeile.
hexdump	Zeigt eine Datei hexadezimal oder in einem anderen Format an.
hwclock	Wird zum Setzen oder Lesen der Hardware-Uhr (auch RTC- oder BIOS-Uhr genannt) benutzt.
ipcrm	Entfernt die angegebene IPC-Ressource (Inter-Process Communication).
ipcs	Gibt IPC Status-Informationen aus.
isozsize	Gibt die Größe eines iso9660-Dateisystems aus.
line	Kopiert eine einzelne Zeile.
logger	Gibt eine Nachricht in das Logsystem ein.
look	Sucht nach Zeilen, die mit einer bestimmten Zeichenkette beginnen, und zeigt sie an.
losetup	Konfiguriert und kontrolliert Loopback-Geräte.
mcookie	Erzeugt magische Cookies (hexadezimale 128-bit Zufallszahlen) für xauth.
mkfs	Erzeugt ein Dateisystem auf einem Gerät (üblicherweise einer Festplattenpartition).
mkfs.bfs	Erzeugt ein SCO-bfs-Dateisystem (Santa Cruz Operations).
mkfs.cramfs	Erzeugt ein cramfs-Dateisystem.
mkfs.minix	Erzeugt ein Minix-Dateisystem.
mkswap	Initialisiert ein Gerät oder eine Datei als Auslagerungsbereich.
more	Ein Filter zum seitenweisen Anzeigen von Text. Less ist jedoch besser.
mount	Hängt das auf dem Gerät vorhandene Dateisystem in einem Ordner ein.
namei	Zeigt die symbolischen Links in Pfadnamen an.
pg	Zeigt eine Textdatei seitenweise an.
pivot_root	Macht ein Dateisystem zu dem neuen root-Dateisystem für den aktuellen Prozess.
ramsize	Kann zum Setzen der Größe einer RAM-Disk in einem bootbaren Abbild benutzt werden.
raw	Bindet ein zeichenorientiertes Linux-raw-Gerät an ein Blockgerät.
rdev	Kann in einem bootfähigen Abbild das root-Gerät abfragen und festlegen.
readprofile	Liest Profiling-Informationen aus dem Kernel.
rename	Benennt eine Datei um und ersetzt ein Zeichenkette durch eine andere.
renice	Verändert die Priorität eines Prozesses.
rev	Dreht die Zeilen einer Datei um.
rootflags	Kann die root-Parameter eines bootfähigen Abbildes festlegen.

script	Erstellt eine Abschrift einer Terminalsitzung.
setfdprm	Setzt benutzerdefinierte Floppy-Disk-Parameter.
setsid	Führt ein Kommando in einer neuen Sitzung aus.
setterm	Stellt Terminal-Attribute ein.
sfdisk	Kann Festplattenpartitionen bearbeiten.
swapdev	Setzt ein Swap-Gerät in einem bootfähigen Abbild.
swapoff	Deaktiviert Auslagerungsdateien und -geräte.
swapon	Aktiviert Auslagerungsdateien und -geräte und zeigt bereits verwendete Geräte und Dateien an.
tunelp	Justiert Parameter eines Zeilendruckers.
ul	Ein Filter zum Übersetzen von Unterstrichen in entsprechende Escape-Sequenzen, die das verwendete Terminal versteht.
umount	Löst ein Dateisystem aus der Ordnerstruktur.
vidmode	Kann zum Setzen des Videomodus in einem bootfähigen Abbild benutzt werden.
whereis	Gibt den Ort der Binärdatei, der Quellen und der Man-pages für ein Kommando aus.
write	Sendet eine Nachricht an einen Benutzer (sofern der Benutzer den Empfang solcher Nachrichten nicht deaktiviert hat).

6.60. Informationen zu Debugging Symbolen

Die meisten Programme und Bibliotheken werden in der Voreinstellung mit Debugging-Symbolen kompiliert (mit der Option `gcc -g`). Wenn Sie ein Programm oder eine Bibliothek debuggen, die mit debugging Symbolen kompiliert wurde, kann Ihnen der Debugger nicht nur die Speicheradressen, sondern auch die Namen der Funktionen und der Variablen im Programm anzeigen.

Doch das Einbinden dieser Debugging-Symbole vergrößert das Programm bzw. die Bibliothek deutlich. Das folgende Beispiel soll Ihnen einen Eindruck über den von Debugging-Symbolen benötigten Speicher geben:

- Eine Bash-Binärdatei mit Debugging-Symbolen: 1200 KB
- Eine Bash-Binärdatei ohne Debugging-Symbole: 480 KB
- Glibc und GCC-Dateien (`/lib` und `/usr/lib`) mit Debugging-Symbolen: 87 MB
- Glibc und GCC-Dateien ohne Debugging-Symbole: 16 MB

Die Größen variieren ein wenig, abhängig vom Compiler und der eingesetzten C-Bibliothek. Aber wenn man Programme mit und ohne Debugging-Symbole vergleicht, liegt der Faktor normalerweise zwischen 2 und 5.

Vermutlich werden Sie niemals einen Debugger mit Ihrer Systemsoftware einsetzen, daher können Sie durch das Entfernen der Symbole eine Menge Platz sparen. Der Einfachheit halber finden Sie im nächsten Kapitel ein Kommando, mit dem Sie alle debugging Symbole von allen Programmen und Bibliotheken auf Ihrem System entfernen können. Weitere Informationen zum Thema Optimierung finden Sie in der Anleitung unter <http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt>.

6.61. Erneutes Stripping

Da Sie Ihre Systemsoftware vermutlich nicht debuggen möchten, können Sie hier ca. 200MB Platz sparen. Dazu entfernen Sie die Debugging-Symbole. Das zieht keine Probleme nach sich, aber Sie können die verkleinerten Programme danach nicht mehr vollständig debuggen.

Normalerweise gibt es mit dem folgenden Kommando keine Schwierigkeiten. Aber Sie könnten z. B. einen Tippfehler machen und dadurch das System unbrauchbar machen. Bevor Sie **Strip** ausführen, sollten Sie ein Backup machen.

Wenn Sie strip ausführen möchten, ist besondere Vorsicht geboten, damit Sie strip nicht auf Programme anwenden, die gerade ausgeführt werden—inklusive der Bash-Shell. Daher müssen Sie die chroot-Umgebung vorerst verlassen:

logout

Und dann erneut betreten:

```
chroot $LFS /tools/bin/env -i \
  HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
  PATH=/bin:/usr/bin:/sbin:/usr/sbin \
  /tools/bin/bash --login
```

Nun können die Debugging-Symbole sicher aus Binärdateien und Bibliotheken entfernt werden:

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
  -exec /tools/bin/strip --strip-debug '{} ' ';'
```

Es werden viele Dateien gemeldet, deren Format nicht erkannt wurde. Die meisten dieser Dateien sind Skripte und keine Binärdateien. Die Warnungen können einfach ignoriert werden.

Wenn Sie sehr wenig Platz auf der Festplatte haben, können Sie `--strip-all` auf die Binärdateien in `{,usr/}{bin,sbin}` anwenden und so nochmals mehrere Megabytes sparen. Benutzen Sie diese Option jedoch nicht mit Bibliotheken—sie würden zerstört werden.

6.62. Aufräumen

Von nun an müssen Sie das folgende Kommando zum Betreten der chroot-Umgebung verwenden:

```
chroot "$LFS" /usr/bin/env -i \  
  HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \  
  PATH=/bin:/usr/bin:/sbin:/usr/sbin \  
  /bin/bash --login
```

Der Grund dafür ist, dass Sie keine Programme mehr aus `/tools` benötigen. Sie können den Ordner nun löschen. Bevor Sie `/tools` löschen, möchten Sie den Ordner vielleicht in ein Tar-Archiv packen und an einem sicheren Ort aufheben, z. B. weil Sie vielleicht bald noch ein LFS-System bauen möchten.



Anmerkung

Wenn Sie `/tools` löschen, werden auch die temporären Kopien von Tcl, Expect und DejaGNU gelöscht (die Sie zum Testen der Toolchain benutzt haben). Wenn Sie diese Programme später noch benutzen möchten, müssen Sie sie neu kompilieren und installieren. Im BLFS-Buch finden Sie die entsprechenden Anleitungen dafür (siehe auch <http://www.linuxfromscratch.org/blfs/>).

Kapitel 7. Aufsetzen der System-Bootskripte

7.1. Einführung

In diesem Kapitel werden Sie die LFS-Bootskripte aufsetzen. Die meisten Skripte funktionieren ohne Anpassungen, aber ein paar benötigen eine Konfigurationsdatei weil sie beispielsweise mit Hardware an Ihrem Computer zu tun haben.

LFS verwendet Bootsripte im gebräuchlichen System-V-Stil. Es gibt auch andere Möglichkeiten. Z. B. finden Sie unter <http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt> eine Anleitung für BSD-Init. Oder durchsuchen Sie die LFS-Mailinglisten nach „depinit“ um eine andere Variante zu versuchen.

Falls Sie sich für etwas ganz anderes entscheiden sollten, können Sie dieses Kapitel ganz überspringen und direkt bei Kapitel 8 fortfahren.

7.2. LFS-Bootskripte-3.2.1

Das Paket LFS-Bootskripte enthält die Skripte zum Starten und Stoppen des Systems beim Booten und Herunterfahren Ihres Computers.

Geschätzte Kompilierzeit: 0.1 SBU

Ungefähr benötigter Festplattenplatz: 0.3 MB

Die Installation ist abhängig von: Bash und Coreutils

7.2.1. Installation von LFS-Bootskripte

Installieren Sie das Paket:

```
make install
```

7.2.2. Inhalt von LFS-Bootskripte

Installierte Skripte: checkfs, cleanfs, console, functions, halt, hotplug, ifdown, ifup, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, template und udev

Kurze Beschreibungen

checkfs	Prüft die Integrität von Dateisystemen bevor sie eingehängt werden (mit der Ausnahme von journal- und netzwerkbasierten Dateisystemen).
cleanfs	Entfernt Dateien, die nicht über einen Neustart hinaus existieren sollten. Dazu gehören zum Beispiel die Dateien in <code>/var/run/</code> und <code>/var/lock/</code> . Es erzeugt <code>/var/run/utmp</code> und entfernt die eventuell vorhandenen Dateien <code>/etc/nologin</code> , <code>/fastboot</code> und <code>/forcefsck</code> .
console	Läd das für Ihre Tastatur korrekte Tastaturlayout und stellt die Bildschirmschriftart ein.
functions	Enthält allgemeine Funktionen die von verschiedenen Skripten genutzt werden. Dazu gehören z. B. Fehler- oder Statusprüfung.
halt	Hält das System an.
hotplug	Lädt Module für Systemgeräte.
ifdown	Unterstützt das Netzwerkskript beim Stoppen von Netzwerkgeräten.
ifup	Unterstützt das Netzwerkskript beim Starten von Netzwerkgeräten.
localnet	Setzt den Hostnamen und das lokale Loopback-Gerät auf.
mountfs	Hängt alle nicht als <i>noauto</i> markierten und nicht netzwerkbasierten Dateisysteme ein.
mountkernfs	Hängt virtuelle Kernel-basierte Dateisysteme ein (z. B. <code>proc</code>).
network	Macht Netzwerkschnittstellen wie z. B. Netzwerkkarten verfügbar und richtet — wenn nötig — das Standard-Gateway ein.

rc	Das Haupt-Runlevel-Kontrollskript. Es ist dafür verantwortlich, alle anderen Skripte eins nach dem anderen in der richtigen Reihenfolge auszuführen.
reboot	Startet das System neu.
sendsignals	Stellt sicher, dass jeder Prozess beendet wird, bevor das System herunterfährt oder neu startet.
setclock	Setzt die Kernelzeit auf lokale Zeit, falls die Hardware-Uhr nicht auf UTC-Zeit eingestellt ist.
static	Stellt Funktionen zum Zuweisen einer statischen IP-Adresse an ein Netzwerkgerät zur Verfügung.
swap	Aktiviert und deaktiviert Swap-Dateien und -Partitionen.
sysklogd	Startet und stoppt die System- und Kernel-Log-Daemons.
template	Eine Vorlage, die Sie verwenden können, um Ihre eigenen Bootskripte für eigene Daemons zu schreiben.
udev	Bereitet <code>/dev</code> vor und startet Udev.

7.3. Wie funktionieren diese Bootskripte?

Linux benutzt eine spezielle Bootmethode mit dem Namen SysVinit. Sie basiert auf dem Konzept der *Runlevel*. Dieses Konzept kann in verschiedenen Distributionen sehr unterschiedlich umgesetzt sein. Nehmen Sie also nicht an, nur weil etwas in Distribution XY funktioniert, geht es in LFS auf die gleiche Weise. LFS respektiert zwar allgemein übliche Standards, geht aber dennoch (wie alle anderen) seinen eigenen Weg.

SysVinit (wir nennen es nun einfach nur „init“) funktioniert nach dem Konzept der Runlevel. Es gibt 7 Runlevel (von 0 bis 6), genaugenommen gibt es sogar noch mehr, aber diese sind für Spezialfälle reserviert und werden üblicherweise nicht benutzt. `init(8)` beschreibt diese Details genauer. Jeder Runlevel korrespondiert mit Skripten oder Diensten, die der Computer beim Hochfahren ausführen bzw. starten oder stoppen soll. Der Standard-Runlevel ist 3. Hier sehen Sie eine Übersicht, wie die Runlevel üblicherweise eingesetzt werden:

```
0: Fährt den Computer herunter
1: Ein-Benutzer-Modus
2: Mehr-Benutzer-Modus ohne Netzwerk
3: Mehr-Benutzer-Modus mit Netzwerk
4: reserviert für eigene Anpassungen, funktioniert ansonsten wie 3
5: genauso wie 4, wird normalerweise für grafischen Login benutzt (wie z. B. X's xdm oder KDE's kdm)
6: Startet den Computer neu
```

Das Kommando zum Wechseln des Runlevel ist `init [Runlevel]`, wobei `[Runlevel]` der Runlevel ist, in den Sie wechseln möchten. Zum Neustarten des Computers würde ein Benutzer zum Beispiel `init 6` eingeben. Das `reboot`-Kommando ist nur ein Alias darauf, genauso wie das Kommando `halt` ein Alias auf `init 0` ist.

Unter `/etc/rc.d` befinden sich eine Menge Ordner mit dem Namen `rc?.d`, wobei das `?` die Nummer eines Runlevels ist. Dort liegt auch der Ordner `rcsysinit.d`, er enthält einige symbolische Links. Einige beginnen mit einem `K`, andere mit einem `S`, gefolgt von einer zweistelligen Zahl. Das `K` bedeutet beenden (*kill*) eines Dienstes, das `S` bedeutet starten (*start*) eines Dienstes. Die Zahlen bestimmen die Reihenfolge, in der die Skripte ausgeführt werden und können zwischen 00 und 99 liegen. Je kleiner die Zahl, desto früher wird das Skript ausgeführt. Wenn `init` in einen anderen Runlevel wechselt, werden die nötigen Skripte gestoppt und andere dafür gestartet.

Bisher war nur von Links die Rede. Die echten Skripte befinden sich in `/etc/rc.d/init.d`. Sie erledigen die eigentliche Arbeit, denn die ganzen symbolischen Links zeigen nur auf sie. Stopp- und Startskripte zeigen jeweils auf dieselbe Datei in `/etc/rc.d/init.d`. Das funktioniert, weil die Bootskripte mit unterschiedlichen Parametern aufgerufen werden können: zum Beispiel `start`, `stop`, `restart`, `reload`, `status`. Wenn ein `K`-Link ausgeführt werden soll, wird das entsprechende Skript mit dem `stop`-Parameter aufgerufen. Wenn ein `S`-Link ausgeführt werden soll, wird das Skript mit dem `start`-Parameter aufgerufen.

Es gibt eine Ausnahme: `S`-Links in den Ordnern `rc0.d` und `rc6.d` starten keine Dienste. Sie werden stattdessen mit dem Parameter `stop` aufgerufen um etwas zu beenden. Die Grund dafür ist, dass Sie wohl kaum einen Dienst starten möchten, wenn Sie rebooten oder das System herunterfahren.

Hier die Beschreibungen, welche Parameter zu einem Skript was bewirken:

```
start
```

Der Dienst wird gestartet.

```
stop
```

Der Dienst wird gestoppt.

restart

Der Dienst wird gestoppt und dann erneut gestartet.

reload

Die Konfiguration des Dienstes wird neu eingelesen. Das verwendet man, nachdem die Konfigurationsdatei eines Dienstes geändert wurde und man nicht den ganzen Dienst neu starten muss.

status

Gibt aus, ob der Dienst läuft, und wenn ja, mit welchen PIDs.

Sie können den Bootprozess natürlich nach Ihren Wünschen anpassen (schlussendlich ist es ja Ihr eigenes Linux). Die Dateien hier sind nur Beispiele dafür, wie man es gut erledigen kann.

7.4. Umgang mit Geräten und Modulen an einem LFS-System

In Kapitel 6 haben Sie Udev installiert. Bevor wir zu den Details kommen wie das alles funktioniert, möchten wir Ihnen erst einen Rückblick darüber geben, wie man früher mit Geräten unter Linux umgegangen ist.

Traditionell hat man unter Linux eine statische Methode zum Erzeugen von Gerätedateien benutzt. Dabei wurden sehr viele Gerätedateien vorab in `/dev` erzeugt (manchmal mehrere tausend). Dabei war es völlig egal, ob die zugehörige Hardware tatsächlich existierte oder nicht. Dies wurde typischerweise durch das Skript **MAKEDEV** erledigt, welches eine Menge Systemaufrufe mit dem Programm **mknod** und den entsprechenden Gerätenummern durchführte und so Gerätedateien zu allen erdenklichen Geräten erzeugte. Mit der Udev-Methode werden nur die Gerätedateien erzeugt, zu denen der Kernel auch ein Gerät gefunden hat. Weil diese Gerätedateien bei jedem Systemstart neu erzeugt werden, speichert man sie auf einem sog. `tmpfs`-Dateisystem. Dieses Dateisystem existiert nur im Arbeitsspeicher und verbraucht daher keinen Festplattenplatz. Gerätedateien benötigen kaum Platz, auf diese Weise wird also nur sehr wenig Arbeitsspeicher verbraucht.

7.4.1. Die Entwicklungsgeschichte von Udev

Im Februar 2000 wurde ein neues Dateisystem mit dem Namen `devfs` in den Kernel 2.3.46 integriert und dann in der 2.4er Serie der stabilen Kernel verfügbar gemacht. Obwohl es in den Kernelquellen selbst verfügbar war, hat diese Methode nie wirkliche Unterstützung von den Kernel-Entwicklern bekommen.

Das Haupt-Problem bei diesem von `devfs` adaptierten Ansatz war die Art und Weise, auf die Geräte erkannt, erzeugt und benannt wurden. Letzteres (Namensvergabe) war wohl das kritischste Problem. Das Dateisystem `devfs` litt außerdem unter sog. Race conditions die mit dem Konzept zusammenhängen und nicht ohne nennenswerte Änderungen am Kernel geändert werden konnten. Außerdem wurde es als „missbilligt“ markiert, weil es nicht mehr gepflegt wurde.

Mit der Entwicklung der 2.5er Entwickler-Kernelserie, die später als 2.6er Serie stabil veröffentlicht wurde, wurde ein neues Dateisystem mit dem Namen `sysfs` eingeführt. Die Aufgabe von `sysfs` ist es, die Systemstruktur an Anwenderprozesse zu exportieren. Mit dieser aus der Anwendersicht sichtbaren Repräsentation der Systemstruktur wurde ein Ersatz für `devfs` realistisch.

7.4.2. Udev-Implementierung

Das Dateisystem `sysfs` wurde oben schon kurz erwähnt. Man fragt sich vielleicht, woher `sysfs` von den Geräten und den zu verwendenden Gerätenummern weiß: Treiber, die direkt in den Kernel integriert wurden, registrieren sich bei `sysfs` sobald sie vom Kernel erkannt werden. Bei Kernel-Modulen geschieht dieser Vorgang beim Laden des Moduls. Sobald `sysfs` in das System eingehängt ist (`/sys`), sind die Daten von den mit `sysfs` registrierten Treibern für Prozesse aus der Anwenderschicht, und damit auch für **udev**, verfügbar.

Das Initskript **S10udev** kümmert sich darum, diese Gerätedateien beim Systemstart zu erzeugen. Als erstes registriert das Skript `/sbin/udevsend` als Programm zur Verarbeitung von Hotplug-Ereignissen. In dieser Phase sollten noch keine Hotplug-Ereignisse generiert werden (dies wird weiter unten erklärt), aber für den Fall der Fälle wird **udev** schon mal registriert. Das Programm **udevstart** durchsucht dann das Dateisystem unter `/sys` und erzeugt die entsprechenden passenden Gerätedateien in `/dev`. Zum Beispiel enthält `/sys/class/tty/vcs/dev` die Zeichenkette „7:0“. Diese Zeichenkette wird von **udevstart** benutzt, um `/dev/vcs` mit der Hauptkennung 7 und der Unterkennung 0 zu erzeugen. Die Namen und Rechte eines jeden Gerätes entnimmt **udevstart** aus den Dateien im Ordner `/etc/udev.d/rules.d/`. Diese Dateien sind ähnlich durchnummeriert wie die LFS Bootskripte. Falls **udev** für eine Gerätedatei keine Datei mit

Berechtigungen finden kann, wird die Voreinstellung `660` und der Besitzer `root:root` festgelegt.

Sobald diese Phase abgeschlossen ist, sind die Geräte, die schon angeschlossen sind und für die ein Treiber im Kernel fest einkompiliert ist, verfügbar und einsatzbereit. Doch was ist mit den Geräten, die modulare Treiber haben?

Weiter oben wurde das Konzept der „Verarbeitung von Hotplug-Ereignissen“ angesprochen. Wenn vom Kernel eine neue Verbindung mit einem Gerät erkannt wird, erzeugt der Kernel ein sog. Hotplug-Ereignis und schaut in der Datei `/proc/sys/kernel/hotplug` nach dem Programm, das die Einbindung des neuen Gerätes vornimmt. Das Initskript **udev** hat **udevsend** als Programm zur Verarbeitung von Hotplug-Ereignissen registriert. Wenn also Hotplug-Ereignisse erzeugt werden, teilt der Kernel **udev** mit, in `/sys` nach den Informationen zu diesem neuen Gerät zu schauen und **udev** erzeugt dann die dazu passende Gerätedatei in `/dev`.

Das führt uns zu dem Problem mit **udev**, das in ähnlicher Form auch schon in `devfs` existierte. Es wird umgangssprachlich als das „Henne und Ei“-Problem bezeichnet. Die meisten Linux-Distributionen laden Module über Einträge in `/etc/modules.conf`. Veranlasst wird das Laden eines Moduls durch den Zugriff auf die zugehörige Gerätedatei. Mit **udev** funktioniert diese Methode aber nicht, weil die Gerätedatei nicht existiert solange das Modul noch nicht geladen ist. Um dieses Problem zu lösen, wurde das Skript **S05modules** zusammen mit `/etc/sysconfig/modules` den LFS-Bootskripten hinzugefügt. Alle in der Datei `modules` eingetragenen Module werden beim Systemstart geladen. Das ermöglicht **udev**, Geräte zu erkennen und die entsprechenden Gerätedateien zu erzeugen.

Bitte beachten Sie, dass manche Treiber sehr viele Gerätedateien erzeugen. Auf langsamen Maschinen kann dieser Vorgang ein paar Sekunden dauern. Daraus folgt, dass manche Geräte nicht sofort nach dem Anschließen verfügbar sind sondern unter Umständen ein paar Sekunden Zeit brauchen.

7.4.3. Der Umgang mit dynamischen bzw. Hotplug-Geräten

Wenn Sie ein Gerät anschließen, wie z. B. einen Universal Serial Bus (USB) MP3-Player, dann erkennt der Kernel dieses neue Gerät und erzeugt ein Hotplug-Ereignis. Wenn der Treiber bereits geladen ist (z. B. weil er bereits in den Kernel einkompiliert ist oder durch **S05modules** geladen wurde), wird **udev** aufgerufen um die nötigen Gerätedateien mit den Daten aus dem `sysfs`-Dateisystem in `/sys` zu erzeugen.

Wenn der Treiber für das neue Gerät zwar als Modul verfügbar, aber zur Zeit noch nicht geladen ist, dann wird Hotplug das entsprechende Treiber-Modul laden und die nötigen Gerätedateien erzeugen. Danach ist das Gerät verfügbar.

7.4.4. Probleme mit dem Erzeugen von Gerätedateien

Es gibt ein paar bekannte Probleme beim automatisierten Erzeugen von Gerätedateien:

1) Ein Kernaltreiber exportiert seine Daten möglicherweise nicht in das `sysfs`-Dateisystem.

Dieses Problem tritt meist auf, wenn ein Treiber nicht aus dem Kernel-Baum sondern von einem Drittanbieter kommt. Udev kann für diese Treiber nicht automatisch eine Gerätedatei erzeugen. Verwenden Sie die Datei `/etc/sysconfig/createfiles` um die nötigen Gerätedateien manuell zu erzeugen. Ziehen Sie die Datei `devices.txt` aus dem Quellbaum des Kernels zu Rate, oder lesen Sie die Dokumentation zu dem Treiber um die passenden hohen und niedrigen Nummern der Gerätedatei herauszufinden.

2) Es wird ein Gerät benötigt, das keine Hardware ist. Auf dieses Problem werden Sie z. B. stoßen, wenn Sie ALSA's (Advanced Linux Sound Architecture) OSS-Kompatibilitätsmodule verwenden. Dieser Gerätetyp kann mit einer dieser zwei Möglichkeiten eingebunden werden:

- Fügen Sie den Modulnamen zu `/etc/sysconfig/modules` hinzu.

- Verwenden Sie eine „install“ Anweisung in `/etc/modules.conf`. Damit wird **modprobe** angewiesen, beim Laden des einen Moduls auch ein anderes zu laden. Beispiel:

```
install snd-pcm modprobe -i snd-pcm ; modprobe \  
    snd-pcm-oss ; true
```

Diese Zeile veranlasst das System, sowohl `snd-pcm` als auch `snd-pcm-oss` zu laden, sobald `snd-pcm` geladen wird.

7.4.5. Nützliche Dokumentation

Weitere hilfreiche Dokumentation finden Sie an den folgenden Stellen:

- A Userspace Implementation of `devfs` http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-1
- udev FAQ <http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ>
- The Linux Kernel Driver Model http://public.planetmirror.com/pub/lca/2003/proceedings/papers/Patrick_Moche

7.5. Einrichten des setclock-Skripts

Das Skript **setclock** liest die Zeit aus der Hardware-Uhr des Computers (auch bekannt als BIOS- oder CMOS-Uhr) und konvertiert sie mit Hilfe von `/etc/localtime` (falls die Hardware Uhr auf GMT gestellt ist) in lokale Zeit. Die Datei `/etc/localtime` enthält die Information, in welcher Zeitzone sich der Anwender befindet. Wenn die Hardware-Uhr auf lokale Zeit eingestellt ist, wird die Zeit nicht konvertiert. Es gibt leider keinen Weg um automatisch herauszufinden, ob die Hardware-Uhr auf GMT gestellt ist oder nicht, deshalb müssen Sie diese Einstellung selber vornehmen.

Falls Sie sich nicht erinnern können, ob die Hardware-Uhr auf GMT eingestellt ist, rufen Sie **hwclock --localtime --show** auf. Dieses Kommando zeigt die Zeit der Hardware-Uhr an. Wenn sie mit der Zeit auf Ihrer Armbanduhr übereinstimmt, dann ist die Hardware-Uhr auf lokale Zeit eingestellt. Wenn die Zeit der Hardware-Uhr abweicht, ist sie wahrscheinlich auf GMT eingestellt. Sie können das überprüfen, indem Sie die entsprechende Anzahl Stunden von der Ausgabe von **hwclock** abziehen bzw. addieren. Wenn Sie zum Beispiel in der Zeitzone MST leben, auch bekannt als GMT-0700, dann addieren Sie sieben Stunden zu der Uhrzeit auf Ihrer Armbanduhr hinzu. Falls es bei Ihnen Sommerzeit gibt, ziehen Sie in den Sommermonaten wieder eine Stunde ab.

Ändern Sie den Wert von UTC zu 0 (Null), wenn Ihre Hardware-Uhr auf lokale Zeit eingestellt ist.

Legen Sie die neue Datei `/etc/sysconfig/clock` mit dem folgenden Kommando an:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# End /etc/sysconfig/clock
EOF
```

Vielleicht möchten Sie sich nun die sehr gute Anleitung unter <http://www.linuxfromscratch.org/hints/downloads/files/time.txt> ansehen. Hier wird erklärt, wie man unter LFS mit der Systemzeit, Zeitzone, UTC und der Variable TZ umgeht.

7.6. Einrichten der Linux Konsole

Dieser Abschnitt behandelt das Bootskript **console**, mit dem die Tastaturbelegung und die Konsoleschriftart eingerichtet wird. Falls Sie nur ASCII-Zeichen verwenden (Britische Pfund oder das Euro-Zeichen sind Beispiele für *nicht*-ASCII-Zeichen) und Ihre Tastatur eine US-Amerikanische ist, dann überspringen Sie diesen Abschnitt. Ohne die Konfigurationsdatei unternimmt dieses Bootskript einfach nichts.

Das Skript benutzt `/etc/sysconfig/console` als Konfigurationsdatei. Entscheiden Sie, welche Tastaturbelegung und Bildschirmschriftarten Sie benutzen werden. Eine vorgefertigte Version von `/etc/sysconfig/console` mit bekannten Einstellungen für verschiedene Länder wurde bereits mit dem Paket LFS-Bootskripte mitinstalliert. Falls Ihr Land bereits unterstützt wird, können Sie den relevanten Abschnitt in der Datei einfach auskommentieren. Wenn Sie unsicher sind, schauen Sie in `/usr/share/kbd` nach gültigen Tastaturbelegungen und Schriftarten. Lesen Sie **loadkeys(1)** und **setfont(8)** und bestimmen Sie die korrekten Parameter zu diesen Programmen. Wenn Sie sich entschieden haben, erstellen Sie mit dem folgenden Kommando die Konfigurationsdatei:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="[Argumente für loadkeys]"
FONT="[Argumente für setfont]"
EOF
```

Beispielsweise sind für deutsche Anwender, die das Euro-Symbol benutzen (erreichbar mit AltGr+E), diese Einstellungen richtig:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="de-latin1-nodeadkeys euro"
FONT="lat9-16 -u lat1"
EOF
```



Anmerkung

Die obige FONT-Zeile ist nur für den Zeichensatz ISO 8859-15 korrekt. Wenn Sie ISO 8859-1 mit dem Pfund-Symbol benutzen, sieht die korrekte FONT-Zeile so aus:

```
FONT="lat1-16"
```

Wenn die Variable `KEYMAP` oder `FONT` nicht eingestellt ist, führt das **console**-Bootskript das zugehörige Programm nicht aus.

Bei manchen Tastaturbelegungen senden die Backspace- und Löschen-Tasten andere Tastencodes als die im Kernel eingebaute voreingestellte Tastaturbelegung. Das bringt bestimmte Anwendungen durcheinander. Zum Beispiel zeigt Emacs seine Hilfe an, anstatt das Zeichen vor dem Cursor zu löschen, wenn Backspace gedrückt wurde. Prüfen Sie, ob die aktuelle Tastaturbelegung davon betroffen ist (funktioniert nur auf i386-Tastaturbelegungen):

```
zgrep '\W14\W' [Pfad/zu/Ihrer/Tastaturtabelle]
```

Wenn keycode 14 Backspace anstatt Delete ist, dann erstellen Sie diesen kleinen Codeschnipsel um das Problem zu lösen:

```
mkdir -pv /etc/kbd && cat > /etc/kbd/bs-sends-del <<"EOF"
        keycode 14 = Delete Delete Delete Delete
    alt keycode 14 = Meta_Delete
altgr alt keycode 14 = Meta_Delete
        keycode 111 = Remove
altgr control keycode 111 = Boot
    control alt keycode 111 = Boot
altgr control alt keycode 111 = Boot
EOF
```

Weisen Sie das Skript **console** an, den Codeschnipsel nach dem Laden der Haupttabelle zu laden:

```
cat >>/etc/sysconfig/console <<"EOF"
KEYMAP_CORRECTIONS="/etc/kbd/bs-sends-del "
EOF
```

Wenn Sie die Tastaturzuweisungstabelle direkt in den Kernel einkompilieren möchten anstatt sie von dem **console**-Skript einstellen zu lassen, dann folgen Sie den Anweisungen in Abschnitt 8.3, „Linux-2.6.11.12“. Auf diese Weise können Sie sicherstellen, dass die Tastatur immer richtig funktioniert, selbst wenn Sie im Wartungsmodus starten (indem Sie den Parameter *init=/bin/sh* an den Kernel übergeben). Im Wartungsmodus wird das **console**-Skript nicht ausgeführt und damit auch die Tastaturbelegung und Bildschirmschriftart nicht korrekt eingestellt. Normalerweise sollte das allerdings sowieso keine Probleme verursachen, weil man im Wartungsmodus üblicherweise nicht auf Sonderzeichen angewiesen ist.

Wenn der Kernel die Tastaturzuweisungstabellen lädt, können Sie die Variable `KEYMAP` in `/etc/sysconfig/console` weglassen. Sie können Sie aber auch problemlos stehen lassen. Das könnte z. B. sinnvoll sein, wenn Sie verschiedene Kernel laufen lassen möchten und nicht in jedem Kernel die Tastaturzuweisungstabelle fest einkompilieren möchten.

7.7. Einrichten des `sysklogd`-Skripts

Das `sysklogd`-Skript ruft `syslogd` mit dem Parameter `-m 0` auf. Dieser Parameter schaltet die periodische Zeitmarke ab, die sonst von `syslogd` alle 20 Minuten in die Protokolldateien geschrieben wird. Falls Sie diese Zeitmarke wieder einschalten möchten, bearbeiten Sie bitte das Skript `sysklogd` und ändern die Option entsprechend. Für weitere Informationen schlagen Sie bitte in `man syslogd` nach.

7.8. Erstellen der Datei `/etc/inputrc`

Die Datei `inputrc` kümmert sich um das Tastaturlayout in bestimmten Situationen. Sie ist die Konfigurationsdatei von Readline — der Bibliothek, die Eingabe-Funktionen für Bash und die meisten anderen Shells zur Verfügung stellt.

Normalerweise braucht man keine benutzerspezifischen Tastaturlayouts, daher erzeugt das folgende Kommando nur die globale Konfigurationsdatei `/etc/inputrc`. Sie wird von jedem Benutzer bzw. der Shell bei der Anmeldung eingelesen und verwendet. Falls Sie später doch eine benutzerspezifische Konfiguration benötigen, können Sie einfach eine Datei mit dem Namen `.inputrc` im Persönlichen Ordner des Benutzers erstellen und dort die angepassten Einstellungen eintragen.

Weitere Informationen zum Anpassen von `inputrc` erhalten Sie mit **info bash** im Abschnitt *Readline Init File*. Eine weitere gute Informationsquelle ist **info readline**.

Sie sehen hier eine generische globale Version der Datei `inputrc`. Darin finden Sie auch erklärende Kommentare zu den verschiedenen Optionen. Beachten Sie bitte, dass sich Kommentare nicht in der gleichen Zeile wie Kommandos befinden dürfen. Erstellen Sie die Datei nun mit dem folgenden Befehl:

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On

# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the
# value contained inside the 1st argument to the
# readline specific functions

"\eOd": backward-word
"\eOc": forward-word

# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert
```

```
# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line

# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# End /etc/inputrc
EOF
```

7.9. Die Startdateien von Bash

Das Shell-Programm `/bin/bash` (im weiteren Verlauf nur „shell“ oder „bash“ genannt) benutzt einige Startdateien zum Einrichten der Benutzerumgebung. Jede Datei hat einen bestimmten Zweck und beeinflusst Login- und Interaktiv-Umgebungen unterschiedlich. Die Bash-Dateien in `/etc` enthalten globale Einstellungen. Wenn eine entsprechende Konfigurations-Datei auch im Persönlichen Ordner des Benutzers existiert, überschreibt sie die globalen Einstellungen.

Nach einem erfolgreichen Login wird mit `/bin/login` eine interaktive Login-Shell gestartet. Dazu wird die Datei `/etc/passwd` eingelesen. Eine interaktive nicht-Login-Shell wird von der Kommandozeile aus gestartet (z. B. `[prompt]$/bin/bash`). Eine nicht-interaktive Shell findet man üblicherweise bei laufenden Shell-Skripten. Sie ist nicht interaktiv, weil Sie ein Skript abarbeitet und zwischen den Kommandos nicht auf Eingaben vom Benutzer wartet.

Weitere Informationen finden Sie mit **info bash** im Abschnitt *Bash Startup Files and Interactive Shells*.

Die Dateien `/etc/profile` und `~/.bash_profile` werden gelesen, wenn die Shell als interaktive Login-Shell aufgerufen wurde.

Die untenstehende Basisversion der Datei `/etc/profile` stellt ein paar notwendige Umgebungsvariablen für NLS-Unterstützung ein. Eine korrekte Einstellung dieser Variablen bewirkt:

- Die Ausgaben von Programmen werden in die Sprache des Anwenders übersetzt
- Korrekte Einordnung von Zeichen als Buchstaben, Zahlen und weiterer Klassen. Die **bash** benötigt diese Einstellungen, um Sonderzeichen in Befehlszeilen in nicht-englischen Locales verarbeiten zu können.
- Korrekte landesspezifische alphabetische Sortierung
- Passende Papiergröße
- Korrekte Formatierung von Währungs-, Zeit- und Datumswerten

Diese Datei setzt auch die Variable `INPUTRC`. Wenn diese Variable gesetzt ist, benutzen Bash und Readline die vorhin erzeugte Datei `/etc/inputrc`.

Ersetzen Sie `[ll]` mit dem zweistelligen Ländercode für die gewünschte Sprache (z. B. „de“) und `[CC]` mit dem zweistelligen Code für das gewünschte Land (z. B. „DE“ oder „AT“). `[charmap]` sollte durch den korrekten Zeichensatz ersetzt werden, z. B. „iso8859-15“.

Mit dem folgenden Kommando erhalten Sie eine Liste aller von Glibc unterstützten Locales:

```
locale -a
```

Locales haben häufig mehrere Synonyme. Z. B. wird „ISO-8859-1“ häufig auch als „iso8859-1“ und „iso88591“ geschrieben. Einige Programme können nicht mit den verschiedenen Synonymen umgehen, daher ist es das sicherste, den korrekten Namen für ein Locale anzugeben. Um den kanonischen Namen für ein Locale herauszufinden, führen Sie das folgende Programm aus, wobei `[locale name]` die Ausgabe von **locale -a** für Ihr bevorzugtes Locale ist (in diesem Beispiel `de_DE.iso88591`).

```
LC_ALL=[locale-Name] locale-Zeichensatz
```

Für das Locale „de_DE.iso88591“ ergibt das obige Kommando:

```
ISO-8859-1
```

Das endgültige Ergebnis lautet also „de_DE.ISO-8859-1“. Bevor Sie diese Locale-Einstellung allerdings in eine der Startdateien der Bash eintragen, sollten Sie sie testen:

```
LC_ALL=[locale-Name] locale country
LC_ALL=[locale-Name] locale language
LC_ALL=[locale-Name] locale charmap
LC_ALL=[locale-Name] locale int_curr_symbol
LC_ALL=[locale-Name] locale int_prefix
```

Das obige Kommando sollte Ihnen folgende Daten ausgeben: Land und Sprache, den vom Locale benutzten Zeichensatz, die Währung und den internationalen Telefonnummern-Prefix. Falls eines der Kommandos eine Fehlermeldung wie die folgende ausgibt, dann wurde die Locale in Kapitel 6 entweder nicht installiert, oder wird von der Standardinstallation von Glibc nicht unterstützt.

```
locale: Cannot set LC_* to default locale: No such file or directory
```

If this happens, you should either install the desired locale using the **localedef** command, or consider choosing a different locale. Further instructions assume that there are no such error messages from Glibc.

Some packages beyond LFS may also lack support for your chosen locale. One example is the X library (part of the X Window System), which outputs the following error message:

```
Warning: locale not supported by Xlib, locale set to C
```

Sometimes it is possible to fix this by removing the charmap part of the locale specification, as long as that does not change the character map that Glibc associates with the locale (this can be checked by running the **locale charmap** command in both locales). For example, one would have to change "de_DE.ISO-8859-15@euro" to "de_DE@euro" in order to get this locale recognized by Xlib.

Other packages can also function incorrectly (but may not necessarily display any error messages) if the locale name does not meet their expectations. In those cases, investigating how other Linux distributions support your locale might provide some useful information.

Wenn Sie die korrekten Locale-Einstellungen herausgefunden haben, erstellen Sie die Datei `/etc/profile`:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

export LANG=[ll]_[CC].[charmap]
export INPUTRC=/etc/inputrc

# End /etc/profile
EOF
```



Anmerkung

Die Locales „C“ (voreinstellung) und „en_US“ (empfohlen für englischsprachige Amerikaner) sind nicht identisch.

Das Einstellen von Tastaturlayout, Bildschirmschriften und Locales sind die einzigen notwendigen Schritte zur Internationalisierung, um Locales mit einfachen 1-Byte-Kodierungen und links-nach-rechts Schreibweise einzurichten. Komplexere Fälle (inklusive UTF-8-basierter Locales) erfordern weitere Schritte und Patches, weil viele Anwendungen dazu neigen, unter diesen Bedingungen nicht richtig zu funktionieren. Diese Schritte und Patches sind derzeit nicht Teil des LFS-Buches und diese Locales werden auch noch nicht von LFS unterstützt.

7.10. Einrichten des localnet-Skripts

Eine Teilaufgabe des **localnet**-Skripts ist das Einstellen des Hostnamens. Dieser muss in der Datei `/etc/sysconfig/network` festgelegt werden.

Erstellen Sie die Datei `/etc/sysconfig/network` und geben Sie den Hostnamen ein:

```
echo "HOSTNAME=[lfs]" > /etc/sysconfig/network
```

`[lfs]` muss hier durch den Namen für Ihren Computer ersetzt werden. Geben Sie nicht den FQDN (Fully Qualified Domain Name -> Vollständigen Domännennamen) ein. Diesen werden Sie erst später in der Datei `/etc/hosts` eintragen. An dieser Stelle wird nur ein einfacher Rechnername benötigt.

7.11. Erstellen der Datei /etc/hosts

Wenn eine Netzwerkkarte eingerichtet werden soll, müssen Sie eine IP-Adresse, den voll qualifizierten Domänennamen und mögliche Aliasnamen in `/etc/hosts` eintragen. Die Syntax ist:

```
<IP-Adresse> meinhost.meinedomain.org aliasname
```

Solange Ihr Computer nicht offiziell im Internet bekannt ist (d. h. Sie haben eine registrierte Domain und einen gültigen zugewiesenen IP-Block, die meisten haben dies nicht), sollten Sie sicherstellen, dass die IP-Adresse im privaten Adressraum liegt. Gültige Adressräume dafür sind:

```
Klasse Netzwerke
A      10.0.0.0
B      172.16.0.0 bis 172.31.0.255
C      192.168.0.0 bis 192.168.255.255
```

Eine gültige IP-Adresse könnte zum Beispiel 192.168.1.1 sein. Ein gültiger voll qualifizierter Domänenname könnte zum Beispiel `www.linuxfromscratch.org` sein (nicht empfohlen, weil dies ein registrierter Name ist und Ihrem DNS-Server Probleme bereiten könnte).

Sie müssen auch dann einen voll qualifizierten Domänennamen eintragen, wenn Sie gar keine Netzwerkkarte haben. Er wird zur korrekten Funktion einiger Programme benötigt.

Erzeugen Sie `/etc/hosts` mit dem folgenden Kommando:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
[192.168.1.1] [<HOSTNAME>.meinedomain.org] [HOSTNAME]

# End /etc/hosts (network card version)
EOF
```

Natürlich müssen Sie `[192.168.1.1]` und `[<HOSTNAME>.meinedomain.org]` nach Ihrem Belieben ändern (bzw. die IP-Adresse und Hostnamen eintragen, die Sie von Ihrem Netzwerkadministrator bekommen haben, falls Ihr Rechner an ein bestehendes Netzwerk angeschlossen wird).

Wenn Sie keine Netzwerkkarte einrichten, erzeugen Sie `/etc/hosts` mit diesem Kommando:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 [<HOSTNAME>.meinedomain.org] [HOSTNAME] localhost

# End /etc/hosts (no network card version)
EOF
```

7.12. Einrichten des network-Skripts

Diesen Abschnitt müssen Sie nur lesen, wenn Sie eine Netzwerkkarte einrichten möchten.

Wenn Sie keine Netzwerkkarte haben, brauchen Sie höchstwahrscheinlich keine Konfigurationsdateien bezüglich Netzwerkkarten einrichten. In diesem Fall sollten Sie alle symbolischen Links mit Namen `network` aus den Runlevel-Ordnern entfernen (`/etc/rc.d/rc*.d`).

7.12.1. Erstellen der Konfigurationsdateien für Netzwerkgeräte

Welche Netzwerkgeräte von den Skripten gestartet und gestoppt werden, hängt von den Dateien und Ordnern in `/etc/sysconfig/network-devices` ab. Dieser Ordner sollte pro Netzwerkgerät einen Unterordner in der Form `ifconfig.xyz` enthalten, wobei „xyz“ der Name des Netzwerkgerätes ist (zum Beispiel `eth0` oder `eth0:1`).

Das folgende Kommando erzeugt die Beispieldatei `ipv4` für `eth0`:

```
cd /etc/sysconfig/network-devices &&
mkdir -v ifconfig.eth0 &&
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

Natürlich müssen die Werte der Variablen in jeder Datei angepasst werden um mit Ihrer tatsächlichen Systemkonfiguration übereinzustimmen. Wenn die `ONBOOT`-Variable auf „yes“ gesetzt ist, wird das `network`-Skript die Netzwerkkarte beim booten starten. Wenn sie auf irgendeinen anderen Wert gesetzt wird, ignoriert das Skript dieses Gerät und startet es dementsprechend auch nicht.

Der Eintrag `SERVICE` legt fest, wie die IP-Adresse vergeben wird. Die LFS-Bootskripte sind in Bezug auf IP-Adressen-Zuordnung modular aufgebaut. Durch das Erstellen weiterer Dateien in `/etc/sysconfig/network-devices/services` können Sie weitere Zuweisungsmethoden definieren. Das könnten Sie z. B. `tun`, um eine IP-Adresse über DHCP zu beziehen (dies wird im BLFS-Buch beschrieben).

Die Variable `GATEWAY` sollte die IP-Adresse Ihres Standard-Gateways enthalten. Wenn Sie kein Standard-Gateway haben, setzen Sie ein Kommentarzeichen vor die Zeile (`#`).

`PREFIX` muss die Anzahl der verwendeten Bits in der Netzwerkmaske enthalten. Jedes Oktett hat acht Bit. Wenn die Netzwerkmaske `255.255.255.0` lautet, dann werden die ersten drei Oktette benutzt ($3 \times 8 = 24$ Bit) um das Netzwerk zu bezeichnen. `255.255.255.240` benutzt die ersten 28 Bit. Prefixe mit mehr als 24 Bit werden häufig von DSL- und Kabelbasierten Internet-Dienstleistern (ISP) verwendet. In diesem Beispiel (`PREFIX=24`) ist die Netzwerkmaske `255.255.255.0`. Passen Sie sie Ihrem Subnetz entsprechend an.

7.12.2. Erstellen der Datei `/etc/resolv.conf`

Wenn Sie mit dem Internet verbunden sind, brauchen Sie höchstwahrscheinlich DNS-Namensauflösung um Internet Domännennamen zu IP-Adressen aufzulösen. Dies erreichen Sie am einfachsten, indem Sie die IP-Adresse des DNS-Servers (stellt Ihr Internet-Provider oder Netzwerkadministrator bereit) in `/etc/resolv.conf` eintragen. Erzeugen Sie die Datei mit diesem Kommando:

```
cat > /etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf

domain {[Ihr Domänenname]}
nameserver [IP-Adresse des primären Nameservers]
nameserver [IP-Adresse des sekundären Nameservers]

# End /etc/resolv.conf
EOF
```

Natürlich müssen Sie `[IP-Adresse des primären Nameservers]` durch die echte IP-Adresse Ihres primären DNS-Servers ersetzen. Oftmals gibt es mehr als einen Eintrag (offizielle Nameserver müssen aus Fallback-Gründen immer auch einen sekundären DNS-Server haben). Die IP-Adresse könnte auch die eines Routers in Ihrem lokalen Netzwerk sein. Wenn Sie keinen zweiten Nameserver haben oder möchten, entfernen Sie den zweiten `nameserver`-Eintrag.

Kapitel 8. Das LFS-System bootfähig machen

8.1. Einführung

Nun ist es an der Zeit Ihr LFS bootfähig zu machen. In diesem Kapitel erstellen Sie die Datei `fstab`, einen neuen Kernel für Ihr LFS-System und Sie installieren den GRUB Bootloader, damit Sie Ihr LFS-System zum booten auswählen können.

8.2. Erstellen der Datei `/etc/fstab`

Die Datei `/etc/fstab` wird von einigen Programmen benutzt, um festzustellen, wo und in welcher Reihenfolge Partitionen eingehängt werden sollen und welche Dateisysteme geprüft werden müssen. Erstellen Sie nun eine neue Tabelle der Dateisysteme:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type  options  dump  fsck
#              order

/dev/[xxx]    /              [fff]  defaults    1      1
/dev/[yyy]    swap           swap   pri=1        0      0
proc          /proc          proc   defaults    0      0
sysfs         /sys           sysfs  defaults    0      0
devpts        /dev/pts       devpts gid=4,mode=620 0      0
shm           /dev/shm       tmpfs  defaults    0      0
# End /etc/fstab
EOF
```

Natürlich müssen Sie `[xxx]`, `[yyy]` und `[fff]` mit den korrekten Werten für Ihr System ersetzen — zum Beispiel `hda2`, `hda5` und `ext2`. Die Details zu den sechs Feldern in dieser Tabelle finden Sie mittels **man 5 fstab**.

Der Mountpunkt `/dev/shm` für das `tmpfs`-Dateisystem wird hier eingefügt um POSIX-konformes shared memory zu gewährleisten. Ihr Kernel muss Unterstützung dafür haben damit das funktioniert — mehr darüber finden Sie im nächsten Abschnitt. Beachten Sie bitte, dass zur Zeit nur wenige Programme POSIX shared memory verwenden. Daher können Sie den Mountpunkt `/dev/shm` als optional betrachten. Mehr Informationen dazu finden Sie in `Documentation/filesystems/tmpfs.txt` im Quellordner Ihrer Kernel-Quellen.

Es gibt noch mehr Zeilen, die Sie vielleicht zu `fstab` hinzufügen wollen. Eine zum Beispiel zum Verwenden von USB-Geräten:

```
usbfs          /proc/bus/usb  usbfs  devgid=14,devmode=0660 0 0
```

Diese Option wird nur funktionieren, wenn „Support for Host-side USB“ und „USB device filesystem“ fest in den Kernel einkompiliert sind (nicht als Modul). Falls „Support for Host-side USB“ als Modul kompiliert wird, muss der Eintrag `usbcore` in `/etc/sysconfig/modules` eingetragen werden.

8.3. Linux-2.6.11.12

Das Paket Linux enthält den Linux-Kernel.

Geschätzte Kompilierzeit: 4.20 SBU

Ungefähr benötigter Festplattenplatz: 181 MB

Die Installation ist abhängig von: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl und Sed

8.3.1. Installation des Kernel

Kompilieren und Installieren des Kernels sind im Grunde nur ein paar Schritte — Konfigurieren, kompilieren und installieren. Falls Sie die hier benutzte Methode nicht mögen, schauen Sie in der Datei README im Kernel-Quellordner nach Alternativen.

Bereiten Sie den Kompilervorgang mit dem folgenden Kommando vor:

```
make mrproper
```

Hierdurch wird sichergestellt, dass der Kernel-Baum absolut sauber ist. Das Kernel-Team empfiehlt, dieses Kommando vor *jedem* Kompilieren des Kernels auszuführen. Sie sollten sich nicht darauf verlassen, dass die Quellen nach dem Entpacken sauber sind.

Wenn Sie sich in Abschnitt 7.6, „Einrichten der Linux Konsole,“ entschieden haben, die Tastaturzuweisungstabelle in den Kernel einzukompilieren, dann führen Sie jetzt dieses Kommando aus:

```
loadkeys -m /usr/share/kbd/keymaps/[Pfad zur keymap] > \
drivers/char/defkeymap.c
```

Wenn Sie zum Beispiel eine deutsche Tastatur haben, würden Sie `/usr/share/kbd/keymaps/i386/qwertz/de-latin1-nodeadkeys.map.gz` als Pfad zur Keymap benutzen.

Richten Sie den Kernel nun mit der menügeführten Oberfläche ein. In BLFS finden Sie unter <http://www.linuxfromscratch.org/blfs/view/svn/longindex.html#kernel-config-index> einige Informationen zu bestimmten Kernel-Voraussetzungen von Software außerhalb von LFS:

```
make menuconfig
```

make oldconfig könnte in einigen Fällen besser geeignet sein. Schauen Sie in die Datei README um mehr Informationen zu erhalten.

Wenn Sie möchten, können Sie die Kernelkonfiguration überspringen und einfach die Kernel-Konfigurationsdatei `.config` von Ihrem Host-System nach `linux-2.6.11.12` kopieren (falls sie verfügbar ist). Das wird allerdings nicht empfohlen, Sie sind besser dran, wenn Sie alle Konfigurationsmenüs durchsehen und Ihre eigene Kernelkonfiguration einrichten.



Anmerkung

NPTL setzt voraus, dass der Kernel mit GCC 3.x oder neuer kompiliert wird; in diesem Fall 3.4.3.

Wir raten davon ab, den Kernel mit GCC 2.95.x zu kompilieren, da dies Fehler in der Glibc Testsuite verursacht. Leider ist die Kernel-Dokumentation veraltet und empfiehlt immer noch GCC 2.95.3 als bevorzugten Compiler.

Kompilieren Sie das Kernel-Abbild und die Module:

```
make
```

Wenn Sie Kernel-Module verwenden, brauchen Sie wahrscheinlich die Datei `/etc/modules.conf`. Informationen zu Modulen und Kernelkonfiguration im Allgemeinen finden Sie in der Kernel-Dokumentation im Ordner `linux-2.6.11.12/Documentation`. Auch `modules.conf(5)` könnte für Sie von Interesse sein.

Seien Sie beim Lesen anderer Dokumentationen bitte vorsichtig, denn die meisten beziehen sich noch auf die 2.4er Kernel-Serie. Soweit wir wissen, sind Konfigurationsprobleme zu Hotplug und Udev noch nicht dokumentiert. Das Problem ist, dass Udev eine Gerätedatei nur erzeugt, wenn Hotplug oder ein Anwender-Skript ein zugehöriges Modul in den Kernel lädt. Aber nicht alle Module sind von Hotplug automatisch erkennbar. Anweisungen in `/etc/modprobe.conf` wie diese funktionieren nicht:

```
alias char-major-XXX irgendein-Modul
```

Aufgrund der Komplikationen mit Hotplug, Udev und Modulen, empfehlen wir dringend, mit einem vollkommen nicht-modularen Kernel zu beginnen. Insbesondere, wenn Sie das erste mal mit Udev zu tun haben.

Installieren Sie die Module, falls Ihre Kernelkonfiguration solche verwendet:

```
make modules_install
```

Das Kompilieren des Kernel ist nun abgeschlossen, aber einige der erzeugten Dateien befinden sich noch im Quellordner. Um die Installation abzuschließen, müssen Sie noch ein paar Dateien in den Ordner `/boot` kopieren.

Der Pfad zur Kerneldatei variiert, abhängig von der benutzten Plattform auf der Sie arbeiten. Das folgende Kommando geht von einem x86-System aus:

```
cp -v arch/i386/boot/bzImage /boot/lfskernel-2.6.11.12
```

`System.map` ist eine Symboldatei für den Kernel. Sie ordnet Funktions-Einstiegspunkte jeder Funktion in der Kernel-API sowie Adressen der Kernel-Datenstrukturen zu. Geben Sie das folgende Kommando ein, um die Datei zu installieren:

```
cp -v System.map /boot/System.map-2.6.11.12
```

`.config` ist die Kernel-Konfigurationsdatei, die durch das obige Kommando `make menuconfig` erzeugt wurde. Sie enthält alle Konfigurationsoptionen für den soeben kompilierten Kernel. Es ist sinnvoll, diese Datei aufzubewahren:

```
cp -v .config /boot/config-2.6.11.12
```

Beachten Sie bitte, dass die Dateien im Kernel-Quellordner nicht *root* gehören. Immer wenn Sie ein Paket als *root*-Benutzer entpacken (so wie Sie es hier im chroot tun), erhalten die entpackten Dateien die Benutzer- und Gruppen ID desjenigen, der das Archiv erstellt hat. Das ist üblicherweise für normale Pakete kein Problem weil Sie den Quellordner nach der Installation löschen. Aber die Linux-Quellen liegen oft sehr lange auf Ihrem Computer, daher ist die Chance groß, dass ein zukünftiger Benutzer auf Ihrem System die Benutzer-ID erhält, die Ihre Kernel-Quellen derzeit haben, und damit wäre er der Besitzer dieser Dateien und hätte dann auch Schreibrechte darauf.

Wenn Sie die Kernelquellen aufbewahren möchten, sollten Sie **chown -R 0:0** auf den Ordner `linux-2.6.11.12` anwenden. So stellen Sie sicher, dass alle Dateien dem Benutzer *root* gehören.



Warnung

Einige Kernaldokumentationen empfehlen das Erzeugen eines Links von `/usr/src/linux` auf den Ordner mit den Kernelquellen. Dies bezieht sich aber nur auf Kernel vor der 2.6er Serie zu und *darf nicht* in einem LFS-System angewendet werden. Es verursacht Probleme beim Kompilieren von Paketen die Sie vielleicht im Nachhinein noch installieren möchten.

Die Header in Ihrem Systemordner `include` sollten *immer* diejenigen sein, mit denen die Glibc kompiliert wurde (also die Linux-Libc-Header) und dürfen daher bei einem Kernelupgrade *keinesfalls* durch die neuen Kernel-Header ersetzt werden.

8.3.2. Inhalt von Linux

Installierte Dateien: `config-2.6.11.12`, `lfskernel-2.6.11.12` und `System.map-2.6.11.12`

Kurze Beschreibungen

<code>config-2.6.11.12</code>	Enthält alle ausgewählten Konfigurationsoptionen für den Kernel.
<code>lfskernel-2.6.11.12</code>	Dies ist der Kernel, der Motor Ihres GNU/Linux-Systems. Nach dem Einschalten Ihres Rechners ist der Kernel der erste Teil des Betriebssystems, der geladen wird. Er erkennt und initialisiert alle Komponenten Ihrer Computer-Hardware und macht diese Komponenten für die Software verfügbar. Er verwandelt eine einzelne CPU in eine Multitasking-Maschine die unzählige Programme scheinbar zur gleichen Zeit ausführen kann.
<code>System.map-2.6.11.12</code>	Enthält eine Liste von Adressen und Symbolen. Sie ordnet Einstiegspunkte und Adressen aller Funktionen und Datenstrukturen dem entsprechenden Kernel zu.

8.4. Das LFS-System bootfähig machen

Ihr frisches LFS-System ist nun beinahe fertig. Sie müssen nun noch sicherstellen, dass es booten kann. Die untenstehende Anleitung gilt nur für Computer mit IA-32-Architektur, dazu gehören alle handelsüblichen PCs. Informationen zum „boot loading“ auf anderen Architekturen finden Sie in den üblichen Dokumentationsquellen zu diesen Architekturen.

Booten kann ein sehr komplexes Thema sein. Hier erstmal ein paar warnende Worte: Sie sollten mit Ihrem jetzigen Bootloader und den Betriebssystemen, die Sie weiter verwenden wollen, vertraut sein. Halten Sie bitte eine „Notfalldiskette“ bereit, damit Sie Ihren Computer starten können, falls Ihr Computer aus irgendwelchen Gründen unbrauchbar wird (weil er zum Beispiel nicht mehr bootet).

Den Grub Bootloader haben Sie bereits installiert. Jetzt müssen ein paar Grub-Dateien an spezielle Orte auf der Festplatte kopiert werden. Bevor Sie das tun, sollten Sie eine Boot-Diskette mit Grub erstellen, nur für den Fall der Fälle. Legen Sie eine leere Diskette ein und führen Sie dieses Kommando aus:

```
dd if=/boot/grub/stage1 of=/dev/fd0 bs=512 count=1
dd if=/boot/grub/stage2 of=/dev/fd0 bs=512 seek=1
```

Entfernen Sie die Diskette und bewahren Sie sie an einem sicheren Ort auf. Starten Sie nun die **grub**-Shell:

```
grub
```

Grub verwendet zur Benennung von Festplatten und Partitionen ein eigenes Schema der Form (hdn,m) , wobei n die Nummer der Festplatte, und m die Nummer der Partition ist. Beide Werte beginnen bei Null. Das bedeutet, dass zum Beispiel die Partition `hda1` für GRUB $(hd0,0)$ ist, und `hdb2` ist $(hd1,1)$. Anders als Linux, betrachtet GRUB CD-Rom Laufwerke nicht als Festplatte. Wenn Sie also ein CD-Rom Laufwerk auf `hdb` haben und eine zweite Festplatte auf `hdc`, dann ist die zweite Festplatte immernoch $(hd1)$.

Bestimmen Sie mit den obigen Informationen den Namen Ihrer `root`-Partition. Im folgenden Beispiel wird angenommen, dass Ihre `root`-Partition `hda4` ist.

Sagen Sie GRUB zuerst, wo die `stage{1,2}`-Dateien zu finden sind—Sie können die Tabulator-Taste verwenden damit Grub Alternativen anzeigt:

```
root (hd0,3)
```



Warnung

Das nächste Kommando überschreibt Ihren bisherigen Bootloader. Wenn Sie das nicht wollen, führen Sie das Kommando nicht aus. Zum Beispiel wenn Sie einen Bootloader von einem Dritthersteller benutzen möchten um Ihren MBR (Master Boot Record) zu verwalten. In dem Fall würde es Sinn machen, Grub in den „Bootsektor“ Ihrer LFS-Partition zu installieren, das folgende Kommando würde dann lauten: **setup (hd0,3)**.

Weisen Sie GRUB nun an, sich in den MBR von `hda` zu installieren:

```
setup (hd0)
```

Wenn alles in Ordnung ist, wird GRUB nun berichten, dass die nötigen Dateien in `/boot/grub` gefunden wurden. Das ist alles soweit, beenden Sie die **grub**-Shell:

```
quit
```

Nun müssen Sie eine „Menü-Liste“ erstellen. Sie definiert das Bootmenü von Grub:

```
cat > /boot/grub/menu.lst << "EOF"
# Begin /boot/grub/menu.lst

# By default boot the first menu entry.
default 0

# Allow 30 seconds before booting the default.
timeout 30

# Use prettier colors.
color green/black light-green/black

# The first entry is for LFS.
title LFS 6.1.1
root (hd0,3)
kernel /boot/lfskernel-2.6.11.12 root=/dev/hda4
EOF
```

Vielleicht möchten Sie einen weiteren Eintrag für Ihr Host-System vornehmen. Dieser könnte z. B. so aussehen:

```
cat >> /boot/grub/menu.lst << "EOF"
title Red Hat
root (hd0,2)
kernel /boot/kernel-2.6.5 root=/dev/hda3
initrd /boot/initrd-2.6.5
EOF
```

Falls Sie Windows dual-booten möchten, könnte der folgende Eintrag hilfreich sein:

```
cat >> /boot/grub/menu.lst << "EOF"
title Windows
rootnoverify (hd0,0)
chainloader +1
EOF
```

Falls Ihnen **info grub** nicht alle benötigten Informationen gibt, finden Sie mehr dazu auf den GRUB-Webseiten unter <http://www.gnu.org/software/grub/>.

FHS setzt voraus, das GRUB's menu.lst nach /etc/grub/menu.lst verlinkt sein sollte. Um diese Voraussetzung zu erfüllen, führen Sie das folgende Kommando aus:

```
mkdir -v /etc/grub &&
ln -sv /boot/grub/menu.lst /etc/grub
```

Kapitel 9. Ende

9.1. Ende

Herzlichen Glückwunsch! Sie sind fertig mit der Installation Ihres eigenen LFS-Systems. Wir wünschen Ihnen viel Freude mit Ihrem brandneuen selbstgebauten Linux.

Sie sollten nun noch die Datei `/etc/lfs-release` erstellen. Mit ihr ist es für Sie (und für uns, wenn Sie uns bei etwas um Hilfe bitten sollten) einfach, herauszufinden, welche LFS-Version Sie haben. Erstellen Sie die Datei mit diesem Kommando:

```
echo 6.1.1 > /etc/lfs-release
```

9.2. Lassen Sie sich zählen

Sie haben nun das ganze Buch durchgearbeitet. Vielleicht möchten Sie sich jetzt als LFS-Benutzer zählen lassen?! Besuchen Sie <http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi> und registrieren Sie sich als LFS-Benutzer indem Sie Ihren Namen und die Versionsnummer Ihres ersten LFS-Systems dort eintragen.

Lassen Sie uns nun Ihr LFS booten...

9.3. Neustarten des Systems

Nachdem nun sämtliche Software installiert ist, wird es Zeit, den Computer neu zu starten. Sie sollten allerdings ein paar Dinge beachten. Das bisher erstellte System ist absolut minimal und hat höchstwahrscheinlich nicht genügend Funktionen, um ernsthaft damit arbeiten zu können. Während Sie weiterhin in der chroot-Umgebung sind, können Sie Pakete aus dem BLFS-Buch installieren. Das versetzt Sie in eine weitaus bessere Lage nach dem Neustart Ihres Systems. Wenn Sie einen textbasierten Webbrowser wie z. B. Lynx installieren, können Sie das BLFS-Buch in einer virtuellen Konsole lesen und in einer anderen Pakete kompilieren. Mit GPM können Sie auch Kopieren und Einfügen zwischen den Konsolen nutzen. Zusätzlich können Sie auch Pakete wie Dhcpcd oder PPP installieren. Dies ist z. B. dann nützlich, wenn Sie keine statische IP-Adresse nutzen können.

Nachdem dies gesagt ist, können Sie nun in Ihr frisch installiertes System booten. Als erstes verlassen Sie die chroot-Umgebung:

```
logout
```

Hängen Sie die virtuellen Dateisysteme aus:

```
umount -v $LFS/dev/pts  
umount -v $LFS/dev/shm  
umount -v $LFS/dev  
umount -v $LFS/proc  
umount -v $LFS/sys
```

Und hängen Sie das LFS-Dateisystem aus:

```
umount -v $LFS
```

Falls Sie sich zu Beginn für mehrere Partitionen entschieden haben, müssen Sie die anderen Partitionen aushängen, bevor Sie die Hauptpartition aushängen:

```
umount -v $LFS/usr  
umount -v $LFS/home  
umount -v $LFS
```

Jetzt können Sie Ihren Computer neu starten:

```
shutdown -r now
```

Unter der Annahme, dass der GRUB Bootloader wie vorgeschlagen installiert wurde, sollte das Standard-Bootmenü automatisch *LFS 6.1.1* booten.

Nach dem Neustart ist Ihr LFS-System bereit; Sie können es nun benutzen und mit der Installation weiterer Software beginnen.

9.4. Was nun?

Vielen Dank, dass Sie dieses Buch gelesen haben. Wir hoffen, dass Sie es nützlich fanden und viel über die Installation von Linux gelernt haben.

Nachdem Sie nun mit der Installation von LFS fertig sind, fragen Sie sich vielleicht: „Was kommt nun?“. Um diese Frage zu beantworten haben wir eine Reihe von Links für Sie zusammengestellt.

- Pflege und Wartung

Für jede Software werden regelmäßig Sicherheitslücken und Fehler gemeldet. Da ein LFS aus den Quellen kompiliert ist, liegt es an Ihnen, diese Berichte zu verfolgen. Es gibt dazu verschiedene Online-Ressourcen die Sie sich ansehen können:

- Freshmeat.net (<http://freshmeat.net/>)

Freshmeat kann Sie (via E-Mail) über neue Programmversionen informieren.

- CERT (Computer Emergency Response Team)

CERT führt eine Mailingliste die Sicherheitswarnungen zu verschiedenen Betriebssystemen und Anwendungen veröffentlicht. Sie können die Liste unter <http://www.us-cert.gov/cas/signup.html> abonnieren.

- Bugtraq

Die Mailingliste Bugtraq ist eine sog. full-disclosure Mailingliste. Auf ihr werden neu entdeckte Sicherheitsprobleme und zum Teil auch Patches zum Beheben der Fehler veröffentlicht. Sie können die Liste unter <http://www.securityfocus.com/archive> abonnieren.

- Beyond Linux From Scratch

Das Buch „Beyond Linux From Scratch“ befasst sich mit der Installation einer Menge Software, die den Rahmen des LFS-Buches sprengen würde. Das BLFS-Projekt finden Sie unter <http://www.linuxfromscratch.org/blfs/>.

- LFS-Hints

Die LFS-Hints sind eine Sammlung von nützlichen Anleitungen und Tipps, die von Freiwilligen aus der LFS-Gemeinschaft eingereicht wurden. Die Anleitungen sind verfügbar unter <http://www.linuxfromscratch.org/hints/list.html>.

- Mailinglisten

Es gibt einige Mailinglisten, die Sie abonnieren können, wenn Sie mal Hilfe benötigen. Weitere Informationen finden Sie in Kapitel 1 - Mailinglisten.

- Das Linux Documentation Project

Das Ziel des Linux Documentation Project ist es, in allen Fragen zu Linux zusammenzuarbeiten. Das LDP verfügt über jede Menge an HOWTOs, Anleitungen und Man-pages. Sie finden es unter <http://www.tldp.org/>.

Teil IV. Anhänge

Anhang A. Akronyme und Begriffe

ABI	Application Binary Interface
ALFS	Automated Linux From Scratch
ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BIOS	Basic Input/Output System
BLFS	Beyond Linux From Scratch
BSD	Berkeley Software Distribution
chroot	change root
CMOS	Complementary Metal Oxide Semiconductor
COS	Class Of Service
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CVS	Concurrent Versions System
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
EGA	Enhanced Graphics Adapter
ELF	Executable and Linkable Format
EOF	End of File
EQN	equation
EVMS	Enterprise Volume Management System
ext2	second extended file system
FAQ	Frequently Asked Questions
FHS	Filesystem Hierarchy Standard
FIFO	First-In, First Out
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GB	Gibabytes
GCC	GNU Compiler Collection

GID	Group Identifier
GMT	Greenwich Mean Time
GPG	GNU Privacy Guard
HTML	Hypertext Markup Language
IDE	Integrated Drive Electronics
IEEE	Institute of Electrical and Electronic Engineers
IO	Input/Output
IP	Internet Protocol
IPC	Inter-Process Communication
IRC	Internet Relay Chat
ISO	International Organization for Standardization
ISP	Internet Service Provider
KB	Kilobytes
LED	Light Emitting Diode
LFS	Linux From Scratch
LSB	Linux Standards Base
MB	Megabytes
MBR	Master Boot Record
MD5	Message Digest 5
NIC	Network Interface Card
NLS	Native Language Support
NNTP	Network News Transport Protocol
NPTL	Native POSIX Threading Library
OSS	Open Sound System
PCH	Pre-Compiled Headers
PCRE	Perl Compatible Regular Expression
PID	Process Identifier
PLFS	Pure Linux From Scratch
PTY	pseudo terminal
QA	Quality Assurance
QOS	Quality Of Service

RAM	Random Access Memory
RPC	Remote Procedure Call
RTC	Real Time Clock
SBU	Standard Build Unit
SCO	The Santa Cruz Operation
SGR	Select Graphic Rendition
SHA1	Secure-Hash Algorithm 1
SMP	Symmetric Multi-Processor
TLDP	Das Linux Documentation Project
TFTP	Trivial File Transfer Protocol
TLS	Thread-Local Storage
UID	User Identifier
umask	user file-creation mask
USB	Universal Serial Bus
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
VC	Virtual Console
VGA	Video Graphics Array
VT	Virtual Terminal

Anhang B. Danksagungen

Wir möchten uns bei allen nachfolgenden Personen und Organisationen für ihr Mitwirken und die Beiträge zu Linux From Scratch bedanken.

- *Gerard Beekmans* <gerard@linuxfromscratch.org> – Gründer von Linux From Scratch, LFS-Projektbetreuer
- *Matthew Burgess* <matthew@linuxfromscratch.org> – LFS-Projektleiter, Release-Betreuer, Buchautor
- *Archaic* <archaic@linuxfromscratch.org> – LFS Buchautor, HLFS-Projektleiter, BLFS-Buchautor, Projektbetreuer von Hints and Patches
- *Nathan Coulson* <nathan@linuxfromscratch.org> – Betreuer der LFS Bootskripte
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – BLFS-Projektleiter
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – LFS/BLFS/HLFS XML- und XSL-Betreuer
- *Jim Gifford* <jim@linuxfromscratch.org> – LFS Buchautor, Patches-Projekt
- *Jeremy Huntwork* <jhuntwork@linuxfromscratch.org> – ALFS-Betreuer, LFS Live-CD-Betreuer, LFS-Buchautor
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Betreuer der Website-Skripte
- *Ryan Oliver* <ryan@linuxfromscratch.org> – LFS-Toolchain-Betreuer
- *James Robertson* <jwrober@linuxfromscratch.org> – Bugzilla-Betreuer
- *Tushar Teredesai* <tushar@linuxfromscratch.org> – BLFS-Buchautor, Betreuer des Hints und Patches Projekts
- Zahllose weitere Personen aus den verschiedenen LFS- und BLFS-Mailinglisten, die mit Vorschlägen, Tests und Fehlerberichten, Anleitungen und Installationserfahrungen zu diesem Buch beitragen.

Übersetzer

- *Manuel Canales Esparcia* <macana@lfs-es.com> – Spanisches LFS-Übersetzerprojekt
- *Johan Lenglet* <johan@linuxfromscratch.org> – Französisches LFS-Übersetzerprojekt
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Portugiesisches LFS-Übersetzerprojekt
- *Thomas Reitelbach* <tr@erdfunkstelle.de> – Deutsches LFS-Übersetzerprojekt

Betreuer der Softwarespiegel

Nordamerikanische Spiegel

- *Scott Kveton* <scott@osuosl.org> – lfs.oregonstate.edu
- *Mikhail Pastukhov* <miha@xuy.biz> – lfs.130th.net

- *William Astle* <lost@l-w.net> – ca.linuxfromscratch.org
- *Jeremy Polen* <jpolen@rackspace.com> – us2.linuxfromscratch.org
- *Tim Jackson* <tim@idge.net> – linuxfromscratch.idge.net
- *Jeremy Utley* <jeremy@linux-phreak.net> – lfs.linux-phreak.net

Südamerikanische Spiegel

- *Andres Meggiotto* <sysop@mesi.com.ar> – lfs.mesi.com.ar
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info
- *Eduardo B. Fonseca* <ebf@aedsolucoes.com.br> – br.linuxfromscratch.org

Europäische Spiegel

- *Barna Koczka* <barna@siker.hu> – hu.linuxfromscratch.org
- *UK Mirror Service* – linuxfromscratch.mirror.ac.uk
- *Martin Voss* <Martin.Voss@ada.de> – lfs.linux-matrix.net
- *Guido Passet* <guido@primerelay.net> – nl.linuxfromscratch.org
- *Bastiaan Jacques* <baafie@planet.nl> – lfs.pagefault.net
- *Roel Neefs* <lfs-mirror@linuxfromscratch.rave.org> – linuxfromscratch.rave.org
- *Justin Knierim* <justin@jrknierim.de> – www.lfs-matrix.de
- *Stephan Brendel* <stevie@stevie20.de> – lfs.netservice-neuss.de
- *Antonin Sprinzl* <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org
- *Fredrik Danerklint* <fredan-lfs@fredan.org> – se.linuxfromscratch.org
- *Parisian sysadmins* <archive@doc.cs.univ-paris8.fr> – www2.fr.linuxfromscratch.org
- *Alexander Velin* <velin@zadnik.org> – bg.linuxfromscratch.org
- *Dirk Webster* <dirk@securewebservices.co.uk> – lfs.securewebservices.co.uk
- *Thomas Skyt* <thomas@sofagang.dk> – dk.linuxfromscratch.org
- *Simon Nicoll* <sime@dot-sime.com> – uk.linuxfromscratch.org

Asiatische Spiegel

- *Pui Yong* <pyng@spam.averse.net> – sg.linuxfromscratch.org
- *Stuart Harris* <stuart@althalus.me.uk> – lfs.mirror.intermedia.com.sg

Australische Spiegel

- *Jason Andrade* <jason@dstc.edu.au> – au.linuxfromscratch.org

Frühere Projektmitglieder

- *Christine Barczak* <theladyskye@linuxfromscratch.org> – LFS Buchautorin
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Website-Entwickler, Betreuer der FAQ
- Alex Groenewoud – LFS Technischer Autor
- Marc Heerdink
- Mark Hymers
- Seth W. Klein – Betreuer der FAQ
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Wiki-Betreuer
- Simon Perreault
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – LFS NNTP Gateway-Betreuer
- *Alexander Patrakov* <semzx@newmail.ru> – LFS Buchautor
- *Greg Schafer* <gschafer@zip.com.au> – LFS Technischer Autor
- Jesse Tie-Ten-Quee – LFS Technischer Autor
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – LFS Buchautor, Bugzilla-Betreuer, Betreuer der LFS Bootskripte
- *Zack Winkles* <zwinkles@gmail.com> – LFS Buchautor

Ein besonderer Dank gilt all unseren Spendern

- *Dean Benson* <dean@vipersoft.co.uk> für etliche Geldspenden
- *Hagen Herrschaft* <hrx@hrxnet.de> für die Spende eines 2,2 GHz P4-Systems, welches nun unter dem Namen Lorien läuft
- *VA Software* die, im Namen von *Linux.com*, eine VA Linux 420 (ehem. StartX SP2) Workstation gespendet haben
- Mark Stone für die Spende von Belgarath, dem linuxfromscratch.org Server

Stichwortverzeichnis

- Autoconf: 154
- Automake: 157
- Bash: 160
 - Werkzeuge: 76
- Binutils: 106
 - Werkzeuge, Durchlauf 1: 38
 - Werkzeuge, Durchlauf 2: 60
- Bison: 134
 - Werkzeuge: 78
- Bootskripte: 210
 - Anwendung: 212
- Bzip2: 164
 - Werkzeuge: 64
- Coreutils: 113
 - Werkzeuge: 63
- DejaGNU: 54
- Diffutils: 166
 - Werkzeuge: 66
- E2fsprogs: 169
- Expect: 52
- File: 162
- Findutils: 123
 - Werkzeuge: 67
- Flex: 140
 - Werkzeuge: 79
- Gawk: 125
 - Werkzeuge: 62
- GCC: 109
 - Werkzeuge, Durchlauf 1: 40
 - Werkzeuge, Durchlauf 2: 55
- Gettext: 142
 - Werkzeuge: 71
- Glibc: 96
 - Werkzeuge: 43
- Grep: 172
 - Werkzeuge: 69
- Groff: 136
- GRUB: 173
 - Einrichten: 237
- Gzip: 175
 - Werkzeuge: 65
- Hotplug: 177
- Iana-Etc: 122
- Inetutils: 145
- IPRoute2: 147
- Kbd: 167
- Less: 135
- Libtool: 163
- Linux: 233
- Linux-Libc-Header: 94
 - Werkzeuge, Header: 42
- M4: 133
 - Werkzeuge: 77
- Make: 181
 - Werkzeuge: 68
- Man: 179
- Man-pages: 95
- Mktemp: 121
- Module-Init-Tools: 182
- Ncurses: 126
 - Werkzeuge: 72
- Patch: 184
 - Werkzeuge: 73
- Perl: 150
 - Werkzeuge: 81
- Procps: 185
- Psmisc: 187
- Readline: 128
- Sed: 139
 - Werkzeuge: 70
- Shadow: 189
 - Einrichten: 191
- Sysklogd: 194
 - Einrichten: 194
- Sysvinit: 196
 - Einrichten: 197
- Tar: 199
 - Werkzeuge: 74
- Tcl: 50
- Texinfo: 152
 - Werkzeuge: 75
- Udev: 200
 - Anwendung: 214
- Util-linux: 202
 - Werkzeuge: 80
- Vim: 130
- Zlib: 119
- a2p: 150 , 151
- acinstall: 157 , 157
- aclocal: 157 , 157
- aclocal-1.9.5: 157 , 157
- addftinfo: 136 , 136
- addr2line: 106 , 107

afmtodit: 136 , 136
 agetty: 202 , 203
 apropos: 179 , 180
 ar: 106 , 107
 arch: 202 , 203
 as: 106 , 107
 autoconf: 154 , 154
 autoheader: 154 , 154
 autom4te: 154 , 154
 automake: 157 , 157
 automake-1.9.5: 157 , 157
 autopoint: 142 , 142
 autoreconf: 154 , 154
 autoscan: 154 , 154
 autoupdate: 154 , 154
 awk: 125 , 125
 badblocks: 169 , 170
 basename: 113 , 114
 bash: 160 , 161
 bashbug: 160 , 161
 bigram: 123 , 123
 bison: 134 , 134
 blkid: 169 , 170
 blockdev: 202 , 203
 bunzip2: 164 , 165
 bzcat: 164 , 165
 bzcmp: 164 , 165
 bzdiff: 164 , 165
 bzegrep: 164 , 165
 bzfgrep: 164 , 165
 bzgrep: 164 , 165
 bzip2: 164 , 165
 bzip2recover: 164 , 165
 bzless: 164 , 165
 bzmores: 164 , 165
 c++: 109 , 110
 c++filt: 106 , 107
 c2ph: 150 , 151
 cal: 202 , 203
 captinfo: 126 , 127
 cat: 113 , 114
 catchsegv: 96 , 101
 cc: 109 , 110
 cfdisk: 202 , 203
 chage: 189 , 192
 chatr: 169 , 170
 chfn: 189 , 192
 chgrp: 113 , 114
 chkdupexe: 202 , 203
 chmod: 113 , 114
 chown: 113 , 114
 chpasswd: 189 , 192
 chroot: 113 , 114
 chsh: 189 , 192
 chvt: 167 , 167
 cksum: 113 , 114
 clear: 126 , 127
 cmp: 166 , 166
 code: 123 , 123
 col: 202 , 203
 colcrt: 202 , 203
 colrm: 202 , 203
 column: 202 , 203
 comm: 113 , 114
 compile: 157 , 157
 compile_et: 169 , 170
 compress: 175 , 175
 config.charset: 142 , 142
 config.guess: 157 , 157
 config.rpath: 142 , 142
 config.sub: 157 , 158
 cp: 113 , 114
 cpp: 109 , 110
 csplit: 113 , 114
 ctrlaltdel: 202 , 203
 ctstat: 147 , 147
 cut: 113 , 115
 cytune: 202 , 203
 date: 113 , 115
 dd: 113 , 115
 ddate: 202 , 203
 deallocvt: 167 , 167
 debugfs: 169 , 170
 depcomp: 157 , 159
 depmod: 182 , 183
 df: 113 , 116
 diff: 166 , 166
 diff3: 166 , 166
 dir: 113 , 116
 dircolors: 113 , 116
 dirname: 113 , 116
 dmesg: 202 , 203
 dprofpp: 150 , 151
 du: 113 , 116
 dumpe2fs: 169 , 170
 dumpkeys: 167 , 167
 e2fsck: 169 , 170
 e2image: 169 , 170

e2label: 169 , 170
 echo: 113 , 116
 efm_filter.pl: 130 , 132
 efm_perl.pl: 130 , 132
 egrep: 172 , 172
 elisp-comp: 157 , 159
 elvtune: 202 , 203
 en2cxs: 150 , 151
 env: 113 , 116
 envsubst: 142 , 142
 eqn: 136 , 136
 eqn2graph: 136 , 137
 ex: 130 , 132
 expand: 113 , 116
 expect: 52 , 53
 expiry: 189 , 192
 expr: 113 , 116
 factor: 113 , 116
 faillog: 189 , 192
 false: 113 , 116
 fdformat: 202 , 203
 fdisk: 202 , 203
 fgconsole: 167 , 167
 fgrep: 172 , 172
 file: 162 , 162
 find: 123 , 123
 find2perl: 150 , 151
 findfs: 169 , 170
 flex: 140 , 141
 fmt: 113 , 116
 fold: 113 , 116
 frcode: 123 , 123
 free: 185 , 185
 fsck: 169 , 170
 fsck.cramfs: 202 , 204
 fsck.ext2: 169 , 170
 fsck.ext3: 169 , 170
 fsck.minix: 202 , 204
 ftp: 145 , 146
 fuser: 187 , 187
 g++: 109 , 110
 gawk: 125 , 125
 gawk-3.1.4: 125 , 125
 gcc: 109 , 110
 gccbug: 109 , 111
 gcov: 109 , 112
 gencat: 96 , 101
 geqn: 136 , 137
 getconf: 96 , 101
 getent: 96 , 101
 getkeycodes: 167 , 167
 getopt: 202 , 204
 gettext: 142 , 142
 gettextize: 142 , 143
 getunimap: 167 , 167
 gpasswd: 189 , 192
 gprof: 106 , 107
 gcat: 125 , 125
 grep: 172 , 172
 grn: 136 , 137
 grodvi: 136 , 137
 groff: 136 , 137
 groffer: 136 , 137
 grog: 136 , 137
 grolbp: 136 , 137
 grolj4: 136 , 137
 groups: 136 , 137
 grotty: 136 , 137
 groupadd: 189 , 192
 groupdel: 189 , 192
 groupmod: 189 , 192
 groups: 113 , 116
 grpck: 189 , 192
 grpconv: 189 , 192
 grpunconv: 189 , 192
 grub: 173 , 173
 grub-install: 173 , 173
 grub-md5-crypt: 173 , 174
 grub-terminfo: 173 , 174
 gtbl: 136 , 137
 gunzip: 175 , 175
 gzexe: 175 , 176
 gzip: 175 , 176
 h2ph: 150 , 151
 h2xs: 150 , 151
 halt: 196 , 198
 head: 113 , 116
 hexdump: 202 , 204
 hostid: 113 , 116
 hostname: 113 , 116
 hostname: 142 , 143
 hotplug: 177 , 178
 hpftodit: 136 , 137
 hwclock: 202 , 204
 iconv: 96 , 101
 iconvconfig: 96 , 101
 id: 113 , 116
 ifcfg: 147 , 147

ifnames: 154 , 156
ifstat: 147 , 147
igawk: 125 , 125
indxib: 136 , 137
info: 152 , 153
infocmp: 126 , 127
infokey: 152 , 153
infotocap: 126 , 127
init: 196 , 198
insmod: 182 , 183
insmod.static: 182 , 183
install: 113 , 116
install-info: 152 , 153
install-sh: 157 , 159
ip: 147 , 148
ipcrm: 202 , 204
ipcs: 202 , 204
isosize: 202 , 204
join: 113 , 116
kbrate: 167 , 167
kbd_mode: 167 , 167
kill: 185 , 185
killall: 187 , 187
killall5: 196 , 198
klogd: 194 , 195
last: 196 , 198
lastb: 196 , 198
lastlog: 189 , 192
ld: 106 , 107
ldconfig: 96 , 101
ldd: 96 , 101
lddlibc4: 96 , 101
less: 135 , 135
less.sh: 130 , 132
lessecho: 135 , 135
lesskey: 135 , 135
lex: 140 , 141
lfskernel-2.6.11.12: 233 , 236
libnetcfg: 150 , 151
libtool: 163 , 163
libtoolize: 163 , 163
line: 202 , 204
link: 113 , 116
lkbib: 136 , 137
ln: 113 , 116
lnstat: 147 , 148
loadkeys: 167 , 167
loadunimap: 167 , 167
locale: 96 , 101
localedef: 96 , 102
locate: 123 , 123
logger: 202 , 204
login: 189 , 192
logname: 113 , 116
logoutd: 189 , 192
logsave: 169 , 170
look: 202 , 204
lookbib: 136 , 137
losetup: 202 , 204
ls: 113 , 116
lsattr: 169 , 170
lsmod: 182 , 183
m4: 133 , 133
make: 181 , 181
makeinfo: 152 , 153
makewhatis: 179 , 180
man: 179 , 180
man2dvi: 179 , 180
man2html: 179 , 180
mapscrn: 167 , 168
mbchk: 173 , 174
mcookie: 202 , 204
md5sum: 113 , 116
mdate-sh: 157 , 159
mesg: 196 , 198
missing: 157 , 159
mkdir: 113 , 116
mke2fs: 169 , 171
mkfifo: 113 , 116
mkfs: 202 , 204
mkfs.bfs: 202 , 204
mkfs.cramfs: 202 , 204
mkfs.ext2: 169 , 171
mkfs.ext3: 169 , 171
mkfs.minix: 202 , 204
mkinstalldirs: 157 , 159
mklost+found: 169 , 171
mknod: 113 , 117
mkpasswd: 189 , 192
mkswap: 202 , 204
mktemp: 121 , 121
mk_cmds: 169 , 171
mmroff: 136 , 137
modinfo: 182 , 183
modprobe: 182 , 183
more: 202 , 204
mount: 202 , 204
mountpoint: 196 , 198

msgattrib: 142 , 144
 msgcat: 142 , 144
 msgcmp: 142 , 144
 msgcomm: 142 , 144
 msgconv: 142 , 144
 msgen: 142 , 144
 msgexec: 142 , 144
 msgfilter: 142 , 144
 msgfmt: 142 , 144
 msggrep: 142 , 144
 msginit: 142 , 144
 msgmerge: 142 , 144
 msgunfmt: 142 , 144
 msguniq: 142 , 144
 mtrace: 96 , 102
 mv: 113 , 117
 mve.awk: 130 , 132
 namei: 202 , 204
 neqn: 136 , 137
 newgrp: 189 , 192
 newusers: 189 , 192
 ngettext: 142 , 144
 nice: 113 , 117
 nl: 113 , 117
 nm: 106 , 107
 nohup: 113 , 117
 nroff: 136 , 137
 nscd: 96 , 102
 nscd_nischeck: 96 , 102
 nstat: 147 , 148
 objcopy: 106 , 107
 objdump: 106 , 107
 od: 113 , 117
 openvt: 167 , 168
 passwd: 189 , 192
 paste: 113 , 117
 patch: 184 , 184
 pathchk: 113 , 117
 pcprofiledump: 96 , 102
 perl: 150 , 151
 perl5.8.7: 150 , 151
 perlbug: 150 , 151
 perlcc: 150 , 151
 perldoc: 150 , 151
 perlvp: 150 , 151
 pfbtops: 136 , 137
 pg: 202 , 204
 pgawk: 125 , 125
 pgawk-3.1.4: 125 , 125
 pgrep: 185 , 185
 pic: 136 , 137
 pic2graph: 136 , 137
 piconv: 150 , 151
 pidof: 196 , 198
 ping: 145 , 146
 pinky: 113 , 117
 pivot_root: 202 , 204
 pkill: 185 , 185
 pl2pm: 150 , 151
 pltags.pl: 130 , 132
 pmap: 185 , 185
 pod2html: 150 , 151
 pod2latex: 150 , 151
 pod2man: 150 , 151
 pod2text: 150 , 151
 pod2usage: 150 , 151
 podchecker: 150 , 151
 podselect: 150 , 151
 post-grohtml: 136 , 137
 poweroff: 196 , 198
 pr: 113 , 117
 pre-grohtml: 136 , 138
 printenv: 113 , 117
 printf: 113 , 117
 ps: 185 , 185
 psed: 150 , 151
 psfaddtable: 167 , 168
 psfgettable: 167 , 168
 psfstriutable: 167 , 168
 psfxtable: 167 , 168
 pstree: 187 , 188
 pstree.x11: 187 , 188
 pstruct: 150 , 151
 ptx: 113 , 117
 pt_chown: 96 , 102
 pwcat: 125 , 125
 pwck: 189 , 192
 pwconv: 189 , 192
 pwd: 113 , 117
 pwunconv: 189 , 192
 py-compile: 157 , 159
 ramsize: 202 , 204
 ranlib: 106 , 107
 raw: 202 , 204
 rcp: 145 , 146
 rdev: 202 , 204
 readelf: 106 , 107
 readlink: 113 , 117

readprofile: 202 , 204
 reboot: 196 , 198
 ref: 130 , 132
 refer: 136 , 138
 rename: 202 , 204
 renice: 202 , 204
 reset: 126 , 127
 resize2fs: 169 , 171
 resizecons: 167 , 168
 rev: 202 , 204
 rlogin: 145 , 146
 rm: 113 , 117
 rmdir: 113 , 117
 rmmod: 182 , 183
 rmt: 199 , 199
 rootflags: 202 , 204
 routef: 147 , 148
 routel: 147 , 148
 rpcgen: 96 , 102
 rpcinfo: 96 , 102
 rsh: 145 , 146
 rtacct: 147 , 148
 rtmon: 147 , 148
 rtpr: 147 , 148
 rtstat: 147 , 148
 runlevel: 196 , 198
 runttest: 54 , 54
 rview: 130 , 132
 rvim: 130 , 132
 s2p: 150 , 151
 script: 202 , 205
 sdiff: 166 , 166
 sed: 139 , 139
 seq: 113 , 117
 setfdprm: 202 , 205
 setfont: 167 , 168
 setkeycodes: 167 , 168
 settled: 167 , 168
 setlogcons: 167 , 168
 setmetamode: 167 , 168
 setsid: 202 , 205
 setterm: 202 , 205
 setvesablank: 167 , 168
 sfdisk: 202 , 205
 sg: 189 , 192
 sh: 160 , 161
 sha1sum: 113 , 117
 showconsolefont: 167 , 168
 showkey: 167 , 168
 shred: 113 , 117
 shtags.pl: 130 , 132
 shutdown: 196 , 198
 size: 106 , 108
 skill: 185 , 185
 sleep: 113 , 117
 sln: 96 , 102
 snice: 185 , 185
 soelim: 136 , 138
 sort: 113 , 117
 splain: 150 , 151
 split: 113 , 117
 sprof: 96 , 102
 ss: 147 , 148
 stat: 113 , 117
 strings: 106 , 108
 strip: 106 , 108
 stty: 113 , 117
 su: 189 , 192
 sulogin: 196 , 198
 sum: 113 , 117
 swapdev: 202 , 205
 swapoff: 202 , 205
 swapon: 202 , 205
 symlink-tree: 157 , 159
 sync: 113 , 117
 sysctl: 185 , 185
 syslogd: 194 , 195
 tac: 113 , 118
 tack: 126 , 127
 tail: 113 , 118
 talk: 145 , 146
 tar: 199 , 199
 tbl: 136 , 138
 tc: 147 , 148
 tcsh: 50 , 51
 tcsh8.4: 50 , 51
 tcltags: 130 , 132
 tee: 113 , 118
 telinit: 196 , 198
 telnet: 145 , 146
 tempfile: 121 , 121
 test: 113 , 118
 texi2dvi: 152 , 153
 texi2pdf: 152 , 153
 texindex: 152 , 153
 tfmtodit: 136 , 138
 tftp: 145 , 146
 tic: 126 , 127

tload: 185 , 185
toe: 126 , 127
top: 185 , 185
touch: 113 , 118
tput: 126 , 127
tr: 113 , 118
troff: 136 , 138
true: 113 , 118
tset: 126 , 127
tsort: 113 , 118
tty: 113 , 118
tune2fs: 169 , 171
tunelp: 202 , 205
tzselect: 96 , 102
udev: 200 , 200
udev: 200 , 201
udevinfo: 200 , 201
udevsend: 200 , 201
udevstart: 200 , 201
udevtest: 200 , 201
ul: 202 , 205
umount: 202 , 205
uname: 113 , 118
uncompress: 175 , 176
unexpand: 113 , 118
unicode_start: 167 , 168
unicode_stop: 167 , 168
uniq: 113 , 118
unlink: 113 , 118
updatedb: 123 , 124
uptime: 185 , 185
useradd: 189 , 192
userdel: 189 , 192
usermod: 189 , 193
users: 113 , 118
utmpdump: 196 , 198
uuidgen: 169 , 171
vdir: 113 , 118
vi: 130 , 132
vidmode: 202 , 205
view: 130 , 132
vigr: 189 , 193
vim: 130 , 132
vim132: 130 , 132
vim2html.pl: 130 , 132
vimdiff: 130 , 132
vimm: 130 , 132
vimspell.sh: 130 , 132
vimtutor: 130 , 132
vipw: 189 , 193
vmstat: 185 , 186
w: 185 , 186
wall: 196 , 198
watch: 185 , 186
wc: 113 , 118
whatis: 179 , 180
whereis: 202 , 205
who: 113 , 118
whoami: 113 , 118
write: 202 , 205
xargs: 123 , 124
xgettext: 142 , 144
xsubpp: 150 , 151
xtrace: 96 , 102
xxd: 130 , 132
yacc: 134 , 134
yes: 113 , 118
ylwrap: 157 , 159
zcat: 175 , 176
zcmp: 175 , 176
zdiff: 175 , 176
zdump: 96 , 102
zegrep: 175 , 176
zfgrep: 175 , 176
zforce: 175 , 176
zgrep: 175 , 176
zic: 96 , 102
zless: 175 , 176
zmore: 175 , 176
znew: 175 , 176
zsoelim: 136 , 138
ld.so: 96 , 102
libanl: 96 , 102
libasprintf: 142 , 144
libbfd: 106 , 108
libblkid: 169 , 171
libBrokenLocale: 96 , 102
libbsd-compat: 96 , 102
libbz2*: 164 , 165
libc: 96 , 102
libcom_err: 169 , 171
libcrypt: 96 , 102
libcurses: 126 , 127
libdl: 96 , 102
libe2p: 169 , 171
libexpect-5.43: 52 , 53
libext2fs: 169 , 171

libfl.a: 140 , 141
libform: 126 , 127
libg: 96 , 102
libgcc*: 109 , 112
libgettextlib: 142 , 144
libgettextpo: 142 , 144
libgettextsrc: 142 , 144
libhistory: 128 , 129
libiberty: 106 , 108
libieee: 96 , 102
libltdl: 163 , 163
libm: 96 , 102
libmagic: 162 , 162
libmcheck: 96 , 103
libmemusage: 96 , 103
libmenu: 126 , 127
libncurses: 126 , 127
libnsl: 96 , 103
libnss: 96 , 103
libopcodes: 106 , 108
libpanel: 126 , 127
libpcprofile: 96 , 103
libproc: 185 , 186
libpthread: 96 , 103
libreadline: 128 , 129
libresolv: 96 , 103
librpcsvc: 96 , 103
librt: 96 , 103
libSegFault: 96 , 102
libshadow: 189 , 193
libss: 169 , 171
libstdc++: 109 , 112
libsupc++: 109 , 112
libtcl8.4.so: 50 , 51
libthread_db: 96 , 103
libutil: 96 , 103
libuuid: 169 , 171
liby.a: 134 , 134
libz: 119 , 120

/etc/hotplug/*.agent: 177 , 178
/etc/hotplug/*.rc: 177 , 178
checkfs: 210 , 210
cleanfs: 210 , 210
console: 210 , 210
 Einrichten: 219
functions: 210 , 210
halt: 210 , 210
hotplug: 210 , 210

ifdown: 210 , 210
ifup: 210 , 210
localnet: 210 , 210
 /etc/hosts: 228
 Einrichten: 227
mountfs: 210 , 210
mountkernfs: 210 , 210
network: 210 , 210
 /etc/hosts: 228
 Einrichten: 229
rc: 210 , 211
reboot: 210 , 211
sendsignals: 210 , 211
setclock: 210 , 211
 Einrichten: 218
static: 210 , 211
swap: 210 , 211
sysklogd: 210 , 211
 Einrichten: 221
template: 210 , 211
udev: 210 , 211

/boot/config-2.6.11.12: 233 , 236
/boot/System.map-2.6.11.12: 233 , 236
/dev/*: 92
/etc/fstab: 232
/etc/group: 90
/etc/hosts: 228
/etc/hotplug.d: 177 , 178
/etc/hotplug/blacklist: 177 , 178
/etc/hotplug/hotplug.functions: 177 , 178
/etc/hotplug/usb.usermap: 177 , 178
/etc/hotplug/{pci,usb}: 177 , 178
/etc/inittab: 197
/etc/inputrc: 222
/etc/ld.so.conf: 101
/etc/lfs-release: 239
/etc/limits: 189
/etc/localtime: 100
/etc/login.access: 189
/etc/login.defs: 189
/etc/nsswitch.conf: 100
/etc/passwd: 90
/etc/profile: 224
/etc/protocols: 122
/etc/resolv.conf: 230
/etc/services: 122
/etc/syslog.conf: 194
/etc/udev: 200 , 201

/etc/vim: 131
/lib/firmware: 177 , 178
/usr/include/{asm,linux}/*.h: 94 , 94
/var/log/btmp: 90
/var/log/hotplug/events: 177 , 178
/var/log/lastlog: 90
/var/log/wtmp: 90
/var/run/utmp: 90
Man-pages: 95 , 95