Linux From Scratch

Version 6.0

Gerard Beekmans

Linux From Scratch: Version 6.0

von Gerard Beekmans

Inhaltsverzeichnis

Einleitung

1. Vorwort

Meine Abenteuer mit Linux fingen vor sechs Jahren an, als ich meine erste Distribution herunterlud und installierte. Nachdem ich damit eine Weile gearbeitet hatte, stieß ich auf Dinge die ich gerne verbessert sehen wollte. Zum Beispiel mochte ich die Zusammenstellung der Bootskripte und die Voreinstellungen von einigen Programmen nicht. Ich probierte einige alternative Distributionen aus, aber jede hatte ihre Vor- und Nachteile. Schlussendlich wurde mir klar, wenn ich wirklich zufrieden sein wollte, dann musste ich mein eigenes Linux von Grund auf selbst erstellen.

Was genau bedeuted dies nun? Ich beschloss, keinerlei vorkompilierte Pakete, keine CD-Roms und keine Bootdisketten jeglicher Art zum Installieren der grundlegenden Werkzeuge zu verwenden. Ich würde mein laufendes Linux-System benutzen, um mein eigenes angepasstes Linux selber zu entwickeln. Dieses „perfekte“ Linux-System würde die Stärken der verschiedenen Distributionen ohne deren Schwächen vereinen. Zu Beginn war die Umsetzunge dieser Idee recht entmutigend. Aber ich blieb engagiert bei der Sache, ein Linux-System zu erstellen, welches meinen Ansprüchen und Wünschen gerecht wurde, anstatt eine Standard-Distribution zu benutzen, die nicht meinen Wünschen entsprach.

Nachdem ich Probleme mit gegenseitigen Abhängigkeiten und Kompilierfehlern ausgeräumt hatte, erstellte ich ein voll funktionsfähiges Linux-System, das meinen individuellen Wünschen entsprach. Dieser Prozess erlaubte mir dann auch, kompakte Linux-Systeme zu erstellen, die schneller waren und weniger Speicher verbrauchten als herkömmliche Betriebssysteme. Ich nannte dieses System Linux From Scratch, oder einfach kurz LFS.

Nachdem ich meine Erfahrungen mit anderen Mitgliedern der Linux-Gemeinschaft geteilt hatte, stellte sich schnell ein wachsendes Interesse an der Fortsetzung meiner Arbeit mit Linux heraus. Solche selbstgebauten LFS-Systeme entsprechen nicht nur einfach Spezifikationen und Anforderungen von Anwendern, sondern sind auch eine ideale Lernbasis für Programmierer und Systemadministratoren um Ihre Linux-Fähigkeiten zu erweitern. Aus diesem breiten Interesse heraus entstand dann das Projekt Linux From Scratch.

Dieses *Linux From Scratch*-Buch gibt dem Leser das nötige Hintergrundwissen und Anleitungen um ein eigenes Linux-System zu entwerfen und zu erstellen. Dieses Buch hebt das Linux From Scratch Projekt und die Vorteile dieses Systems hervor. Anwender können alle Eigenschaften ihres Systems selber vorgeben, inklusive dem Layout der Ordnerstruktur, Skript-Einstellungen und Sicherheit. Das fertige System wird direkt aus dem Quellkode

kompiliert und der Anwender kann selber entscheiden, wo, warum und wie Programme installiert werden. Dieses Buch gibt Anwendern die Möglichkeit, Linux-Systeme an ihre eigenen Bedürfnisse anzupassen und mehr Kontrolle über das System zu erlangen.

Ich wünsche Ihnen viel Spaß 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. Die Zielgruppe

Es gibt viele Gründe, warum jemand dieses Buch möglicherweise lesen möchte. Der Hauptgrund ist, ein Linux-System direkt aus den Quelltexten erstellen zu wollen. Eine oft gestellte Frage ist: „Warum soll ich mir die große Mühe machen, ein Linux-System von Grund auf zu erstellen, wenn ich einfach ein existierendes Linux herunterladen und installieren kann?“. Das ist natürlich eine berechtigte Frage und gleichzeitig auch der Anstoß für diesen Abschnitt des Buches.

Ein wichtiger Grund für die Existenz von LFS besteht darin, 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 der Lernprozess Sie dazu befähigt, Linux an Ihre eigenen Bedürfnisse 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, ein sehr kompaktes Linux-System erstellen zu können. Wenn Sie eine übliche Linux-Distribution installieren, sind Sie für gewöhnlich gezwungen, viele Programme zu installieren, die Sie höchstwahrscheinlich niemals benutzen werden. Diese liegen dann unnütz auf der Festplatte und verbrauchen Speicherplatz (oder noch schlimmer, CPU-Ressourcen). Es ist leicht, ein LFS-System unter 100 MB zu installieren. Das klingt immer noch zu groß? Einige von uns haben daran gearbeitet, ein sehr kleines Embedded-Linux zu bauen. Wir haben es geschafft, einen Apache-Webserver auf einem Linux From Scratch mit gerade mal 8 Mb belegtem Festplattenspeicher laufen zu lassen. Durch weitere Beschneidungen 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 vergleichen, den man in einer Fast Food Kette kauft—man weiß nie genau was man isst. LFS auf der anderen Seite wäre kein Hamburger, sondern vielmehr das Rezept, wie man einen Hamburger macht. Das ermöglicht es, das Rezept zu überprüfen, ungewollte Zutaten wegzulassen und eigene Zutaten nach Geschmack und Belieben hinzuzufügen. Wenn man dann mit dem Rezept zufrieden ist, kann man es zubereiten. Dies tut man dann so wie man es gerne hätte—braten, backen, tiefgefrieren, grillen oder roh essen, ganz wie man will.

Man könnte noch eine weitere Analogie heranziehen. Vergleichen Sie LFS mit einem Fertighaus. LFS wäre der Grundrissplan vom Haus, aber bauen müssen Sie es schon selbst. Jeder hat die Freiheit, den Plan ganz nach Belieben zu verändern.

Nicht zuletzt ist auch Sicherheit ein Vorteil eines selbstgebauten Linux-Systems. Wer ein Linux-System selber aus den Quellen kompiliert hat, kann sämtliche Quelltexte sichten und alle Sicherheitspatches installieren, die man für wichtig hält. Man muss nicht darauf warten, dass jemand anders Binärpakete zur Behebung von Sicherheitslöchern bereitstellt. Solange man Patches nicht selber überprüft und installiert, gibt es keine Garantie, dass das Binärpaket korrekt kompiliert wurde und dass es auch wirklich das Problem behebt.

Unser Ziel von Linux From Scratch ist, ein vollständiges, lauffähiges und grundsolides System zu erstellen. Wenn Sie nur interessiert, 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, welches 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 einfach Ihr Ziel vor Augen; wenn Sie ein komplettes Linux installieren und nebenbei ein bisschen dazulernen wollen, dann ist dieses Buch eine gute Wahl.

Es gibt einfach zu viele gute Gründe, warum man sein eigenes LFS-System erstellen könnte, um sie hier alle aufzuzählen. Dieses Kapitel ist nur die Spitze des Eisberges. Wenn Sie mit LFS arbeiten und Erfahrungen sammeln, werden Sie selbst schnell feststellen, wieviel Macht in Informationen und Wissen über das Linux-System liegt.

3. Voraussetzungen

In diesem Buch wird davon ausgegangen, dass der Leser ein angemessenes Vorwissen zur Installation von Linux-Software hat. Sie sollten diese HOWTOs lesen, bevor Sie mit der Installation Ihres LFS-Systems beginnen:

- Software-Building-HOWTO *http://www.tldp.org/HOWTO/Software-Building-HOWTO.html*

  Dies ist ein umfangreiches Handbuch zum Erstellen und Installieren „allgemeiner“ UNIX-Softwarepakete unter Linux.

- The Linux Users' Guide *http://espc22.murdoch.edu.au/~stewart/guide/guide.html*

  Dieses Handbuch behandelt die Verwendung ausgewählter Linux-Software.

- The Essential Pre-Reading Hint *http://www.linuxfromscratch.org/hints/downloads/files/essentia*

  Dies ist eine LFS-Anleitung, die speziell für neue Linux-Anwender geschrieben wurde. Es ist hauptsächlich eine Linksammlung sehr guter Informationsquellen zu allen möglichen Themen. Jeder der LFS installieren möchte, sollte zumindest den Großteil der dort behandelten Themen verstehen.

4. Konventionen in diesem Buch

Zum besseren Verständnis gibt es einige typografische Konventionen, die in diesem Buch befolgt werden. Nachfolgend einige Beispiele:

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

Solange nicht anders angegeben, muss Text in dieser Textform exakt so eingegeben werden, wie er hier zu sehen ist. Diese Form wird auch in den erläuternden Abschnitten verwendet, um eindeutig anzugeben, auf welche Kommandos sich der Abschnitt bezieht.

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

Diese Textform (Text mit fester Zeichenbreite) symbolisiert Bildschirmausgaben, üblicherweise als Ergebnis von eingegebenen Befehlen. Ausserdem wird diese Textform für Dateinamen wie z. B. `/etc/ld.so.conf` verwendet.

Hervorhebung

Diese Textform wird für verschiedene Zwecke benutzt, hauptsächlich, um wichtige Details in den Vordergrund zu stellen.

http://www.linuxfromscratch.org/

Diese Textform wird für Links benutzt, sowohl innerhalb des Buches als auch zu externen Seiten wie HOWTOs, Download-Adressen und Webseiten.

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

Solche Textabschnitte werden hauptsächlich verwendet, wenn Konfigurationsdateien erstellt werden. Das erste Kommando erzeugt die Datei `$LFS/etc/group` mit dem Inhalt der nachfolgend eingegeben wird, bis die Zeichenfolge EOF erkannt wird. Normalerweise wird Text in dieser Textform exakt so eingegeben wie er hier zu lesen ist.

`[ERSETZTER TEXT]`

Diese Textform wird zum Darstellen von Text verwendet, der nicht einfach blindlings abgeschrieben oder werden darf.

5. Aufbau

Dieses Buch ist in die nachfolgenden Abschnitte unterteilt.

5.1. Teil I - Einführung

Teil I erläutert einige wichtige Dinge zur Installation und schafft Grundlagen zu allgemeinen Dingen des Buches.

5.2. Teil II - Vorbereitungen zur Installation

Teil II beschreibt, wie der Installationsprozess vorbereitet wird—Anlegen einer Partition, Herunterladen der Pakete und Kompilieren der temporären Werkzeuge.

5.3. Teil III - Installation des LFS-Systems

Teil III führt Sie durch die eigentliche Installation von LFS—Kompilieren und Installieren aller Pakete Schritt für Schritt, 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.

Teil I. Einführung

Kapitel 1. Einführung

1.1. Vorgehen zum Installieren eines LFS-Systems

Sie werden Ihr LFS-System mit Hilfe einer bereits laufenden Linux-Distribution (wie z. B. Debian, Mandrake, Red Hat oder SuSE) installieren. Das bestehende Linux-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 installiert, wenn Sie bei der Installation Ihrer Distribution die Kategorie „Entwicklung“ bei den zu installierenden Programmen ausgewählt haben.

Kapitel 2 in diesem Buch beschreibt das Erstellen einer neuen Linux-Partition und eines Dateisystems, auf dem Ihr neues LFS-System kompiliert und installiert wird. Kapitel 3 erklärt, welche Pakete und Patches heruntergeladen und auf dem Dateisystem gespeichert werden müssen. Kapitel 4 behandelt dann das Einrichten einer funktionsfähigen Arbeitsumgebung für die weiteren Schritte. Bitte lesen Sie Kapitel 4 aufmerksam durch, dort werden einige wichtige Probleme behandelt, die Ihnen vor der Arbeit mit Kapitel 5 und den nachfolgenden Kapiteln bekannt sein sollten.

Kapitel 5 beschreibt dann die Installation einiger Pakete für die grundlegende Entwicklungsumgebung (im weiteren Verlauf des Buches *Toolchain* genannt), die benötigt wird um dann in Kapitel 6 das endgültige System zu erstellen. Einige dieser Pakete werden benötigt, um rekursive Abhängigkeiten aufzulösen—zum Beispiel benötigen Sie einen Compiler, um einen Compiler zu kompilieren.

Kapitel 5 erklärt auch, wie eine erste Version der Basiswerkzeuge, bestehend aus Binutils und GCC, erzeugt wird. „erste Version“ bedeutet in diesem Zusammenhang, dass diese beiden Kernpakete noch ein zweites Mal installiert werden. Die Programme aus diesen Paketen werden statisch verlinkt, damit sie unabhängig vom Host-System benutzt werden können. Im nächsten Schritt bauen Sie Glibc, die C-Bibliothek. Glibc wird mit den Programmen der im ersten Schritt erstellten Basiswerkzeuge kompiliert. Im dritten Schritt erstellen Sie eine zweite Version der Basiswerkzeuge. Dieses Mal verlinken Sie 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.

Vielleicht sind Sie der Meinung, dass dies „eine ganze Menge Arbeit ist, nur um von der

Host-Distribution unabhängig zu werden“. Nun, eine vollständige Erklärung finden Sie am Anfang von Kapitel 5, inklusive einiger Hinweise auf die Unterschiede zwischen statisch und dynamisch verlinkter Programme.

In Kapitel 6 wird das eigentliche vollständige LFS-System erstellt. Wir benutzen das Programm chroot (change root, wechseln der Wurzel), um eine Shell in einer virtuellen Umgebung zu starten, in der der Basisordner auf die LFS-Partition eingestellt ist. Das ist ähnlich wie Neustarten und Einhängen der LFS-Partition als root-Partition. Der Grund, warum Sie nicht wirklich Neustarten, sondern stattdessen chroot'en, ist, dass das Erstellen eines bootfähigen Systems zusätzliche Arbeit erfordert, die im Moment noch unnötig ist. Der große Vorteil gegenüber dem Neustart ist, dass das „chroot'en“ des Systems die Weiternutzung des Host-Betriebssystems erlaubt, während Sie das LFS-System installieren. Während Sie warten bis das Kompilieren aller Pakete abgeschlossen ist, 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 Boot-Skripte eingerichtet, der Kernel und der Bootloader werden in Kapitel 8 eingerichtet, und Kapitel 9 enthält Verweise auf Seiten, wo Sie Hilfe finden, wenn Sie das Buch zu Ende gelesen haben. Abschließend ist der Computer bereit für einen Neustart mit dem neuen LFS-System.

Dies ist die ganze Vorgehensweise in zusammengefasster Form. Detaillierte Informationen über alle Schritte werden im Einzelnen in den Kapiteln behandelt, während Sie diese durcharbeiten. Machen Sie sich keine Gedanken, falls jetzt noch etwas unklar sein sollte, alle Fragen werden im weiteren Verlauf beantwortet werden.

1.2. Ressourcen

1.2.1. FAQ

Wenn Sie beim Erstellen des LFS-Systems 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/*.

1.2.2. Mailinglisten

Der `linuxfromscratch.org`-Server stellt einige Mailinglisten für die Entwicklung des LFS-Projektes bereit. Unter anderem befinden sich dort auch die Entwickler- und Support-Mailinglisten.

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*.

1.2.3. IRC

Viele Mitglieder der LFS-Gemeinschaft bieten Hilfe auf unserem IRC-Server an. Bevor Sie hier Hilfe suchen, möchten wir Sie bitten, zumindest die LFS-FAQ und die Archive unserer Mailinglisten nach einer Antwort auf Ihre Frage zu durchsuchen. Der IRC-Server ist zu erreichen unter `irc.linuxfromscratch.org` oder `irc.linux-phreak.net`. Der Support-Chatraum heist #LFS-support.

1.2.4. News-Server

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

Der News-Server ist erreichbar unter `news.linuxfromscratch.org`.

1.2.5. Wiki

Weitere Informationen zu Paketen, neueren Versionen, Einstellmöglichkeiten, persönliche Erfahrungen und vieles mehr finden Sie in unserem LFS-Wiki unter *http://wiki.linuxfromscratch.org/*. Sie können dort auch eigene Informationen hinzufügen und auf diese Weise anderen Benutzern helfen.

1.2.6. Referenzen

Weitere Informationen zu Paketen und nützliche Tipps finden Sie unter *http://www.linuxfromscratch.org/~matthew/LFS-references.html*.

1.2.7. Softwarespiegel

Das LFS-Projekt hat viele Softwarespiegel über die ganze Welt verteilt, die die Website zur Verfügung stellen und den Download der benötigten Programme vereinfachen. Bitte besuchen Sie *http://www.linuxfromscratch.org/*, um eine Liste der aktuellen Softwarespiegel einzusehen.

1.2.8. Kontakt

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

1.3. 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*.

Wenn das nicht nützt, sind die meisten Leute im Internet Relay Chat (IRC) und auf den Mailinglisten (Abschnitt 1.2, „Ressourcen“) gern bereit, Ihnen zu helfen. Aber um sie bei der Problemdiagnose zu unterstützen, sollten Sie schon in der ersten Hilfsanfrage möglichst alle relevanten Informationen mitsenden.

1.3.1. Dinge, die Sie angeben sollten

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

- Die Version des Buches, das Sie benutzen (in diesem Fall Version 6.0),
- die Host-Distribution und -Versionsnummer, die Sie benutzen, um LFS zu installieren,
- Das Paket 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.

Anmerkung

Beachten Sie: Nur weil Sie möglicherweise von den Anweisungen im Buch abgewichen sind, bedeutet das längst *nicht*, dass wir Ihnen nicht helfen werden. Es ist ein Grundsatz von LFS, die Wahl zu haben. Ihr Hinweis hilft uns lediglich, mögliche Ursachen für Ihr Problem besser erkennen zu können.

1.3.2. Probleme mit configure-Skripten

Wenn beim Durchlaufen der configure-Skripte ein Problem auftritt, schauen Sie erst einmal in die Datei `config.log`. Diese Datei enthält viele Fehlermeldungen, die auf dem Bildschirm sonst nicht angezeigt werden. Geben Sie diese Fehlermeldungen mit an, wenn Sie

um Hilfe bitten.

1.3.3. Probleme beim Kompilieren

Um Ihnen zu helfen, sind sowohl Bildschirmausgaben als auch die Inhalte verschiedener Dateien nützlich. Die Ausgaben des **./configure**-Skriptes und die des **make**-Befehls können sehr hilfreich sein. Bitte kopieren Sie nicht einfach blindlings die gesamte Ausgabe; auf der anderen Seite sollte es aber auch nicht zu wenig sein. Als Beispiel für sinnvolle Informationen soll Ihnen folgende Bildschirmausgabe 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 Beispielfall kopieren viele leider nur den unteren Teil:

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

Das reicht für uns aber nicht, um Ihnen bei der Fehlerdiagnose helfen zu können, denn es sagt uns nur, *dass* etwas schiefgelaufen ist, aber nicht *was*. Der ganze oben gezeigte Abschnitt sollte angegeben werden, 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 in dem Dokument. So erhöhen Sie Ihre Chance, dass Sie auf Ihre Frage eine Antwort erhalten, mit der Sie auch etwas anfangen können.

1.3.4. Probleme mit Testsuites

Viele Pakete enthalten eine Testsuite. Abhängig von der Wichtigkeit eines Paketes empfehlen wir Ihnen, die Testsuite durchlaufen zu lassen. Manchmal erzeugen die Pakete Fehlermeldungen oder unerwartete Ergebnisse. Falls Sie solchen Problemen begegnen, können Sie im LFS-Wiki unter *http://wiki.linuxfromscratch.org/* nachsehen, ob diese Probleme bereits bekannt sind und untersucht wurden. Wenn das Problem bereits bekannt ist, brauchen Sie sich im Normalfall keine weiteren Sorgen machen.

1.4. Informationen zu der beigelegten CD

Diesem Buch liegt eine CD mit allen für LFS benötigten Quellpaketen bei. Die CD ist bootfähig und enthält eine stabile Arbeitsumgebung zum Erstellen von LFS. Dieses Buch referenziert dieses System als das *Host-System*.

Zusätzlich zu den benötigten Quellen für LFS sind auf dem Host-System der CD noch einige nützliche Werkzeuge installiert:

- Eine HTML-Version dieses Buches
- Das X Window System
- Web-Werkzeuge
  - Wget (zum Herunterladen von Dateien an der Kommandozeile)
  - Lynx (ein Textbasierter Web-Browser)
  - Irssi (ein IRC-Client für die Kommandozeile)
  - Firefox (ein graphischer Web-Browser)
  - Xchat (ein X-Basierter IRC-Client)
- Text-Editoren
  - Vim
  - Nano
- Netzwerk-Werkzeuge
  - SSH-Server und -Client
  - NFS-Server und -Client
  - Smbmount (mount.cifs) zum Einhängen von Windows-Freigaben
  - Subversion
  - Dhcpcd (DHCP-Client)

- Werkzeuge für Dateisysteme
  - Reiserfsprogs
  - Xfsprogs
- nALFS - Ein Werkzeug zum automatisierten Erstellen von LFS

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 ein Dateisystem darauf und hängen sie anschließend ein (mounten).

2.2. Erstellen einer neuen Partition

Um ein neues Linux-System zu erstellen, brauchen Sie Platz in Form einer Festplattenpartition. Wenn Ihr Computer keine freie Partition oder keinen Platz zum Erstellen einer solchen hat, können Sie LFS auch auf der gleichen Partition wie Ihre Host-Distribution installieren.

Anmerkung

Dieses fortgeschrittene Verfahren wird nicht für Ihre erste LFS-Installation empfohlen, aber wenn Sie zu wenig Festplattenplatz haben, hilft Ihnen eventuell dieses Dokument: *http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt*.

Für ein minimales System benötigen Sie eine Partition mit etwa 1,3 GB Platz. Das reicht aus, um die Quellpakete zu speichern und alle Pakete zu installieren. Aber wenn Sie Ihr LFS später als primäres Betriebssystem nutzen wollen, möchten Sie später vermutlich noch weitere Software hinzufügen, und dann brauchen Sie mehr Platz, wahrscheinlich um die 2 bis 3 GB. Das LFS-System selbst benötigt selbstverständlich nicht so viel Speicherplatz. Eine Menge dieses Platzes wird als temporärer Speicherplatz benötigt. Das Kompilieren von Paketen kann eine Menge Festplattenspeicher in Anspruch nehmen, dieser wird nach dem Kompiliervorgang aber wieder freigegeben.

Weil man häufig zu wenig Random-Access-Memory (RAM, Arbeitsspeicher) hat, ist es eine gute Idee, eine kleine Partition als Swap-Partition zu benutzen -- das ist Speicherplatz, den der Kernel verwendet um selten genutzte Daten auszulagern. 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 bereits eine funktionsfähige Swap-Partition haben, müssen Sie nicht noch eine weitere Partition 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 benötigt). Bitte lesen Sie in der Man-Page zu **cfdisk** oder **fdisk** nach, wenn Sie nicht wissen, wie man diese Programme bedient.

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. Die Bezeichnungen werden später für die Datei `/etc/fstab` benötigt.

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 bei den großen Festplatten von heute werden die Journaling-Dateisysteme immer beliebter. An dieser Stelle werden Sie ein ext2-Dateisystem erstellen. Anleitungen für andere Dateisysteme können Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html* finden.

Um ein ext2-Dateisystem auf der LFS-Partition zu erzeugen, führen Sie bitte folgendes Kommando aus:

```
mke2fs /dev/[xxx]
```

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

Wenn Sie eine Swap Partition erstellt haben, müssen Sie diese als Swap-Partition initialisieren (wird auch als formatieren bezeichnet, so wie Sie es oben schon mit **mke2fs** getan haben), indem Sie dieses Kommando ausführen. Wenn Sie eine bereits existierende Swap-Partition verwenden, muss diese nicht initialisiert werden.

```
mkswap /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, möchten Sie auch darauf zugreifen können. Dazu müssen Sie erst einen Mountpunkt wählen und es dann dort einhängen (mounten). In diesem Buch wird angenommen, dass das Dateisystem unter `/mnt/lfs` eingehängt wird, aber im Grunde ist es egal, welchen Ordner Sie sich aussuchen.

Wählen Sie einen Mountpunkt und weisen Sie diesen der Umgebungsvariable `LFS` zu. Führen Sie dazu folgendes Kommando aus:

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

Als nächstes erzeugen Sie den Mountpunkt und hängen das LFS-Dateisystem mit diesen Kommandos ein:

```
mkdir -p $LFS
mount /dev/[xxx] $LFS
```

Ersetzen Sie *xxx* mit der Bezeichnung der LFS-Partition.

Falls Sie sich entschieden haben, mehrere Partitionen für LFS zu verwenden (z. B. eine für `/` und eine andere für `/usr`), dann hängen Sie diese wie folgt ein:

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

Natürlich müssen Sie auch hier wieder *xxx* und *yyy* durch die korrekten Partitions-Bezeichnungen ersetzen.

Stellen Sie sicher, dass die Zugriffsrechte für die neue Partition beim Einhängen nicht zu restriktiv sind (wie zum Beispiel mit den Optionen „nosuid“, „nodev“ oder „noatime“). Rufen Sie **mount** ohne Parameter auf, um zu sehen mit welchen Optionen Ihre LFS-Dateisysteme eingehängt sind. Wenn Sie *nosuid*, *nodev* oder *noatime* sehen, müssen Sie Ihre Partition erneut einhängen.

Jetzt, nachdem Sie Platz zum Arbeiten geschaffen haben, beginnen Sie mit dem Herunterladen der notwendigen Pakete.

Teil II. Vorbereitungen zur Installation

Kapitel 3. Pakete und Patches

3.1. Einführung

Die untenstehende Liste enthält alle Pakete, die Sie für ein minimales Linux-System herunterladen müssen. Die Versionsnummern entsprechen Softwareversionen, von denen *bekannt* ist, dass Sie funktionieren, und das Buch basiert darauf. Wenn Sie wenig Erfahrung mit LFS haben, wird dringend empfohlen, keine neueren Versionen zu probieren. Die angegebenen 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.

Soweit möglich, verweisen alle URLs auf die Projektseite unter *http://www.freshmeat.net/*. Die Freshmeat-Seiten bieten einfachen Zugriff auf die offiziellen Download- und Projektseiten, Mailinglisten, FAQ's, Changelogs und vieles mehr.

Es kann nicht garantiert werden, dass die Download-Ressourcen immer verfügbar sind. Falls sich eine Download-Adresse nach Erscheinen des Buches geändert haben sollte, nutzen Sie bitte Google zum Suchen nach dem entsprechenden Paket (*http://www.google.com*). Sollten Sie auch hier erfolglos sein, 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 Arbeit mit dem gesamten Buch bequemen Zugriff haben. Weiterhin brauchen Sie einen Arbeitsordner, in dem Sie die Quellen entpacken und kompilieren können. Es ist eine gute Vorgehensweise, den Ordner `$LFS/sources` zum Speichern der Quellen und Patches *und* als Arbeitsordner zu benutzen. So ist alles benötigte immer auf der LFS-Partition abgelegt und in allen Arbeitsschritten des Buches verfügbar.

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

```
mkdir $LFS/sources
```

Machen Sie diesen Ordner beschreibbar und sticky. „Sticky“ bedeuetet, wenn mehrere Benutzer in dem Ordner Schreibrechte haben, darf dennoch nur der Besitzer einer Datei diese auch löschen, wenn sie in einem solchen Sticky-Ordner liegt. Das folgende Kommando schaltet Schreib- und Sticky-Modus ein:

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

3.2. Alle Pakete

Laden Sie die folgenden Pakete herunter:

- Autoconf (2.59) - 903 (KB):
  http://freshmeat.net/projects/autoconf/
- Automake (1.9.1) - 681 KB:
  http://freshmeat.net/projects/automake/
- Bash (3.0) - 1.910 KB:
  http://freshmeat.net/projects/gnubash/
- Binutils (2.15.91.0.2) - 10.666 KB:
  http://freshmeat.net/projects/binutils/?branch_id=12688
- Bison (1.875a) - 796 KB:
  ftp://ftp.linuxfromscratch.org/pub/lfs/lfs-packages/conglomeration/bison/
- Bzip2 (1.0.2) - 650 KB:
  http://freshmeat.net/projects/bzip2/
- Coreutils (5.2.1) - 3.860 KB:
  http://freshmeat.net/projects/coreutils/
- DejaGNU (1.4.4) - 1.055 KB:
  http://freshmeat.net/projects/dejagnu/
- Diffutils (2.8.1) - 762 KB:
  http://freshmeat.net/projects/diffutils/
- E2fsprogs (1.35) - 3.003 KB:
  http://freshmeat.net/projects/e2fsprogs/
- Expect (5.42.1) - 510 KB:
  http://freshmeat.net/projects/expect/
- File (4.10) - 356 KB:
  http://freshmeat.net/projects/file/

Anmerkung

Anmerkung 1) Wenn Sie das hier lesen ist File (4.10) möglicherweise nicht 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. *ftp://ftp.linuxfromscratch.org/pub/lfs/*.

- Findutils (4.1.20) - 760 KB:
  http://freshmeat.net/projects/findutils/
- Flex (2.5.31) - 372 KB:
  http://freshmeat.net/projects/flex/
- Gawk (3.1.4) - 1.692 KB:
  http://freshmeat.net/projects/gnuawk/
- GCC (3.4.1) - 27.000 KB:
  http://freshmeat.net/projects/gcc/
- Gettext (0.14.1) - 6.397 KB:
  http://freshmeat.net/projects/gettext/
- Glibc (2.3.4-20040701) - 13,101 KB:
  http://freshmeat.net/projects/glibc/

Anmerkung

Anmerkung 2) Die freigegebenen Pakete von Glibc sind für unsere Zwecke nicht neu genug. Daher erstellen Sie bitte ein Tar-Archiv aus dem Concurrent Versions System (CVS) mit den folgenden Kommandos:

```
cvs -z 3 -d \
    :pserver:anoncvs@sources.redhat.com:/cvs/glibc \
    export -d glibc-2.3.4-20040701 \
    -D "2004-07-01 17:30 UTC" libc
sed -i -e "s/stable/2004-07-01/" \
    -e "s/2\.3\.3/2.3.4/" \
    glibc-2.3.4-20040701/version.h
tar jcvf glibc-2.3.4-20040701.tar.bz2 \
    glibc-2.3.4-20040701
```

> Als Alternative hat das LFS-Team ein Tar-Archiv vorbereitet. Dieses kann mittels File Transfer Protokoll (FTP) von den LFS Softwarespiegeln (*http://www.linuxfromscratch.org/lfs/packages.html#http*) unter `/pub/lfs/packages/conglomeration/glibc` heruntergeladen werden. Das Tar-Archiv ist mittels GNU Privacy Guard (GPG) signiert. Es wird dringend empfohlen, über die Signatur die Authentizität zu prüfen. Anweisungen zur Installation von GPG werden in dem Buch Beyond Linux From Scratch (BLFS) unter *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/gnupg.html* gegeben.

- Grep (2.5.1) - 545 KB:
  http://freshmeat.net/projects/grep/
- Groff (1.19.1) - 2.360 KB:
  http://freshmeat.net/projects/groff/
- Grub (0.95) - 902 KB:
  ftp://alpha.gnu.org/pub/gnu/grub/
- Gzip (1.3.5) - 324 KB:
  ftp://alpha.gnu.org/gnu/gzip/
- Iana-Etc (1.01) - 161 KB:
  http://freshmeat.net/projects/iana-etc/
- Inetutils (1.4.2) - 1.019 KB:
  http://freshmeat.net/projects/inetutils/
- IPRoute2 (2.6.8-040823) - 264 KB:
  http://developer.osdl.org/dev/iproute2/download/
- Kbd (1.12) - 617 KB:
  http://freshmeat.net/projects/kbd/
- Less (382) - 259 KB:
  http://freshmeat.net/projects/less/
- LFS-Bootscripts (2.2.2) - 16 KB:
  http://downloads.linuxfromscratch.org/
- Libtool (1.5.8) - 2.602 KB:
  http://freshmeat.net/projects/libtool/

- Linux (2.6.8.1) - 34.793 KB:
  http://freshmeat.net/projects/linux/?branch_id=46339
- Linux-Libc-Headers (2.6.8.1) - 2.602 KB:
  http://ep09.pld-linux.org/~mmazur/linux-libc-headers/
- M4 (1.4.2) - 337 KB:
  http://freshmeat.net/projects/gnum4/
- Make (3.80) - 899 KB:
  http://freshmeat.net/projects/gnumake/
- Man (1.5o) - 223 KB:
  http://freshmeat.net/projects/man/
- Man-pages (1.67) - 1.586 KB:
  http://freshmeat.net/projects/man-pages/
- Mktemp (1.5) - 69 KB:
  http://freshmeat.net/projects/mktemp/
- Module-Init-Tools (3.0) - 118 KB:
  ftp://ftp.kernel.org/pub/linux/utils/kernel/module-init-tools/
- Ncurses (5.4) - 2.019 KB:
  http://freshmeat.net/projects/ncurses/
- Patch (2.5.4) - 182 KB:
  http://freshmeat.net/projects/patch/
- Perl (5.8.5) - 9.373 KB:
  http://freshmeat.net/projects/perl/
- Procps (3.2.3) - 265 KB:
  http://freshmeat.net/projects/procps/
- Psmisc (21.5) - 375 KB:
  http://freshmeat.net/projects/psmisc/
- Readline (5.0) - 940 KB:
  http://freshmeat.net/projects/gnureadline/
- Sed (4.1.2) - 749 KB:
  http://freshmeat.net/projects/sed/

- Shadow (4.0.4.1) - 795 KB:
  http://freshmeat.net/projects/shadow/
- Sysklogd (1.4.1) - 80 KB:
  http://freshmeat.net/projects/sysklogd/
- Sysvinit (2.85) - 91 KB:
  http://freshmeat.net/projects/sysvinit/
- Tar (1.14) - 1.025 KB:
  http://freshmeat.net/projects/tar/
- Tcl (8.4.7) - 3.363 KB:
  http://freshmeat.net/projects/tcltk/
- Texinfo (4.7) - 1.385 KB:
  http://freshmeat.net/projects/texinfo/
- Udev (030) - 374 KB:
  ftp://ftp.kernel.org/pub/linux/utils/kernel/hotplug/
- Udev Rechte-Konfiguration - 2 KB:
  http://downloads.linuxfromscratch.org/udev-config-2.permissions
- Udev Regel-Konfiguration - 1 KB:
  http://downloads.linuxfromscratch.org/udev-config-1.rules
- Util-linux (2.12b) - 1.921 KB:
  http://freshmeat.net/projects/util-linux/
- Vim (6.3) - 3.612 KB:
  http://freshmeat.net/projects/vim/
- Vim (6.3) Sprachdateien (optional) - 1.033 KB:
  http://freshmeat.net/projects/vim/
- Zlib (1.2.1) - 277 KB:
  http://freshmeat.net/projects/zlib/

Gesamtgröße dieser Pakete: 135 MB

3.3. Erforderliche Patches

Neben all den Paketen benötigen Sie auch einige Patches. Diese beheben entweder kleine Fehler, die vom Paket-Betreuer noch endgültig behoben werden, oder beinhalten Modifikationen und Anpassungen an unser LFS. Sie brauchen folgende Patches um ein LFS-System zu erstellen:

- Bash Display Wrap Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/bash-3.0-display_wrap-1.patch
- Coreutils Suppress Uptime, Kill, Su Patch - 16 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/co /
- Coreutils Uname Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/coreutils-5.2.1-uname-2.patch
- Expect Spawn Patch - 6 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/expect-5.42.1-spawn-1.patch
- Flex Brokenness Patch - 8 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/flex-2.5.31-debian_fixes-2.patch
- GCC Linkonce Patch - 12 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-linkonce-1.patch
- GCC No-Fixincludes Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-no_fixincludes-1.patch
- GCC Specs Patch - 11 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-specs-1.patch
- Inetutils Kernel Headers Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/inetutils-1.4.2-kernel_headers- 1.patch
- Inetutils No-Server-Man-Pages Patch - 4 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/in /

- IPRoute2 Disable DB Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/iproute2-2.6.8_040823-remove_db-1.patch
- Man 80-Columns Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/man-1.5o-80cols-1.patch
- Mktemp Tempfile Patch - 3 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/mktemp-1.5-add_tempfile-1.patch
- Perl Libc Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/perl-5.8.5-libc-1.patch
- Readline Display Wrap Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/readline-5.0-display_wrap-1.patch
- Sysklogd Kernel Headers Patch - 3 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysklogd-1.4.1-kernel_headers- 1.patch
- Sysklogd Signal Handling Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysklogd-1.4.1-signal-1.patch
- Sysvinit /proc Title Length Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysvinit-2.85-proclen-1.patch
- Texinfo Segfault Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/texinfo-4.7-segfault-1.patch
- Util-Linux Sfdisk Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/util-linux-2.12b-sfdisk-2.patch
- Zlib Security Patch - 1KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/zlib-1.2.1-security-1.patch

Zusätzlich zu den benötigten Patches gibt es noch zahlreiche weitere optionale Patches, die von der LFS-Gemeinschaft erstellt wurden. Die meisten beheben kleine Probleme oder schalten Funktionen ein, die in der Voreinstellung abgeschaltet sind. Durchsuchen Sie ruhig die Patch-Datenbank unter *http://www.linuxfromscratch.org/patches/* und laden Sie zusätzliche Patche herunter, die Sie benutzen möchten.

Kapitel 4. Letzte Vorbereitungen

4.1. Über $LFS

Die Umgebungsvariable LFS wird im gesamten Verlauf des Buches häufig benutzt. Es ist sehr wichtig, dass diese Variable immer definiert ist. Sie sollte auf den Mountpunkt eingestellt sein, den Sie für Ihre LFS-Partition gewählt haben. Überprüfen Sie nochmals mit dem folgenden Kommando, dass die LFS-Variable korrekt gesetzt ist:

```
echo $LFS
```

Stellen Sie sicher, dass die Ausgabe den Pfad zu Ihrer LFS-Partition anzeigt. Dieser sollte /mnt/lfs sein, wenn Sie unserem Beispiel gefolgt sind. Wenn die Ausgabe falsch ist, können Sie die Variable jederzeit neu setzen:

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

Wenn diese Variable gesetzt ist haben Sie den Vorteil, dass Sie ein Kommando wie z. B. **mkdir $LFS/tools** genau so eingeben können wie Sie es lesen. Die Shell wird „$LFS“ durch „/mnt/lfs“ ersetzen, während sie Ihre Eingabe verarbeitet.

Vergessen Sie nicht, jedesmal den Inhalt von $LFS zu prüfen, wenn Sie Ihre Arbeitsumgebung verlassen und neu betreten (wie z. B. wenn Sie „su“ zu *root* oder einem anderen Benutzer ausführen).

4.2. Erstellen des Ordners $LFS/tools

Alle in Kapitel 5 kompilierten Programme werden unter `$LFS/tools` installiert. Dadurch werden sie von den Programmen getrennt, die in Kapitel 6 installiert werden. Die hier kompilierten Programme sind nur übergangsweise Hilfsmittel und werden nicht Teil des endgültigen LFS-Systems sein. Wenn Sie diese Programme in einem separaten Ordner installieren, können sie später leichter gelöscht werden. Ausserdem wird so sichergestellt, dass die Programme nicht versehentlich in Ihrem produktiven Host-System enden (könnte in Kapitel 5 leicht passieren), was wirklich schlecht wäre.

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

```
mkdir $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 -s $LFS/tools /
```

Anmerkung

Das obige Kommando ist in dieser Form korrekt; der Befehl **ln** hat verschiedene Syntax-Varianten, also überprüfen Sie erst die Manpage, bevor Sie einen vermeintlichen Fehler berichten.

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

4.3. Hinzufügen des LFS-Benutzers

Als *root* eingeloggt können kleinste Fehler ein System beschädigen oder gar zerstören. Deshalb wird empfohlen, dass Sie die Pakete in diesem Kapitel mit Hilfe eines unprivilegierten Benutzers kompilieren. Natürlich können Sie Ihren eigenen Benutzernamen dazu verwenden, aber es ist einfacher, eine saubere Arbeitsumgebung zu erstellen, wenn Sie dazu den Benutzer *lfs* in der ebenfalls neuen Gruppe *lfs* erstellen und diesen während des ganzen Installationsvorgangs benutzen. Bitte führen Sie als *root* dieses Kommando aus, um den neuen Benutzer zu erzeugen:

```
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`
Dieses Option fügt den Benutzer *lfs* der Gruppe *lfs* hinzu.

`-m`
Dies erzeugt den Persönlichen Ordner für *lfs*.

`-k /dev/null`
Dieser Parameter verhindert das Kopieren von Dateien aus einem skeleton-Ordner (Voreinstellung ist `/etc/skel`), indem der Quellpfad dieser Dateien auf das spezielle Null-Gerät eingestellt wird.

`lfs`
Dies ist der Name der erzeugten Gruppe und Benutzer.

Um sich als *lfs* einzuloggen (im Gegensatz zum Wechseln zum Benutzer *lfs* während Sie als *root* angemeldet sind, was für den Benutzer *lfs* kein Passwort erfordert), müssen Sie *lfs* ein Passwort geben:

```
passwd lfs
```

Gewähren Sie *lfs* vollen Zugriff auf den Ordner `$LFS/tools`. Dazu machen Sie *lfs* zum Besitzer des Ordners:

```
chown lfs $LFS/tools
```

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

```
chown lfs $LFS/sources
```

Als nächstes loggen Sie sich bitte als *lfs* ein. Sie können das über eine virtuelle Konsole, über den Display-Manager oder mit dem folgenden Kommando tun:

```
su - lfs
```

Das „-“ weist **su** an, eine Login-Shell anstelle einer Nicht-Login-Shell zu starten. Der Unterschied zwischen den beiden Typen wird in der Bash-Manpage und den Info-Seite erklärt.

4.4. Vorbereiten der Arbeitsumgebung

Um Ihre Arbeitsumgebung für die weiteren Schritte vorzubereiten, erstellen Sie zwei Dateien für die Shell **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 wahrscheinlich 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, ausser der `HOME`, `TERM` und `PS1` Variablen. Daduch 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 aussehen, führt aber zu unserem Ziel, nämlich einer absolut leeren Arbeitsumgebung.

Die neue Instanz der Shell ist eine sog. *Nicht-Login*-Shell; diese liest weder `/etc/profile` noch `.bash_profile` ein. Stattdessen liest sie die Datei `.bashrc`. Erstellen Sie diese Datei 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 von **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 so ein ständiges Durchsuchen aller Ordner zu vermeiden. Jedoch müssen Sie 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 stellt sicher, dass neu erzeugte Dateien nur durch ihren Besitzer beschreibbar sind, aber für alle anderen les- und ausführbar (unter der Annahme, dass der open(2) Systemaufruf Standard-Datei-Modi benutzt werden alle erzeugten Dateien die Rechte 644 und alle erzeugten 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 voranstellen von `/tools/bin` an die Standard-Umgebungsvariable `PATH` werden alle in Kapitel 5 installierten Programme von der Shell direkt nach deren Installation 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 auf unserem System existiert.

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

```
source ~/.bash_profile
```

4.5. Über SBUs

Die meisten Leute möchten vorher wissen, wie lange das Kompilieren und Installieren der einzelnen 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 *Static Binutils Unit* entschieden (Abgekürzt *SBU*).

Das funktioniert so: Das erste Paket, das Sie kompilieren werden, ist das statisch gelinkte Binutils Paket in Kapitel 5. Die Zeit, die Sie zum Kompilieren dieses Pakets benötigen, entspricht einer „Static Binutils Unit“ oder auch „SBU“. Alle anderen Kompilierzeiten werden relativ zu dieser Zeit angegeben.

Um zum Beispiel die statische Version von GCC zu kompilieren werden 4,5 SBUs benötigt. Wenn das Kompilieren der statischen Binutils also 10 Minuten gedauert hat, dann braucht es *ungefähr* 45 Minuten, um die statische Version von GCC zu bauen. Zum Glück sind die meisten Kompilierzeiten kürzer als die der Binutils.

Falls der Compiler auf Ihrem Host-System noch ein GCC 2.x ist, könnten die SBU-Angaben etwas unterdimensioniert sein. Die SBU-Angaben basieren auf dem ersten kompilierten Paket, welches allerdings mit Ihrem alten (System-)GCC kompiliert wurde, während der Rest der Pakete aber mit der neuen Version gebaut wird. GCC-3.4.1 ist dafür bekannt, ca. 30% langsamer zu sein. Ausserdem sind SBUs auf Symmetrik-Multi-Prozessor-Maschinen (SMP) nicht akkurat.

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

Grundsätzlich sind SBUs nicht sehr genau weil sie auf vielen Faktoren basieren, nicht nur der GCC-Version. SBUs sollen Ihnen eine ungefähre Vorstellung davon geben, wieviel Zeit das Installieren eines Paketes benötigt. Die Angaben können allerdings unter Umständen dutzende Minuten abweichen.

4.6. Über die Testsuites

Die meisten Pakete stellen eine Testsuite zur Verfügung. 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 garantiert das natürlich nicht für Fehlerfreiheit.

Bestimmte Tests sind wichtiger als andere. 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 benötigen sehr viel Zeit, vor allem auf langsamer Hardware, werden aber dennoch dringend empfohlen.

Anmerkung

Die Erfahrung hat uns gezeigt, dass man nicht viele Vorteile durch das Durchlaufen der Testsuites in Kapitel 5 hat. 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 nur temporär und werden ohnehin später wieder gelöscht. Es wird empfohlen, 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 ist es, zu wenig Pseudo-Terminals zur Verfügung zu haben (abgekürzt PTY's). Ein typisches Symptom dafür sind ungewöhnlich viele fehlschlagende Tests. Das kann aus vielen verschiedenen Gründen geschehen. Der wahrscheinlichste Grund dafür ist, dass das `devpts` Dateisystem des Host-Systems nicht korrekt aufgesetzt ist. Dies wird später in Kapitel 5 ausführlicher behandelt.

Manchmal produzieren Testsuites eines Pakets falschen Alarm. Sehen Sie im LFS-Wiki unter *http://wiki.linuxfromscratch.org/* nach, um zu prüfen, 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 minimales Linux-System kompilieren und installieren. Das System wird gerade genug Werkzeuge beinhalten, um mit dem Erstellen des endgültigen LFS-Systems in Kapitel 6 beginnen zu können und nur ein wenig mehr komfortabel sein als minimal notwendig.

Das Erstellen dieses minimalen Systems erfolgt in zwei Schritten: Als erstes erzeugen Sie eine brandneue Host-unabhängige Toolchain (Compiler, Assembler, Linker und Bibliotheken und ein paar nützliche Werkzeuge). Dann benutzen Sie diese, um alle weiteren essentiellen Werkzeuge zu kompilieren.

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

Die Anweisungen zum Kompilieren setzen voraus, dass Sie die Shell **Bash** benutzen. Ausserdem wird grundsätzlich vorausgesetzt, dass Sie die Archive bereits als Benutzer *lfs* entpackt haben (das wird gleich erklärt) und mit **cd** bereits in den jeweiligen Ordner mit den Quellen gewechselt haben, bevor Sie die Kompilieranleitung umsetzen.

Einige der Pakete werden vor dem Kompilieren gepatcht, aber nur um ein potentielles Problem zu umgehen. Meist wird ein Patch sowohl in diesem als auch im folgenden Kapitel benötigt, manchmal aber auch nur in einem. Machen Sie sich also 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 wirklich nur um Warnungen -- meistens wegen missbilligter (aber dennoch korrekter) Benutzung von 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.

Solange nicht anders angegeben, sollten Sie die Quell- und Kompilierordner nach dem Installieren des jeweiligen Paketes löschen, zum Beispiel um aufzuräumen oder Platz zu sparen. Das Löschen der Quellen verhindert ausserdem mögliche Fehlkonfigurationen, falls ein Paket später nochmal installiert werden muss. Nur bei drei Paketen müssen Sie die Quell- und Kompilierordner für eine Weile aufbewahren, damit ihr Inhalt später noch verwendet werden kann. Übersehen Sie die entsprechenden Hinweise nicht.

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

```
echo $LFS
```

Die Ausgabe sollte den Pfad zum Mountpunkt Ihrer LFS-Partition anzeigen, normalerweise `/mnt/lfs`, wenn Sie unserem Beispiel gefolgt sind.

5.2. 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 den Ausgaben des vorigen Kommandos hervorgeht, dass der Host-Kernel nicht mit GCC-3.0 oder höher kompiliert wurde, muss er neu kompiliert werden. Das Host-System muss danach neu gestartet werden, damit der neue Kernel auch benutzt wird. Anweisungen zum Kompilieren eines Kernels und zum Einrichten des Bootloaders finden Sie in Kapitel 8.

5.3. Technische Anmerkungen zur Toolchain

Dieser Abschnitt soll einige technische Details zum gesamten Kompilier- und Installationsprozess erläutern. Es ist nicht zwingend erforderlich, dass Sie alles hier sofort verstehen. Das meiste ergibt sich von selbst, wenn Sie die ersten Pakete installiert haben. Scheuen Sie sich nicht, zwischendurch noch einmal in diesem Abschnitt nachzulesen.

Das Ziel von Kapitel 5 ist es, eine gut funktionierende temporäre Arbeitsumgebung zu erschaffen, 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 so eine in sich geschlossene Toolchain erzeugen. Bitte beachten Sie, dass der gesamte Prozess dafür ausgelegt ist, jegliche Risiken für neue Leser zu minimieren und gleichzeitig den Lerneffekt zu maximieren. Kurz gesagt, man könnte auch fortgeschrittenere Techniken einsetzen, um das System zu erstellen.

Wichtig

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

Sie sollten auch den Namen des *dynamischen Linkers* für Ihre Plattform kennen (manchmal auch als *dynamischer Lader* bezeichnet), nicht zu verwechseln mit dem Standard-Linker **ld**, der Bestandteil der Binutils ist. Der dynamische Linker kommt mit Glibc und seine Aufgabe ist es, von einem Programm benötigte gemeinsame 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 Kompilierprozess 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 des Paketes Glibc hervor—Es ist kompiliertechnisch gesehen sehr eigenständig und nicht von Voreinstellungen der Toolchain abhängig.

Nach der Installation von Glibc nehmen Sie noch ein paar Anpassungen vor; damit 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 spec-Datei von **gcc** so, dass sie auf den neuen Dynamischen Linker in `/tools/lib` verweist. Der letzte Schritt ist entscheidend für den gesamten Prozess. 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.

Diese Notwendigkeit, den neuen Linker zu benutzen, ist auch der Grund dafür, dass Sie den Specs-Patch für den zweiten GCC Durchlauf anwenden. Ein Fehler dabei würde dazu führen, dass die GCC-Programme selbst den Linker-Namen des Ordners `/lib` des Host-Systems eingebettet hätten, und das würde unserem Ziel, uns vom Host-System zu trennen, entgegenwirken.

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.

5.3.1. Anmerkungen zum statischen Linken

Fast alle Programme führen neben ihrer eigentlichen Aufgabe noch einige sehr typische, manchmal völlig triviale Dinge aus. Das beinhaltet das Reservieren von Speicher, Durchsuchen von Ordnern, Lesen und Schreiben von Dateien, Verarbeitung von Zeichenketten, Mustersuche, Arithmetik und viele andere Dinge. Anstatt Programme zu zwingen, das "Rad neu zu erfinden", stellt das GNU System alle diese Basisfunktionen in fertigen Bibliotheken zur Verfügung. Die wichtigste dieser Bibliotheken auf jedem Linux-System ist die Glibc.

Es gibt zwei elementare Wege, wie man Funktionen einer Bibliothek in ein Programm einbinden kann—statisch oder dynamisch. Beim statischen Linken eines Programmes wird der Kode der genutzten Funktionen in die ausführbare Datei eingebettet; das resultiert in einem relativ großen und klobigen ausführbaren Programm. Beim dynamischen Linken eines Programmes wird in der ausführbaren Datei nur eine Referenz auf den dynamischen Linker, den Namen der Bibliothek und den Namen der Funktion eingebettet; daraus ergibt sich eine wesentlich kleinere ausführbare Datei. (Ein dritter möglicher Weg besteht darin, die Programmierschnittstelle des dynamischen Linkers zu benutzen. Schauen Sie für weitere Informationen bitte in die Manpage von *dlopen*.)

Dynamisches Linken ist unter Linux die Voreinstellung und hat drei große Vorteile gegenüber dem statischen Linken. Erstens braucht man nur eine Kopie des ausführbaren Kodes, anstelle vieler Kopien des selben Kodes, die in allen möglichen ausführbaren Dateien eingebunden sind -- nebenbei spart das auch Speicherplatz auf der Festplatte. Zweitens: Benutzen mehrere Programme die gleichen Bibliotheksfunktionen gleichzeitig, so wird dennoch nur eine Kopie der Funktion geladen -- das spart Arbeitsspeicher. Drittens: Wenn in einer Bibliotheksfunktion ein Fehler behoben wird oder sie auch einfach nur verbessert/erweitert wird, müssen Sie nur diese eine Bibliothek neu kompilieren und nicht jedes Programm, das diese Funktion benutzt.

Wenn der dynamische Linker so viele Vorteile hat, warum werden die ersten beiden Pakete in diesem Kapitel dann statisch gelinkt? Das hat drei Gründe—historische, den Lerneffekt und technische Hintergründe. Historische Gründe deshalb, weil in früheren LFS-Versionen alle Pakete in diesem Kapitel statisch verlinkt wurden. Lerntechnische Gründe, weil es Sinn macht, den Unterschied zu kennen. Technische, weil Sie durch diesen Schritt noch einen Schritt unabhängiger vom Host-System werden. Das bedeutet, diese Programme können unabhängig vom Host-System eingesetzt werden. Natürlich könnten Sie auch dann noch ein gut funktionierendes LFS-System erstellen, wenn diese Pakete dynamisch gelinkt werden.

5.4. Binutils-2.15.91.0.2 - Durchlauf 1

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

Geschätzte Kompilierzeit: 1.0 SBU
Ungefähr benötigter Festplattenplatz: 194 MB

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

5.4.1. Installieren 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 nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel `CFLAGS` und `CXXFLAGS` - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

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

```
mkdir ../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 dieses Pakets benötigen. Dies können Sie relativ einfach mit folgendem Kommando tun: **time { ./configure ... && ... && ... && make install; }**.

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.15.91.0.2/configure --prefix=/tools \
    --disable-nls
```

Die Bedeutung der configure-Parameter:

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

`--disable-nls`
Deaktiviert Internationalisierung; sie wird für die statischen Programmen nicht benötigt und NLS kann beim statischen Linken Probleme verursachen.

Fahren Sie mit dem Kompilieren des Pakets fort:

```
make configure-host
make LDFLAGS="-all-static"
```

Die Bedeutung der make-Parameter:

`configure-host`
Dies erzwingt die sofortige Konfiguration alle Unterordner. Eine statisch gebaute Version würde ansonsten fehlschlagen. Diese Option wird zur Umgehung dieses Problems benutzt.

`LDFLAGS="-all-static"`
Dies teilt dem Linker mit, dass alle Binutils Programme statisch gelinkt werden sollen. Genaugenommen wird *`-all-static`* zuerst an **libtool** übergeben, welches dann wiederum *`-static`* an den Linker übergibt.

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, weil die Programme aus dem ersten Durchlauf sehr bald durch die aus dem zweiten Durchlauf ersetzt werden.

Installieren Sie das Paket:

```
make install
```

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

```
make -C ld clean
make -C ld LDFLAGS="-all-static" 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 LDFLAGS="-all-static" LIB_PATH=/tools/lib`
Diese Option kompiliert alles im Unterordner `ld` erneut. Die Angabe der Makefile-Variable `LIB_PATH` auf der Kommandozeile überschreibt den Standardwert und zeigt auf den temporären tools-Ordner. Der Wert dieser Variable spezifiziert den Standard-Bibliotheksuchpfad für den Linker. Sie werden später in diesem Kapitel sehen, wie diese Vorbereitung zur Anwendung kommt.

Warnung

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

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

5.5. GCC-3.4.1 - Durchlauf 1

Das Paket GCC enthält die GNU-Compiler-Sammlung, die auch die C- und C++-Compiler beinhaltet.

Geschätzte Kompilierzeit: 4.4 SBU
Ungefähr benötigter Festplattenplatz: 300 MB

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

5.5.1. Installieren von GCC

Entpacken Sie nur den GCC-core Tarball; Sie brauchen zunächst weder den C++-Compiler noch die Testsuite.

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel CFLAGS und CXXFLAGS - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

Die GCC-Dokumentation empfiehlt, GCC nicht im Quellordner sondern in einem gesonderten Ordner zu kompilieren:

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

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-3.4.1/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 Schalters 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 diese Option hier wichtig.

--enable-shared
: Dieser Schalter scheint hier erstmal nicht besonders einleuchtend. Aber durch ihn kompilieren wir sowohl `libgcc_s.so.1` als auch `libgcc_eh.a`, und die Präsenz von `libgcc_eh.a` stellt sicher, dass das configure-Skript für Glibc (das nächste zu kompilierende Paket) korrekte Ergebnisse erzielt. Beachten Sie, dass **gcc** selbst trotzdem statisch gelinkt wird; dies wird durch den Parameter *-static* in `BOOT_LDFLAGS` im nächsten Schritt erreicht.

--enable-languages=c
: Diese Option stellt sicher, dass nur der C-Compiler gebaut wird. Diese Option wird nur benötigt, wenn Sie das komplette GCC-Archiv heruntergeladen und entpackt haben.

Fahren Sie mit dem Kompilieren des Pakets fort:

```
make BOOT_LDFLAGS="-static" bootstrap
```

Die Bedeutung der make-Parameter:

BOOT_LDFLAGS="-static"
: Dies weist GCC an, seine Programme statisch zu verlinken.

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 Kompiliervorgang. Abschließend werden die Ergebnisse des zweiten und dritten Kompiliervorgangs verglichen, um sicherzustellen, dass GCC sich selbst problemlos kompilieren konnte. Das bedeutet normalerweise, dass alles korrekt kompiliert wurde.

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, 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 benutzen das Kommando **cc** anstelle von **gcc**; Das dient dem Zweck, die Programme generisch zu halten, damit sie auf verschiedenen Unix-Systemen verwendbar sind. 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 -s gcc /tools/bin/cc
```

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

5.6. Linux-Libc-Header-2.6.8.1

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

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 22 MB

Linux-Libc-Header ist abhängig von: Coreutils

5.6.1. Installieren 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 Kernel-Header Programmierschnittstelle (API) zu bewahren.

Installieren Sie die Header-Dateien:

```
cp -R include/asm-i386 /tools/include/asm
cp -R 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“

5.7. Linux-2.6.8.1 Header

Das Linux-Kernel-Paket enthält die Kernel-Quellen und die von Glibc verwendeten Header-Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 186 MB

Linux Header sind abhängig von: Coreutils und Make

5.7.1. Installation der Kernel-Header

Da einige Pakete die Kernel Header referenzieren, entpacken Sie nun das Kernelarchiv, konfigurieren es und kopieren die benötigten Dateien an eine Stelle, wo **gcc** sie später finden kann.

Bereiten Sie die Installation der Header 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.

Erstellen Sie die Datei `include/linux/version.h`:

```
make include/linux/version.h
```

Erstellen Sie den plattformspezifischen symbolischen Link `include/asm`:

```
make include/asm
```

Installieren Sie die plattformspezifischen Header-Dateien:

```
mkdir /tools/glibc-kernheaders
cp -HR include/asm /tools/glibc-kernheaders
cp -R include/asm-generic /tools/glibc-kernheaders
```

Installieren Sie die Multiplattform-Headerdateien:

```
cp -R include/linux /tools/glibc-kernheaders
```

Details zu diesem Paket finden Sie in Abschnitt 8.3.2, „Inhalt von Linux“

5.8. Glibc-2.3.4-20040701

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: 800 MB

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

5.8.1. Installieren von Glibc

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel CFLAGS und CXXFLAGS - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

Grundsätzlich gilt; weichen Sie von dem in diesem Buch beschriebenen Weg zum Kompilieren von Glibc ab, dann riskieren Sie die Stabilität Ihres gesamten LFS-Systems.

Die Glibc-Dokumentation empfiehlt, nicht im Quellordner sondern in einem gesonderten Ordner zu kompilieren:

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

Als nächstes bereiten Sie Glibc zum Kompilieren vor:

```
../glibc-2.3.4-20040701/configure --prefix=/tools \
    --disable-profile --enable-add-ons=nptl --with-tls \
    --with-__thread --enable-kernel=2.6.0 \
    --with-binutils=/tools/bin --without-gd --without-cvs \
    --with-headers=/tools/glibc-kernheaders
```

Die Bedeutung der configure-Parameter:

`--disable-profile`
Dadurch werden die Bibliotheken ohne Profiling-Informationen kompiliert. Lassen Sie diese Option weg, wenn Sie mit den Temporären Werkzeugen Profiling betreiben möchten.

`--enable-add-ons=nptl`
Dadurch verwendet Glibc NPTL als die Threading-Bibliothek.

`--with-tls`
Hierdurch wird die Unterstützung für Thread-Local Storage (TLS) eingeschaltet. Es wird zur korrekten Funktion von NPTL benötigt.

`--with-__thread`
Dadurch wird die allgemeine Unterstützung für Threads in Glibc eingeschaltet. Es wird zum korrekten Kompilieren von TLS benötigt.

`--enable-kernel=2.6.0`
Dadurch wird die Glibc mit Unterstützung für Kernel der Serie 2.6.x gebaut.

`--with-binutils=/tools/bin`
Diese Option 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.).

`--without-cvs`
Diese Option soll verhindern, dass ein Makefile eventuell automatische CVS-Downloads durchführt (falls ein CVS-Schnappschuss verwendet wird). Genaugenommen wird diese Option zur Zeit nicht benötigt. Wir setzen sie dennoch ein, um eine Warnung bezüglich des fehlenden Programmes autoconf zu verhindern.

`--with-headers=/tools/glibc-kernheaders`
Dadurch wird Glibc mit den „rohen“ Kenerl-Headern kompiliert. So können alle Funktionen des Kernels erkannt, und die Glibc optimiert werden.

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 des Paketes 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 es problemlos durchläuft.

Kompilieren Sie das Paket:

```
make
```

Der Kompiliervorgang 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
```

Eine Information über die wichtigen Fehler finden Sie in Abschnitt 6.11, „Glibc-2.3.4-20040701“

Die Glibc Testsuite ist sehr stark von einigen Funktionen Ihres Host-Systems abhängig. Glibc-Fehler in diesem Kapitel sind normalerweise nicht so schlimm. 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 Nachricht ist, die Installationsroutine von Glibc wird sich über die fehlende Datei `/tools/etc/ld.so.conf` beschweren. Beheben Sie diese störende Warnung mit:

```
mkdir /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
```

Als Alternative zu dem vorigen Kommando können Sie auch nur die von Ihnen benötigten oder gewünschten Locales installieren. Das erreichen Sie mit dem Kommando **localedef**. Informationen dazu finden Sie in der Datei `INSTALL` in den Quellen zu Glibc. Jedoch gibt es einige Locales, die essentiell für die Tests von weiteren Paketen sind, im einzelnen die *libstdc++* Tests von GCC. Die folgenden Anweisungen anstelle des oben verwendeten Targets *`install-locales`* installieren einen minimalen Satz von Locales, die notwendig sind, um die nachfolgenden Tests erfolgreich durchführen zu können:

```
mkdir -p /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“

5.9. Anpassen der Toolchain

Jetzt, nachdem die temporären C-Bibliotheken installiert sind, wollen wir alle Werkzeuge, die im Rest des Kapitels kompiliert werden, 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 am Schluss des ersten Binutils-Durchlauf 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. Die Folge ist 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 wir später den zweiten Durchlauf der Binutils installieren.

Nun, nachdem der angepasste Linker installiert ist, müssen Sie die Binutils-Ordner löschen.

Als nächstes muss die GCC specs-Datei 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 per 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 5.3, „Technische Anmerkungen zur Toolchain“ nach.

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

```
rm -f /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 die Verknüpfung von `/tools/bin/cc`. Gehen Sie zurück zu Abschnitt 5.5, „GCC-3.4.1 - 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 dummy.c a.out
```

5.10. Tcl-8.4.7

Das Tcl Paket enthält die Tool Command Language.

Geschätzte Kompilierzeit: 0.9 SBU
Ungefähr benötigter Festplattenplatz: 23 MB

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

5.10.1. Installieren 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 allerwichtigsten 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 Greenwhich 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 `tcl8.4.7` *noch nicht entfernen*, weil das nächste Paket die internen Header-Dateien benötigt.

Erstellen Sie einen nötigen symbolischen Link:

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

5.10.2. Inhalt von Tcl

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

5.11. Expect-5.42.1

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

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3.9 MB

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

5.11.1. Installieren von Expect

Spielen Sie erst einen Patch ein; dieser behebt einen Fehler, der ansonsten Fehlalarme beim Durchlaufen der GCC Testsuite verursachen könnte:

```
patch -Np1 -i ../expect-5.42.1-spawn-1.patch
```

Bereiten Sie nun Expect zum Kompilieren vor:

```
./configure --prefix=/tools --with-tcl=/tools/lib --with-x=no
```

Die Bedeutung der configure-Parameter:

`--with-tcl=/tools/lib`
: So stellen Sie sicher, dass das configure-Skript die Tcl-Installation in unserem temporären Ordner findet. Es sollte keine möglicherweise auf dem Host-System installierte Version gefunden werden.

`--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 existieren möglicherweise auf dem Host-System.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie das Kommando **`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.

Sie können nun die Quellordner von Tcl und Expect entfernen.

5.11.2. Inhalt von Expect

Installiertes Programm: expect
Installierte Bibliothek: libexpect-5.42.a

Kurze Beschreibungen

expect	„Spricht“ mit anderen interaktiven Programmen und benutzt dazu ein anpassbares Skript
`libexpect-5.42.a`	Enthält Funktionen, mit denen man Expect als TCL-Erweiterung oder direkt aus C/C++ (ohne TCL) nutzen kann

5.12. DejaGNU-1.4.4

Das Paket DejaGnu enthält ein Grundgerüst zum Testen anderer Programme.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 8.6 MB

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

5.12.1. Installieren von DejaGNU

Bereiten Sie DejaGNU zum Kompilieren vor:

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

Kompilieren und installieren Sie das Paket:

```
make install
```

5.12.2. Inhalt von DejaGNU

Installiertes Programm: runtest

Kurze Beschreibungen

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

5.13. GCC-3.4.1 - Durchlauf 2

Geschätzte Kompilierzeit: 11.0 SBU
Ungefähr benötigter Festplattenplatz: 274 MB

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

5.13.1. Neuinstallation von GCC

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel `CFLAGS` und `CXXFLAGS` - für den Kompiliervorgang 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. Eine Sache 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 zu lassen. Wenn Sie mehr Informationen zum Einrichten von PTYs brauchen, schauen Sie am besten in das LFS-Wiki unter *http://wiki.linuxfromscratch.org/*.

Diesmal kompilieren Sie sowohl den C- als auch den C++-Compiler. Entpacken Sie die GCC core- und g++- Tarballs (und -testsuite, falls Sie die Testsuite durchlaufen lassen möchten). Die Archive entpacken sich in einen einzigen Unterordner namens `gcc-3.4.1/`.

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

```
patch -Np1 -i ../gcc-3.4.1-no_fixincludes-1.patch
patch -Np1 -i ../gcc-3.4.1-specs-1.patch
```

Der erste Patch schaltet das **fixincludes**-Skript von GCC ab. Wir haben das vorher bereits kurz erwähnt; hier wollen wir eine nähere Erklärung dazu geben. 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 System Include-Ordnern 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 Kompiliervorgang gegen die neue Glibc gelinkt werden.

Wichtig

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

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

```
mkdir ../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-3.4.1/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 configure-Optionen:

--enable-clocale=gnu
: Diese Option 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 Kode mit Threads ein.

--enable-__cxa_atexit
: Diese Option 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++
: Diese Option 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 Schalter *-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-3.4.1/contrib/test_summary
```

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

Sie können Ihre Ergebnisse mit denen in der gcc-testresults-Mailingliste veröffentlichten vergleichen, die eine ähnliche System-Konfiguration wie Sie haben. Ein Beispiel wie GCC-3.4.1 auf i686-pc-linux-gnu aussehen sollte, finden Sie unter *http://gcc.gnu.org/ml/gcc-testresults/2004-07/msg00179.html*.

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 5.9, „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“

5.14. Binutils-2.15.91.0.2 - Durchlauf 2

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

Geschätzte Kompilierzeit: 1.5 SBU
Ungefähr benötigter Festplattenplatz: 108 MB

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

5.14.1. Neuinstallation von Binutils

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel CFLAGS und CXXFLAGS - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

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

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.15.91.0.2/configure --prefix=/tools \
    --enable-shared --with-lib-path=/tools/lib
```

Die Bedeutung der neuen configure-Option:

--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 Kompiliervorgang 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“

5.15. Gawk-3.1.4

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

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 17 MB

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

5.15.1. Installieren von Gawk

Bereiten Sie Gawk zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen (nicht unbedingt nötig), führen Sie das Kommando **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

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

5.16. Coreutils-5.2.1

Das Paket Coreutils enthält eine große Anzahl an Shell-Werkzeugen zum Einstellen der grundlegenden Systemeigenschaften.

Geschätzte Kompilierzeit: 0.9 SBU
Ungefähr benötigter Festplattenplatz: 69 MB

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

5.16.1. Installieren 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“

5.17. Bzip2-1.0.2

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. Bei Textdateien wird eine wesentlich bessere Kompressionsrate als mit dem traditionellen Kommando **gzip** erreicht.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.5 MB

Bzip2 ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc und Make

5.17.1. Installieren 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“

5.18. Gzip-1.3.5

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

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.6 MB

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

5.18.1. 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“

5.19. Diffutils-2.8.1

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

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7.5 MB

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

5.19.1. Installieren 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“

5.20. Findutils-4.1.20

Das Paket Findutils enthält Programme zum Auffinden von Dateien, entweder durch rekursive Suche in einer Ordnerstruktur oder über den Zugriff auf eine Datenbank (was häufig schneller ist, aber die Gefahr birgt, dass die Datenbank nicht den aktuellen Zustand wiederspiegelt).

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 7.6 MB

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

5.20.1. Installieren von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

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

5.21. Make-3.80

Das Paket Make enthält Programme zum Kompilieren umfangreicher Pakete.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 8.8 MB

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

5.21.1. Installieren von Make

Bereiten Sie Make zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.48.2, „Inhalt von Make“

5.22. Grep-2.5.1

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 5.8 MB

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

5.22.1. Installieren von Grep

Bereiten Sie Grep zum Kompilieren vor:

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

Die Bedeutung der configure-Parameter:

`--disable-perl-regexp`
Dies stellt sicher, dass **grep** nicht gegen die PCRE-Bibliothek verlinkt wird, die eventuell auf dem Host-System installiert ist (aber dann später in der chroot-Umgebung nicht mehr verfügbar wäre).

`--with-included-regex`
Dies stellt sicher, dass Grep seinen eingebauten Code für Reguläre Ausdrücke benutzt. Ohne diesen würde es den Code von Glibc benutzen, der aber bekannt dafür ist, ein wenig fehlerhaft zu sein.

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

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

5.23. Sed-4.1.2

Das Paket Sed enthält einen Stream-Editor.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 5.2 MB

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

5.23.1. Installieren von Sed

Bereiten Sie Sed zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

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

5.24. Gettext-0.14.1

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

Geschätzte Kompilierzeit: 0.5 SBU
Ungefähr benötigter Festplattenplatz: 55 MB

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

5.24.1. Installieren von Gettext

Bereiten Sie Gettext zum Kompilieren vor:

```
./configure --prefix=/tools --disable-libasprintf \
    --disable-csharp
```

Die Bedeutung der configure-Parameter:

`--disable-libasprintf`
Dieser Schalter 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.

`--disable-csharp`
Das bewirkt, dass Gettext keinen C#-Compiler verwendet, selbst wenn auf dem Host-System ein C#-Compiler vorhanden ist. Das ist erforderlich, weil später in der chroot-Umgebung C# nicht mehr verfügbar sein wird.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**. Die Gettext Testsuite braucht sehr viel Zeit (ca. 7 SBU). Es ist bekannt, dass die Gettext Testsuite 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“

5.25. Ncurses-5.4

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

Geschätzte Kompilierzeit: 0.7 SBU
Ungefähr benötigter Festplattenplatz: 26 MB

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

5.25.1. 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`
Das bewirkt, dass Ncurses ohne Ada-Bindungen erstellt wird, selbst wenn auf dem Host-System ein Ada-Compiler vorhanden ist. Das ist erforderlich, weil später in der chroot-Umgebung Ada nicht mehr verfügbar sein wird.

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

5.26. Patch-2.5.4

Das Paket Patch enthält ein Programm zum Modifizieren von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1.9 MB

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

5.26.1. Installieren 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 6.50.2, „Inhalt von Patch“

5.27. Tar-1.14

Das Paket Tar enthält ein Archivprogramm.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 10 MB

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

5.27.1. Installieren von Tar

Bereiten Sie Tar zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.56.2, „Inhalt von Tar“

5.28. Texinfo-4.7

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

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 16 MB

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

5.28.1. Installieren von Texinfo

Bereiten Sie Texinfo zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

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

5.29. Bash-3.0

Das Paket Bash enthält die Bourne-Again-SHell.

Geschätzte Kompilierzeit: 1.2 SBU
Ungefähr benötigter Festplattenplatz: 27 MB

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

5.29.1. Installieren von Bash

Bereiten Sie Bash zum Kompilieren vor:

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

Die Bedeutung der configure-Option:

`--without-bash-malloc`
Diese Option schaltet Bash's memory allocation (malloc) Funktion ab; sie ist bekannt dafür, Speicherzugriffsfehler zu verursachen. Durch das Abschalten der Funktion, wird Bash anstelle dessen die malloc-Funktion der Glibc benutzen. Diese ist stabiler.

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 -s bash /tools/bin/sh
```

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

5.30. M4-1.4.2

M4 enthält einen Makroprozessor.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3.0 MB

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

5.30.1. Installation von M4

Bereiten Sie M4 zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

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

5.31. Bison-1.875a

Bison erstellt ein Programm, das die Struktur einer Textdatei analysiert.

Geschätzte Kompilierzeit: 0.6 SBU
Ungefähr benötigter Festplattenplatz: 10.6 MB

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

5.31.1. Installation von Bison

Bereiten Sie Bison zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

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

5.32. Flex-2.5.31

Das Programm Flex wird benutzt um Programme zu erzeugen, die Muster in Texten erkennen können.

Geschätzte Kompilierzeit: 0.6 SBU
Ungefähr benötigter Festplattenplatz: 10.6 MB

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

5.32.1. Installation von Flex

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

```
patch -Np1 -i ../flex-2.5.31-debian_fixes-2.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 das Kommando **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

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

5.33. Util-linux-2.12b

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

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 16 MB

Util-linux ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed und Zlib

5.33.1. Installieren von Util-linux

Util-linux verwendet nicht die gerade frisch installierten Header und Bibliotheken im Ordner `/tools`. Das korrigieren Sie durch Anpassen des configure-Skriptes:

```
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
```

Da Sie nur ein paar ausgewählte Werkzeuge aus diesem Paket benötigen, kompilieren Sie auch nur diese:

```
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 6.58.3, „Inhalt von Util-linux“

5.34. Perl-5.8.5

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

Geschätzte Kompilierzeit: 0.8 SBU
Ungefähr benötigter Festplattenplatz: 74 MB

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

5.34.1. Installieren von Perl

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

```
patch -Np1 -i ../perl-5.8.5-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-Option:

`-Dstatic_ext='IO Fcntl POSIX'`
: Damit wird Perl angewiesen, die notwendigsten statischen Erweiterungen zu installieren, die im nächsten Kapitel für die Coreutils benötigt werden.

Kompilieren Sie nur ein paar benötigte Programmteile:

```
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.

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

```
cp perl pod/pod2man /tools/bin
mkdir -p /tools/lib/perl5/5.8.5
cp -R lib/* /tools/lib/perl5/5.8.5
```

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

5.35. Udev-030

Das Paket Udev enthält Programme zum dynamischen Erzeugen von Gerätedateien.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 5.2 MB

Udev ist abhängig von: Coreutils und Make

5.35.1. Installation von Udev

Das Programm **udevstart** hat den Pfad zu **udev** fest einkompiliert, was zu Problemen führt, weil **udev** an einem anderem Ort installiert wurde. Beheben Sie das Problem indem Sie dieses Kommando ausführen:

```
sed -i 's@/sbin/udev@/tools/sbin/udev@g' udevstart.c
```

Stellen Sie ausserdem sicher, dass **udev** den korrekten Pfad zu seinen Konfigurationsdateien kennt:

```
sed -i 's@/etc@/tools/etc@g' etc/udev/udev.conf.in
```

Nun kompilieren Sie Udev:

```
make prefix=/tools etcdir=/tools/etc
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make DESTDIR=/tools udevdir=/dev install
```

Die voreingestellte Konfiguration von Udev ist alles andere als optimal, daher installieren Sie LFS-spezifische Konfigurationsdateien:

```
cp ../udev-config-2.permissions \
    /tools/etc/udev/permissions.d/00-lfs.permissions
cp ../udev-config-1.rules /tools/etc/udev/rules.d/00-lfs.rules
```

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

5.36. Stripping

Die Schritte in diesem Abschnitt sind optional. Wenn Ihre LFS-Partition sehr klein ist, werden Sie froh sein, einige unnötige Dinge loswerden zu können. Die ausführbaren Dateien und Bibliotheken, die Sie bis hierher erstellt haben, enthalten ungefähr 130 MB nicht benötigte Debugging-Symbole. So entfernen Sie diese Symbole:

```
strip --strip-debug /tools/lib/*
strip --strip-unneeded /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-unneeded` 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/{doc,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 um die Fehlerwahrscheinlichkeit zu minimieren. Der Schlüssel zum Erlernen, wie Linux intern funktioniert, ist, zu wissen, wofür ein Paket benutzt wird und warum ein Benutzer (oder das System) es benötigt. Aus diesem Grund gibt es zu jedem Paket eine Zusammenfassung des 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 sogar beim 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-Optmierungen kompilierbar ist, besteht die Gefahr, dass es fehlerhaft kompiliert wurde (z. B. wegen des komplexen Zusammenspiels zwischen Kode und den Compilerwerkzeugen). Kurz gesagt, der potientelle 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 Paketinhalts, 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 Kommandos als Benutzer *root* ausführen, und nicht als *lfs*.

6.2. Einhängen der virtuellen Kernel-Dateisysteme

Verschiedene vom Kernel exportierte Dateisysteme sind nicht physikalisch auf der Festplatte vorhanden, sondern werden zum Kommunizieren mit dem Kernel verwendet.

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

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

Und hängen Sie sie ein:

```
mount -t proc proc $LFS/proc
mount -t 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 -f -t ramfs ramfs $LFS/dev
mount -f -t tmpfs tmpfs $LFS/dev/shm
mount -f -t 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, um mit dem Installieren 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, weil alles, was Sie tun, ausschließlich auf das LFS-System beschränkt ist -- denn das, was die Shell für den Ordner / hält, ist in Wirklichkeit der Wert der Variable $LFS-Variable, die dem chroot-Kommando übergeben wurde.

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. Nun, 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 (Neustart zum Beispiel), 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!“ anzeigen. Das ist normal, weil die Datei /etc/passwd noch nicht erstellt wurde.

6.4. Ändern des Besitzers

Im Augenblick gehört der Ordner `/tools` dem Benutzer *lfs*, ein Benutzer, der aber nur auf dem Host-System existiert. Auch wenn Sie den Ordner `/tools` nach der fertigen Installation von LFS löschen möchten, entscheiden Sie sich vielleicht, es doch aufzubewahren, zum Beispiel, um weitere LFS-Systeme zu bauen. Doch wenn Sie den `/tools`-Ordner in seinem jetzigen Zustand behalten, haben Sie Dateien mit einer Benutzer-ID, zu der es kein Benutzerkonto gibt. Das ist gefährlich, denn ein später erstelltes Konto könnte genau diese ID bekommen und wäre damit plötzlich der Besitzer des Ordners `/tools` und aller Dateien darin. 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 den Inhalt des Ordners `/tools` dem Benutzer *root* zuordnen. Benutzen Sie dazu folgendes Kommando:

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

Das Kommando benutzt *`0:0`* anstelle von *`root:root`*, weil **chown** den Namen „root“ nicht auflösen kann, solange die Passwortdatei noch nicht erzeugt wurde. Wir gehen in diesem Buch davon aus, dass Sie das **chown**-Kommando ausführen.

6.5. Erstellen der Ordnerstruktur

Lassen Sie uns nun Struktur in unser LFS-System bringen und einige Ordner erstellen. Das folgende Kommando erstellt eine mehr oder weniger standardkonforme Ordnerstruktur:

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

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

Die erste Rechteänderung legt fest, dass nicht jeder den Ordner `/root` betreten darf—das gleiche, was ein normaler Benutzer mit seinem Persönlichen Ordner auch tun würde. 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 (verfügbar unter *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 hier aber noch nicht existieren. Deshalb erstellen Sie eine Reihe symbolischer Links, die aber im weiteren Verlauf des Kapitels beim Installieren der restlichen Software durch echte Dateien ersetzt werden.

```
ln -s /tools/bin/{bash,cat,pwd,stty} /bin
ln -s /tools/bin/perl /usr/bin
ln -s /tools/lib/libgcc_s.so.1 /usr/lib
ln -s 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 relevanten 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:
EOF
```

Die erzeugten Gruppen sind nicht Teil irgendeines Standards—es sind Gruppen, die Udev im nächsten Abschnitt 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 frei durch den Anwender gewählt werden, weil gut geschriebene Pakete sich nicht auf GID-Nummern verlassen 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 Benutzung der Option *+h*. Das weist **bash** an, kein internes Pfad-Hashing zu benutzen. Ohne diese Anweisung würde **bash** sich die Pfade zu ausführbaren Dateien merken. Weil Sie aber frisch installierte Programme sofort nach der Installation an ihrem neuen Ort benutzen möchten, schalten Sie die Funktion in diesem Kapitel aus.

Die Programme **login**, **agetty**, und **init** (und einige weitere) verwenden Logdateien zum Protokollieren von Informationen, wie z. B. wer sich zu welcher Zeit an das System angemeldet hat. Diese Programme schreiben aber nur in Logdateien, wenn diese auch existieren. Daher initialisieren Sie die Logdateien und vergeben die richtigen Rechte:

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

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

6.8. Bestücken von /dev

6.8.1. Erstellen der anfänglichen Gerätedateien

Beim Booten des Kernel müssen ein paar wenige Gerätedateien vorhanden sein, im einzelnen `console` und `null`. 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 ramfs und Bestücken von /dev

Der Ideale Weg, `/dev` zu bestücken, ist, `ramfs` genau wie `tmpfs` in `/dev` einzuhängen und die Gerätedateien bei jedem Bootvorgang zu erzeugen. Weil das System aber noch nicht gebootet wurde, müssen Sie das tun, was sonst die Bootskripte erledigen würden und auf diese Weise `/dev` bestücken. Hängen Sie nun `/dev` ein:

```
mount -n -t ramfs none /dev
```

Führen Sie nun das installierte Programm **udevstart** aus, um die ersten Gerätedateien auf Basis von `/sys` zu installieren:

```
/tools/sbin/udevstart
```

LFS benötigte einige symbolische Links und Ordner die nicht von Udev erzeugt werden. Erstellen Sie diese nun:

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

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

```
mount -t devpts -o gid=4,mode=620 none /dev/pts
mount -t 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 auch nicht benötigt um die Dateisysteme korrekt einzuhängen. Daher kann die Warnung problemlos ignoriert werden.

6.9. Linux-Libc-Header-2.6.8.1

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

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 22 MB

Linux-Libc-Header ist abhängig von: Coreutils

6.9.1. Installieren 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 -R include/asm-i386 /usr/include/asm
cp -R include/linux /usr/include
```

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

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

6.9.2. Inhalt von Linux-Libc-Header

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

Kurze Beschreibungen

`/usr/include/{asm,linux}/*.h` Die Linux Header-API

6.10. Man-pages-1.67

Das Paket Man-pages enthält über 1.200 Hilfetexte.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 15 MB

Man-pages 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

Kurze Beschreibungen

`Hilfeseiten` Beschreiben z. B. C- und C++-Funktionen, wichtigen Geräte- und Konfigurationsdateien.

6.11. Glibc-2.3.4-20040701

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: 784 MB

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

6.11.1. Installieren von Glibc

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel `CFLAGS` und `CXXFLAGS` - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

Das Installationssystem der Glibc ist sehr eigenständig und installiert perfekt, 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 Glibc Autoconf-Tests dann falsche Resultate ergeben würden.

Die Glibc-Dokumentation empfiehlt, nicht im Quellordner sondern in einem gesonderten Ordner zu kompilieren:

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

Bereiten Sie Glibc zum Kompilieren vor:

```
../glibc-2.3.4-20040701/configure --prefix=/usr \
    --disable-profile --enable-add-ons=nptl --with-tls \
    --with-__thread --enable-kernel=2.6.0 --without-cvs \
    --libexecdir=/usr/lib/glibc \
    --with-headers=/tools/glibc-kernheaders
```

Die Bedeutung der neuen configure-Option:

`--libexecdir=/usr/lib/glibc`
Das wird das Programm **pt_chown** in `/usr/lib/glibc` anstelle von `/usr/libexec` installieren.

Kompilieren Sie das Paket:

```
make
```

Wichtig

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

Testen Sie das Ergebnis:

```
make check
```

Die Glibc Testsuite ist sehr stark von einigen Funktionen Ihres Host-Systems abhängig, insbesondere dem Kernel. Grundsätzlich wird erwartet, dass die Glibc-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 besitzen. Bestimmte Optimierungseinstellungen haben hier ebenfalls einen gewissen Einfluss.
- Der *gettext*-Test schlägt manchmal aufgrund von Host-System bedingten Problemen fehl. Die genauen Ursachen sind noch nicht ganz geklärt.
- Der *atime*-Test schlägt fehl, wenn die LFS-Partition mit der Option *`noatime`* eingehängt wurde.
- Der *shm*-Test kann fehlschlagen, wenn auf dem Host-System das Dateisystem `devfs` verwendet wird, aber aufgrund fehlender Kernelunterstützung kein `tmpfs` Dateisystem unter `/dev/shm` gemountet ist.
- Auf alter oder langsamer Hardware können einige Tests aufgrund von Zeitüberschreitungen fehlschlagen.

Auch wenn es nur eine harmlose Nachricht 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 installiert. Holen Sie das nach:

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

Als Alternative zu dem vorigen Kommando können Sie auch nur die von Ihnen benötigten oder gewünschten Locales installieren. Das erreichen Sie mit dem Kommando **localedef**. Informationen dazu finden Sie in der Datei `INSTALL` in den Quellen zu Glibc. Jedoch gibt es einige Locales, die essentiell für die Tests von weiteren Paketen sind, im einzelnen die *libstdc++* Tests von GCC. Die folgenden Anweisungen anstelle des oben verwendeten Targets *`install-locales`* installieren einen minimalen Satz von Locales, die notwendig sind, um die nachfolgenden Tests erfolgreich durchführen zu können:

```
mkdir -p /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
```

Einige von **make localedata/install-locales** installierte Locales werden von bestimmten Anwendungen im LFS- bzw. BLFS-Buch nicht richtig unterstützt. Aufgrund der zahlreichen Probleme, die auftreten, wenn Programmierer Annahmen machen, die die Locales zerstören, sollte LFS nicht mit Locales benutzt werden, die Multi-Byte Zeichen (inklusive UTF-8) oder von-rechts-nach-links-schrift verwenden. Viele unoffizielle und instabile Patches werden benötigt, um diese Probleme in den Griff zu bekommen. Die LFS-Entwickler haben sich

entschieden, solch komplexe Locales nicht zu unterstützen. Das betrifft auch die Locales ja_JP und fa_IR—sie wurden nur installiert, damit GCC und Gettext keine Probleme in den Testsuites haben. Das Programm **watch** aus dem Paket Procps funktioniert z. B. nicht richtig in diesen Locales. Verschiedene Versuche, diese Restriktionen zu umgehen, sind in den Tipps zur internationalisierung nachzulesen.

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

```
make -C ../glibc-2.3.4-20040701/linuxthreads/man
```

Installieren Sie diese:

```
make -C ../glibc-2.3.4-20040701/linuxthreads/man install
```

6.11.2. Einrichten von Glibc

Sie müssen die Datei `/etc/nsswitch.conf` erstellen, denn obwohl Glibc bei einer fehlenden oder kaputten Datei Standardwerte vorgibt, funktionieren diese Standardwerte nicht gut mit Netzwerken. Ausserdem müssen Sie die Zeitzone korrekt einstellen.

Erstellen Sie die neue Datei `/etc/nsswitch.conf`, indem Sie das folgende Kommando ausführen:

```
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
```

Um herauszufinden, in welcher Zeitzone Sie sind, führen Sie dieses Skript aus:

```
tzselect
```

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

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

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

Die Bedeutung der Option zu cp:

--remove-destination
: Dadurch wird das Entfernen des bereits existierenden symbolischen Links erzwungen. Der Grund, warum wir kopieren anstatt einen symbolischen Link zu benutzen, ist der, dass wir den Fall abdecken wollen, dass `/usr` auf einer separaten Partition liegt. Das könnte z. B. problematisch werden, wenn in den 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 dynamischen Bibliotheken, die von ausführbaren Programmen zur Laufzeit benötigt werden. Wenn allerdings Bibliotheken 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`. Fügen Sie diese Ordner 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 (wenn 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**-Modul 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-Kode zum Implementieren des RPC-Protokolls
rpcinfo	Generiert eine RPC-Anfrage an einen RPC-Server
sln	Das statisch gelinkte Programm **ln**
sprof	Liest Profiling-Daten zu Shared-Objects und zeigt sie an
tzselect	Stellt dem Anwender einige Fragen zu seinem Standort und erzeugt 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
`ld.so`	Ist ein Hilfsprogramm für ausführbare gemeinsame Bibliotheken
`libBrokenLocale`	Wird von Programmen wie z. B. Mozilla verwendet, um Probleme mit defekten Locales zu beheben
`libSegFault`	Handhabt Speicherzugriffsfehler
`libanl`	Eine Bibliothek zum asynchronen Nachschlagen von Namen
`libbsd-compat`	Ermöglicht einigen BSD-Programmen unter Linux zu laufen
`libc`	Die C-Bibliothek
`libcrypt`	Die Kryptographie-Bibliothek
`libdl`	Eine Schnittstellenbibliothek zum dynamischen Linker
`libg`	Eine Laufzeitbibliothek zu **g++**
`libieee`	Die Fließkomma-Bibliothek des Institute of Electrical and Electronic Engineers (IEEE)
`libm`	Die mathematische Bibliothek

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

6.12. Erneutes Anpassen der Toolchain

Nun, nachdem die neue C Bibliothek 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 genau 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`.

Als erstes wird der Linker angepasst. Aus diesem Grunde haben wir die Quell- und Kompilierordner aus dem zweiten Durchlauf von Binutils bestehen lassen. 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. Daraus resultiert, 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 Programme *nur* gegen die Bibliotheken in `/usr/lib` und `/lib` gelinkt. Das zusätzliche *`INSTALL=/tools/bin/install`* wird benötigt, weil das Makefile aus dem zweiten Durchlauf immer noch die Referenz auf **/usr/bin/install** enthält, welches wir 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 die GCC specs-Datei ergänzen, so dass sie den neuen dynamischen Linker referenziert. Ein einfaches sed-Kommando erledigt diese Aufgabe:

```
sed -i 's@ /tools/lib/ld-linux.so.2@ /lib/ld-linux.so.2@g' \
    `gcc --print-file specs`
```

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

Wichtig

Wenn Sie an 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 dem Namen des Linkers für Ihre Plattform ersetzen. Wenn nötig, schlagen Sie nochmal im Abschnitt Abschnitt 5.3, „Technische Anmerkungen zur Toolchain“ nach.

Achtung

Es ist an diesem Punkt zwingend notwendig, die grundlegenden Funktionen (Kompilieren und Linken) der angepassten Toolchain zu überprüfen. Aus diesem Grund führen Sie 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 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 dummy.c a.out
```

6.13. Binutils-2.15.91.0.2

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

Geschätzte Kompilierzeit: 1.4 SBU
Ungefähr benötigter Festplattenplatz: 167 MB

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

6.13.1. Installieren von Binutils

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel CFLAGS und CXXFLAGS - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

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

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

Wenn Sie die folgende Meldung sehen, ist Ihre chroot-Umgebung nicht für PTY's 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 ../binutils-build
cd ../binutils-build
```

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.15.91.0.2/configure --prefix=/usr \
    --enable-shared
```

Kompilieren Sie das Paket:

```
make tooldir=/usr
```

Normalerweise ist *tooldir* (der Ordner, in den die ausführbaren Dateien installiert werden) auf `$(exec_prefix)/$(target_alias)` gesetzt, welches dann zum Beispiel zu `/usr/i686-pc-linux-gnu` aufgelöst wird. Da wir aber nur für unser eigenes System installieren, brauchen wir diesen speziellen Ordner in `/usr` nicht. Diese Konfiguration würde benutzt werden, 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

Die Binutils-Testsuite in diesem Abschnitt wird 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 ../binutils-2.15.91.0.2/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-Profiling-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 Verwirft Symbole aus Objektdateien

`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-3.4.1

Das Paket GCC enthält die GNU-Compiler-Sammlung, die auch die C- und C++-Compiler beinhaltet.

Geschätzte Kompilierzeit: 11.7 SBU
Ungefähr benötigter Festplattenplatz: 294 MB

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

6.14.1. Installieren von GCC

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel CFLAGS und CXXFLAGS - für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

Entpacken Sie die Archive GCC-core *und* GCC-g++—sie entpacken sich in den gleichen Ordner. Auf die gleiche Weise entpacken Sie bitte auch das GCC-Testsuite-Paket. Das vollständige GCC-Paket enthält noch weitere Compiler. Eine Anleitung, wie Sie diese installieren können, finden Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/general/gcc.html*.

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

```
patch -Np1 -i ../gcc-3.4.1-no_fixincludes-1.patch
```

GCC hat Probleme beim Kompilieren einiger Programme, die nicht zu einer Basis-LFS-Installation gehören (z. B. Mozilla und kdegraphics), wenn GCC zusammen mit einer neuen Version von Binutils eingesetzt wird. Wenden Sie den folgenden Patch an, um das Problem zu beheben:

```
patch -Np1 -i ../gcc-3.4.1-linkonce-1.patch
```

Wenden Sie nun einen Sed-Befehl an; dadurch wird die Installation von libiberty.a verhindert. Wir möchten die von Binutils bereitgestellte Version von libiberty.a verwenden:

```
sed -i 's/install_to_$(INSTALL_DEST) //' libiberty/Makefile.in
```

Die GCC-Dokumentation empfiehlt, GCC nicht im Quellordner sondern in einem gesonderten Ordner zu kompilieren:

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

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-3.4.1/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

Die Testsuite in diesem Abschnitt wird 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 5.13, „GCC-3.4.1 - 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 -s ../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 -s 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 möglichweise 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,so] und libsupc++.a

Kurze Beschreibungen

cc	Der C-Compiler
cpp	Der C-Präprozessor. Er wird vom Compiler benutzt, um #include, #define und ähnliche Anweisungen im Quellkode durch ihren endgültigen Kode zu ersetzen.
c++	Der C++-Compiler
g++	Der C++-Compiler
gcc	Der C-Compiler
gccbug	Ein Shellskript, mit dem man gute 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 eine große Anzahl an Shell-Werkzeugen zum Einstellen der grundlegenden Systemeigenschaften.

Geschätzte Kompilierzeit: 0.9 SBU
Ungefähr benötigter Festplattenplatz: 69 MB

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

6.15.1. Installieren von Coreutils

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

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

Verhindern Sie, dass Coreutils Programme installiert, die später von anderen Paketen zur Verfügung gestellt werden:

```
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 macht einige Annahmen in Hinsicht auf die Existenz von Benutzern und Gruppen, die in dem frühen Stadium unseres LFS-Systems noch nicht vorhanden sind. Sie müssen noch einige Dinge einrichten, bevor Sie die Tests laufen lassen können. Falls Sie diese Testsuite nicht ausführen möchten, fahren Sie mit „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 richtige Stelle:

```
mv /usr/bin/{[,basename,cat,chgrp,chmod,chown,cp,dd,df} /bin
mv /usr/bin/{date,echo,false,head,hostname,install,ln} /bin
mv /usr/bin/{ls,mkdir,mknod,mv,pwd,rm,rmdir,sync} /bin
mv /usr/bin/{sleep,stty,test,touch,true,uname} /bin
mv /usr/bin/chroot /usr/sbin
```

Schließlich erstellen Sie noch einen symbolischen Link, um FHS-konform zu sein:

```
ln -s ../../bin/install /usr/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 Standard-Ausgabe aus bzw. fügt sie zusammen

chgrp Ändert die Gruppenzugehörigkeit einer Datei. Die Gruppe kann entweder als Name oder als numerische ID angegeben werden

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

chroot Führt einen Befehl mit dem angegebenen Ordner als `/` aus

cksum Gibt die CRC-Prüfsumme (Cyclic Redundancy Check) und die Anzahl der Bytes für jede angegebene 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 jeder neuen Datei 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 das Kommando **ls**)

dircolors Gibt Kommandos zum Setzen der Umgebungsvariable `LS_COLOR` 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 Abschlusskode, 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	Startet ein Programm mit geänderter Priorität
nl	Nummeriert die Zeilen der angegebenen Dateien
nohup	Führt ein Programm aus, so dass es immun gegen „hangup“s ist, die Ausgaben 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 printf Funktion sehr ähnlich
ptx	Erzeugt aus dem Inhalt von Dateien einen vertauschten Index, mit jedem Stichwort im Kontext
pwd	Gibt den Namen des aktuellen 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 Statuskode, 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 das Kommando **ls -l**
wc	Gibt die Anzahl Zeilen, Wörter und Bytes einer Datei aus. Und eine Summe, falls mehrere Dateien angegeben wurden.
who	Gibt aus, 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-1.2.1

Zlib enthält die Bibliothek libz. Sie wird von einigen Programmen zum Komprimieren und Dekomprimieren genutzt.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1.5 MB

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

6.16.1. Installation von Zlib

Der folgende Patch behebt ein „Denial of Service“-Problem in der Zlib Kompressions-Bibliothek:

```
patch -Np1 -i ../zlib-1.2.1-security-1.patch
```

Anmerkung

Vorsicht: Zlib baut seine gemeinsamen Bibliotheken falsch, wenn die Umgebungsvariable `CFLAGS` gesetzt ist. Wenn 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
```

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie die gemeinsamen Bibliotheken:

```
make install
```

Kompilieren Sie die statische Bibliothek:

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

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

Installieren Sie die statische Bibliothek:

```
make install
```

Und korrigieren Sie die Zugriffsrechte auf die statische Bibliothek:

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

Wichtige gemeinsame Bibliotheken sollten in `/lib` installiert werden. Auf diese Weise haben Systemprogramme beim Booten, wo `/usr` möglicherweise noch nicht verfügbar (weil nicht eingehängt) ist, trotzdem Zugriff auf diese Bibliotheken.

Aus dem obigen Grund verschieben Sie die Laufzeitkomponenten der gemeinsamen Zlib-Bibliothek in den Ordner `/lib`:

```
mv /usr/lib/libz.so.* /lib
```

Korrigieren Sie den symbolischen Link `/usr/lib/libz.so`:

```
ln -sf ../../lib/libz.so.1 /usr/lib/libz.so
```

6.16.2. Inhalt von Zlib

Installierte Bibliotheken: libz[a,so]

Kurze Beschreibungen

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

6.17. Mktemp-1.5

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

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 317 KB

Mktemp ist abhängig von: Coreutils, Make und Patch

6.17.1. Installation von Mktemp

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

```
patch -Np1 -i ../mktemp-1.5-add_tempfile-1.patch
```

Bereiten Sie Mktemp zum Kompilieren vor:

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

Die Bedeutung der configure-Option:

`--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 Mktemp

Installierte Programme: mktemp und tempfile

Kurze Beschreibungen

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

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

6.18. Iana-Etc-1.01

Das Paket Iana-Etc stellt Daten zu Netzwerkdiensten und Protokollen zur Verfügung.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 641 KB

Iana-Etc ist abhängig von: Make

6.18.1. Installation von Iana-Etc

Auffinden der Daten:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.18.2. Inhalt von Iana-Etc:

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

Kurze Beschreibungen

`/etc/protocols`	Beschreibt verschiedene DARPA Internet Protokolle, die im TCP/IP-Subsystem verfügbar sind
`/etc/services`	Ermöglicht eine Zuordnung von leicht zu lesenden Namen für Internetdienste und den zugehörigen Port-Nummern und Protokolltypen

6.19. Findutils-4.1.20

Das Paket Findutils enthält Programme zum Auffinden von Dateien, entweder durch rekursive Suche in einer Ordnerstruktur oder über den Zugriff auf eine Datenbank (was häufig schneller ist, aber die Gefahr birgt, dass die Datenbank nicht den aktuellen Zustand wiederspiegelt).

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 7.5 MB

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

6.19.1. Installieren von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

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

Die obige *localstatedir*-Anweisung 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 das Kommando **`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. Es benutzt die sogenannte front-Komprimierung, welche die Datenbankgröße um den Faktor 4 bis 5 verkleinert.

locate Durchsucht eine Datenbank mit Dateinamen und gibt die Dateien aus, die eine bestimmte Zeichenkette enthalten oder auf ein bestimmtes Muster 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: 17 MB

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

6.20.1. Installieren 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 das Kommando **`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 pwcat

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 `/etc/group` an
igawk	Ermöglicht gawk das Einbinden von Dateien
pgawk	Die Profiling-Version von **gawk**
pgawk-3.1.4	Ein harter Link auf **pgawk**
pwcat	Zeigt die Passwortdatenbank `/etc/passwd` 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: 27 MB

Ncurses 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
```

Setzen Sie die Ausführungsrechte für die Ncurses-Bibliothek:

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

Korrigieren Sie eine Bibliothek, die nicht ausführbar sein sollte:

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

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

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

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

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

6.21.2. Inhalt von Ncurses

Installierte Programme: captoinfo (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

captoinfo 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 die Readline-Kommandozeilen-Bibliothek.

Geschätzte Kompilierzeit: 0.11 SBU
Ungefähr benötigter Festplattenplatz: 3.8 MB

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

6.22.1. Installieren von Readline

Der folgende Patch behebt ein Problem, bei dem Readline manchmal nur 33 Zeichen einer Zeile anzeigt und dann zur nächsten Zeile springt.

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

Bereiten Sie Readline zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make SHLIB_XLDFLAGS=-lncurses
```

Die Bedeutung der make-Option:

`SHLIB_XLDFLAGS=-lncurses`
Diese Option zwingt Readline, gegen die Bilbiothek `libncurses` zu linken.

Installieren Sie das Paket:

```
make install
```

Vergeben Sie Readline's dynamischen Bibliotheken passendere Zugriffsrechte:

```
chmod 755 /usr/lib/*.5.0
```

Und verschieben Sie die dynamischen Bibliotheken an eine bessere Stelle:

```
mv /usr/lib/lib{readline,history}.so.5* /lib
```

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

```
ln -sf ../../lib/libhistory.so.5 /usr/lib/libhistory.so
ln -sf ../../lib/libreadline.so.5 /usr/lib/libreadline.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: 34 MB

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

Alternativen zu Vim

Wenn Sie einen anderen Editor als Vim 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 den Konfigurationsdateien `vimrc` und `gvimrc` nach `/etc`:

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

Bereiten Sie Vim zum Kompilieren vor:

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

Der optionale—aber dringend empfohlene—Parameter *`--enable-multibyte`* schaltet die Unterstützung zum Editieren von Dateien mit Multi-Byte Zeichenkodierung in **vim** ein. Das wird benötigt, wenn Sie ein Locale mit Multi-Byte Zeichensatz verwenden. Dieser Schalter ist auch hilfreich, wenn Sie Dateien bearbeiten möchten, die mit Distributionen wie z. B. Fedora Core erzeugt wurden. Diese Distributionen benutzen UTF-8 als voreingestellen Zeichensatz.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu testen, kann das folgende Kommando verwendet werden: **`make test`**. Die Testsuite gibt jedoch jede Menge sinnlose Zeichen auf dem Bildschirm aus und könnte die Einstellungen Ihres Terminals durcheinander bringen. Das Durchlaufen dieser Testsuite ist daher ausdrücklich optional.

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 -s vim /usr/bin/vi
```

Wenn Sie später das X-Window-System auf Ihrem LFS installieren möchten, sollten Sie nach der Installation von X Ihren Vim erneut installieren. Vim bringt eine schöne grafische Oberfläche mit, die allerdings X und ein paar weitere Bibliotheken voraussetzt. Weitere Informationen finden Sie in der Vim Dokumentation 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. Ausserdem wird daran erinnert, dass „compatible“ als erstes eingestellt werden muss, wenn Sie zum kompatiblen Modus wechseln möchten. Das ist nötig, weil diese Einstellung viele Parameter einstellt, und Änderungen an diesen Parametern müssen danach erfolgen, weil sie sonst keine Auswirkung hätten. 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 **vim**s Hervorheben von Syntax. Schließlich stellt die *if*-Verzweigung sicher, dass mittels *`set background=dark`* die Hintergrundfarbe von bestimmten Terminals besser eingestellt ist. Dadurch wird das Hervorheben von 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	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	Der Editor
vim	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. M4-1.4.2

M4 enthält einen Makroprozessor.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3.0 MB

M4 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 das Kommando **`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-1.875a

Bison erstellt ein Programm, das die Struktur einer Textdatei analysiert.

Geschätzte Kompilierzeit: 0.6 SBU
Ungefähr benötigter Festplattenplatz: 10.6 MB

Bison 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 das Kommando **`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 zu 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: 3.4 MB

Less 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-Option:

`--sysconfdir=/etc`
: Diese Option 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 lässt Sie darin blättern, nach Zeichenketten suchen und zu Markierungen springen.

lessecho Wird zum Expandieren von Metazeichen in Unix-Dateinamen benötigt, so wie z. B. * und ?.

lesskey Wird zum Festlegen der Tastenbelegung für **less** benutzt

6.27. Groff-1.19.1

Groff enthält verschiedene Programme zur Verarbeitung und Formatierung von Text.

Geschätzte Kompilierzeit: 0.5 SBU
Ungefähr benötigter Festplattenplatz: 43 MB

Groff 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 alle in den Vereinigten Staaten ist *PAGE=letter* korrekt. Wenn Ihr Standort woanders ist, ersetzen Sie besser *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 -s soelim /usr/bin/zsoelim
ln -s eqn /usr/bin/geqn
ln -s 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 font-metrische 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

eqn 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, rät, welche der **groff**-Optionen `-e`, `-man`, `-me`, `-mm`, `-ms`, `-p`, `-s` und `-t` zum Drucken benötigt werden, und gibt das nötige **groff**-Kommando aus

grolbp Ein **groff**-Treiber für Canon CAPSL-Drucker (Laserdrucker der LBP-4 und LBP-8 Serie)

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 Die GNU-Implementierung von **tbl**

hpftodit 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 (Standard Code for Information Interchange)

nroff Ein Skript, das **nroff**-Kommandos mit **groff** emuliert

pfbtops Übersetzt eine Postscript-Schrift im `.pfb`-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 standard Ausgabe, ausser das Zeilen zwischen *.[* und *.]* als Zitat interpretiert werden und Zeilen zwischen *.R1* und *.R2* als Kommandos behandelt werden, die angeben, wie mit Zitaten

	umgegangen werden soll
soelim	Liest Dateien und ersetzt Zeilen der Form *.so Datei* durch den Inhalt der erwähnten *Datei*
tbl	Kompiliert in troff Eingabedateien eingebettete Beschreibungen von Tabellen zu Kommandos, die von **troff** unterstützt werden
tfmtodit	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	Die GNU-Implementierung von **soelim**

6.28. Sed-4.1.2

Das Paket Sed enthält einen Stream-Editor.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 5.2 MB

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

6.28.1. Installieren von Sed

Bereiten Sie Sed zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie das Kommando **`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

Das Programm Flex wird benutzt um Programme zu erzeugen, die Muster in Texten erkennen können.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3.4 MB

Flex 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-2.patch
```

Die GNU autotools erkennen, dass der Quellkode von Flex durch den vorhergehenden Patch verändert wurde und versucht, 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 das Kommando **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

Es existieren einige Programme, die die `lex`-Bibliothek in `/usr/lib` erwarten. Erstellen Sie daher einen entsprechenden symbolischen Link:

```
ln -s 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 755 /usr/bin/lex
```

6.29.2. Inhalt von Flex

Installierte Programme: flex, flex++ (Link auf 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 Regeln nach denen es suchen soll ein Programm, das nach diesen Mustern sucht.
flex++	Startet eine Version von **flex**, die exklusiv für C++-Scanner verwendet wird
lex	Ein Skript, welches **flex** im **lex**-Emulationsmodus startet
`libfl.a`	Die `flex`-Bibliothek

6.30. Gettext-0.14.1

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

Geschätzte Kompilierzeit: 0.5 SBU
Ungefähr benötigter Festplattenplatz: 55 MB

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

6.30.1. Installieren 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`**. Dies braucht sehr lange, etwa 7 SBUs.

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 und xgettext
Installierte Bibliotheken: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] und libgettextsrc[a,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 rohe Übersetzungen in eine einzige Datei
msgunfmt	Erzeugt aus einem binären Katalog einen rohen 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
`libasprintf`	Definiert die *autosprintf*-Klasse; sie macht C-formatierte Routinen in C++ Programmen verfügbar, vor allem zur Verwendung mit <*string*> Strings und den <*iostream*> Streams
`libgettextlib`	Eine private Bibliothek, die die allgemeinen Routinen der verschiedenen gettext-Programme enthält. Sie sind nicht zur normalen Verwendung gedacht
`libgettextpo`	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**)
`libgettextsrc`	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: 11 MB

Inetutils 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 wird Inetutils die Man-pages zu diesen Programmen installieren. Der folgende Patch korrigiert 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-Dämonen übergibt. Sie installieren ihn nicht, weil etwas später durch Util-Linux eine bessere Version installiert wird.

`--disable-syslogd`
: Diese Option verhindert die Installation des System-Log-Dämonen, weil Sie einen mit dem Sysklogd Paket installieren.

`--disable-whois`

Dies verhindert das Kompilieren des **whois**-Clients, welcher leider elendig veraltet ist. Eine Anleitung für einen besseren **whois**-Client finden Sie im BLFS-Buch.

`--disable-servers`
Das verhindert die Installation verschiedener Netzwerkserver die dem Inetutils-Paket beiliegen. Diese gelten in einem basis LFS-System als nicht angebracht. Einige sind von Natur aus unsicher und nur in vertrauenswürdigen Netzen sicher einsetzbar. Mehr Informationen finden Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html*. Beachten Sie, dass es für fast alle dieser Netzwerkserver einen besseren Ersatz gibt.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Und verschieben Sie das Programm **ping** an die korrekte Stelle:

```
mv /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	Eine Schnittstelle zum TELNET-Protokoll
tftp	Das Programm zu TFTP (Trivial File Tranfer Protocol)

6.32. Iproute2-2.6.8-040823

Das Paket Iproute2 enthält verschiedene Programme zur grundlegenden IPv4-basierten Netzwerkunterstützung.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: .6 MB

Iproute2 ist abhängig von: GCC, Glibc, Make, Linux-Headers und Sed

6.32.1. Installieren von Iproute2

Das Programm **arpd** aus diesem Paket ist von Berkeley DB abhängig. Weil **arpd** in einem Basis-System aber nicht besonders häufig benötigt wird, entfernen Sie die Abhängigkeit zu Berkeley DB mit dem folgenden Patch. Falls das Programm **arpd** doch benötigt wird, finden Sie eine Anleitung zum Installieren von Berkeley DB im BLFS-Buch unter *http://www.linuxfromscratch.org/blfs/view/svn/content/databases.html#db*.

```
patch -Np1 -i ../iproute2-2.6.8_040823-remove_db-1.patch
```

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: ifstat, ip, nstat, routef, routel, rtmon, rtstat, ss und tc.

Kurze Beschreibungen

ifstat Zeigt Schnittstellenstatistiken an, inklusive der Menge der gesendeten und empfangenen Pakete pro Schnittstelle

ip Dies ist die wesentliche ausführbare Datei. Sie hat verschiedene Funktionen:

ip link ***[gerät]*** zeigt den Gerätestatus an und ermöglicht Änderungen an den Einstellungen

ip addr zeigt Adressen und ihre Eigenschaften an, fügt neue Adressen hinzu und löscht alte

ip neighbor zeigt Bindungen und Eigenschaften von benachbarten Geräten an, fügt neue Nachbargerätebindungen hinzu und löscht alte

ip rule zeigt Routingregeln an und bearbeitet sie

ip route ermöglicht das Anzeigen und Ändern von Routingtabellen

ip tunnel zeigt IP-Tunnel und die Eigenschaften an und ermöglicht Änderungen daran

ip maddr zeigt Multicast-Adressen und ihre Eigenschaften an und ermöglicht Änderungen

ip mroute setzt, ändert oder löscht Multicast-Routen

ip monitor ermöglicht, dauerhaft den Status von Netzwerkgeräten, Adressen und Routen zu überwachen.

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

rtmon	Ein Werkzeug zum Überwachen des Routing
rtstat	Ein Werkzeug für den Routingstatus
ss	Ähnlich wie das Kommando **netstat**. Zeigt aktive Verbindungen an.
tc	Programm zur Kontrolle des Netzwerkverkehrs (Traffic Controlling). Implementiert Quality of Service (QOS) und Class Of Service (COS): **tc qdisc** ermöglicht das Einstellen der Warteschlangen-Regeln **tc class** ermöglicht das Einrichten von Klassen, basierend auf einer Warteschlangen-Regelung **tc estimator** ermöglicht das Schätzen des Netzwerk-Flusses in ein Netzwerk **tc filter** ermöglicht das Erstellen von QOS/COS Paketfiltern **tc policy** ermöglicht das Erstellen von QOS/COS Regelwerken

6.33. Perl-5.8.5

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

Geschätzte Kompilierzeit: 2.9 SBU
Ungefähr benötigter Festplattenplatz: 143 MB

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

6.33.1. Installieren von Perl

Wenn Sie die vollständige Kontrolle über die Art haben möchten, wie Perl sich selbst zum Installieren einrichtet, dann können Sie stattdessen das interaktive **Configure**-Skript benutzen. Wenn Sie mit den (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-Option:

`-Dpager="/bin/less -isR"`
Dies korrigiert einen Fehler in **perldoc** in Zusammenhang mit dem Programm **less**.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite ausführen möchten, müssen Sie erst eine Basisversion der Datei `/etc/hosts` erstellen. Diese wird benötigt, damit einige der Tests den Hostnamen localhost auflösen können:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Wenn Sie möchten, können Sie nun die Tests ausführen:

```
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, perl5.8.5 (Link auf perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (Link auf s2p), pstruct (Link auf c2ph), s2p, splain und xsubpp
Installierte Bibliotheken: Einige hundert, die hier nicht alle aufgelistet werden können

Kurze Beschreibungen

a2p Übersetzt awk zu Perl

c2ph Gibt C-Strukturen aus, die von **cc -g -S** erzeugt wurden

dprofpp Zeigt Perl-Profiling-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 `.h` C Header-Dateien zu `.ph` Perl Header-Dateien

h2xs Konvertiert `.h` C Header-Dateien zu Perl-Erweiterungen

libnetcfg Kann zum Einrichten von `libnet` benutzt werden

perl Kombiniert die besten Eigenschaften von C, sed, awk und sh in einer einzigen universellen Sprache

perl5.8.5 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 `.pl`-Dateien in Perl5 `.pm`-Module
pod2html	Konvertiert pod-Dateien in das HTML-Format
pod2latex	Konvertiert Dateien im pod-Format nach LaTeX
pod2man	Konvertiert pod-Daten zu formatierter *roff-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-Dokumentation Dateien
podselect	Zeigt ausgewählte Abschnitte einer pod-Dokumentation an
psed	Die Perl-Version des Stream-Editors **sed**
pstruct	Gibt C-Strukturen aus, die von **cc -g -S** erzeugt wurden
s2p	Übersetzt sed zu Perl
splain	Erzwingt ausführliche Analyse von Warnungen in Perl
xsubpp	Konvertiert Perl XS-Kode zu C-Kode

6.34. Texinfo-4.7

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

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 17 MB

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

6.34.1. Installieren von Texinfo

Der folgende Patch behebt das Problem, dass **info** manchmal beim Drücken der *Entfernen*-Taste abstürzt:

```
patch -Np1 -i ../texinfo-4.7-segfault-1.patch
```

Bereiten Sie Texinfo zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie das Kommando **`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-Variable `TEXMF` enthält den Standort Ihres TeX Basisordners, falls später

ein TeX-Paket installiert wird.

Das Info-Dokumentationssystem speichert seine 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, können Sie dazu das folgende Kommando verwenden:

```
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 und texindex

Kurze Beschreibungen

info Wird zum Lesen von Info-Dokumenten benutzt. Info-Dokumente sind ähnlich wie Man-pages, aber gehen oft tiefer in die Materie als einfach nur die möglichen Parameter zu beschreiben. Vergleichen Sie zum Beispiel **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

texindex Sortiert Texinfo-Indexdateien

6.35. Autoconf-2.59

Autoconf erstellt Shell-Skripte, die automatisch Quelltexte zum Kompilieren einrichten.

Geschätzte Kompilierzeit: 0.5 SBU
Ungefähr benötigter Festplattenplatz: 7.7 MB

Autoconf 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 das Kommando **`make check`** benutzen. Dies dauert lange; etwa 2 SBUs.

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 automatisch Quellcode-Pakete einrichten, um sie an unterschiedliche Unix-System anzupassen. 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 *#define*-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 `configure.in`-Datei für ein Softwarepaket behilflich sein. Es untersucht die Quelldateien in einem Ordner und sucht nach üblichen Portabilitätsproblemen und erzeugt eine `configure.scan`-Datei, die als Basis für eine `configure.in`-Datei zu dem Softwarepaket dienen kann.

autoupdate Verändert eine `configure.in`-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 helfen, herauszufinden welche Tests **configure** durchführen muss. Es kann einige Lücken in `autoscan`-generierten `configure.in`-Dateien füllen.

6.36. Automake-1.9.1

Automake generiert Makefile-Dateien zur Verwendung mit Autoconf.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 6.8 MB

Automake 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 das Kommando **`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-1.9.1, automake, automake-1.9.1, 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, welches M4-Dateien im aclocal-Stil installiert
aclocal	Erzeugt auf dem Inhalt von `configure.in`-Dateien basierend, entsprechende `aclocal.m4`-Dateien
aclocal-1.9.1	Ein harter Link auf **aclocal**

automake	Ein Werkzeug zum automatischen Erzeugen von `Makefile.in`'s aus sog. `Makefile.am`-Dateien. Um alle `Makefile.in`-Dateien eines Pakets zu erzeugen, lassen Sie dieses Programm im Basisordner des Pakets laufen. Durch das Scannen von `configure.in` findet es automatisch jede nötige `Makefile.am`-Datei und erzeugt die entsprechende `Makefile.in`-Datei.
automake-1.9.1	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-Kode
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
mkinstalldirs	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: 27 MB

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

6.37.1. Installieren von Bash

Der folgende Patch ist nötig, wenn Readline nicht wie empfohlen installiert wurde. Dieser Patch behebt ein Problem, bei dem Bash manchmal nur 33 Zeichen in einer Zeile ausgibt und dann die nächste Zeile beginnt. Wenn Readline wie empfohlen gepatcht und installiert wurde, ist dieser Patch nicht nötig, denn das Readline-Paket behebt das Problem bereits.

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

Bereiten Sie Bash zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin \
    --without-bash-malloc --with-installed-readline
```

Die Bedeutung der configure-Option:

`--with-installed-readline`
Diese Option lässt Bash die von uns installierte `readline`-Bibliothek anstelle der Bash-internen 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 ist 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 einheitlichen Format zu Erstellen und per Email 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-4.10

File ist ein kleines Werkzeug zum Identifizieren von Dateitypen.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.3 MB

File 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

Installiertes Programm: file
Installierte Bibliotheken: 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-1.5.8

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: 20 MB

Libtool 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 das Kommando **`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. Bzip2-1.0.2

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. Bei Textdateien wird eine wesentlich bessere Kompressionsrate als mit dem traditionellen Kommando **gzip** erreicht.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3.0 MB

Bzip2 ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc und Make

6.40.1. Installieren von Bzip2

Bereiten Sie Bzip2 zum Kompilieren vor:

```
make -f Makefile-libbz2_so
make clean
```

Der Schalter *-f* veranlasst Bzip2, ein anderes `Makefile` (in diesem Fall `Makefile-libbz2_so`) zu verwenden. Dieses erzeugt eine dynamische Bibliothek `libbz2.so` und verlinkt die Bzip2-Werkzeuge damit.

Kompilieren Sie das Paket:

```
make
```

Falls Sie Bzip2 neu installieren müssen, müssen Sie zuerst **`rm -f /usr/bin/bz*`** ausführen, ansonsten schlägt **make install** fehl.

Installieren Sie die Programme:

```
make install
```

Installieren Sie die ausführbare Datei **bzip2** nach `/bin`. Dann erzeuen Sie ein paar nötige symbolische Links und räumen auf:

```
cp bzip2-shared /bin/bzip2
cp -a libbz2.so* /lib
ln -s ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm /usr/bin/{bunzip2,bzcat,bzip2}
ln -s bzip2 /bin/bunzip2
ln -s bzip2 /bin/bzcat
```

6.40.2. Inhalt von Bzip2

Installierte Programme: bunzip2 (Link auf bzip2), bzcat (Link auf bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless und bzmore
Installierte Bibliotheken: libbz2.a, libbz2.so (Link auf libbz2.so.1.0), libbz2.so.1.0 (Link auf libbz2.so.1.0.2) und libbz2.so.1.0.2

Kurze Beschreibungen

bunzip2	Dekomprimiert bzip2-Dateien
bzcat	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 Textkompressionsalgorythmus 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
bzmore	Führt **more** auf bzip2-Dateien aus
`libbz2*`	Die Bibliothek, die verlustlose blocksortierende Datenkompression mit Hilfe des Burrows-Wheeler-Algorythmus 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: 7.5 MB

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

6.41.1. Installieren 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: 12 MB

Kbd 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), psfstriptable (Link auf psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start und unicode_stop

Kurze Beschreibungen

chvt	Ändert das vordergründige 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 aktuell verwendete Unimap 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 Konsoletreiber 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**
psfstriptable	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-Kode 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-1.35

E2fsprogs stellt die Dateisystemwerkzeuge für die Benutzung des `ext2`-Dateisystems zur Verfügung. Auch `ext3` wird unterstützt (ein Journaling Dateisystem).

Geschätzte Kompilierzeit: 0.6 SBU
Ungefähr benötigter Festplattenplatz: 4.9 MB

E2fsprogs ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed und Texinfo

6.43.1. Installation von E2fsprogs

Es wird empfohlen, E2fsprogs ausserhalb des Quellordners zu kompilieren:

```
mkdir 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. Wenn zum Beispiel `/usr` nicht eingehängt ist, müssen diese Programme trotzdem verfügbar sein. Sie gehören in Ordner wie `/lib` und `/sbin`. Wenn diese Option nicht an E2fsprogs configure-Skript übergeben wird, würden die Programme entgegen unserem Willen im Ordner `/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. Dieses Plugin ist nicht auf dem aktuellen Stand der neuesten internen EVMS Schnittstellen und EVMS wird nicht als Teil des LFS Basis-Systems installiert; daher

brauchen wir dieses Plugin nicht. Besuchen Sie die Webseite von EVMS unter *http://evms.sourceforge.net/* um mehr Informationen über EVMS zu erhalten.

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Meiste aus dem Paket:

```
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 des `ext2`-Dateisystem)

compile_et Ein Fehlertabellen-Compiler. Er konvertiert eine Tabelle mit Fehlerkode-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 Hilfsmeldungen zu C-Quellkode, der dann mit der `libss` Subsystem-Bibliothek verwendet werden kann
mke2fs	Wird zum Erstellen eines ext2-Dateisystems auf einem Gerät verwendet
mkfs.ext2	Erzeugt in der Voreinstellung ein `ext2`-Dateisystem
mkfs.ext3	Erzeugt in der Voreinstellung ein `ext3`-Dateisystem
mklost+found	Wird benutzt, um den Ordner `lost+found`-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 `ext2`-Dateisystems verwendet werden
tune2fs	Wird zum Einstellen von veränderbaren Parametern auf einem `ext2`-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.
`libblkid`	Enthält Routinen zum Identifizieren von Geräten und zum Extrahieren von Token
`libcom_err`	Die allgemeine Routine zum Anzeigen von Fehlern
`libe2p`	Wird von **dumpe2fs**, **chattr** und **lsattr** benutzt

`libext2fs`	Enthält Routinen, die Programme im Benutzerkontext zum Manipulieren eines `ext2`-Dateisystems verwenden können
`libss`	Wird von **debugfs** benutzt
`libuuid`	Enthält Routinen zum Erzeugen von einmaligen Bezeichnern für Objekte, die hinter dem lokalen System verfügbar sein könnten

6.44. Grep-2.5.1

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 5.8 MB

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

6.44.1. Installieren von Grep

Bereiten Sie Grep zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin --with-included-regex
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie das Kommando **`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-0.95

Das Paket Grub enthält den GRand Unified Bootloader.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 10 MB

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

6.45.1. Installation von Grub

Dieses Paket funktioniert nicht gut, wenn nicht die Standard Optimierungseinstellungen (inklusive der Optionen *-march* und *-mcpu*) benutzt werden. Deshalb sollten eventuell gesetzte Umgebungsvariablen, die die Standardoptimierung überschreiben - zum Beispiel CFLAGS und CXXFLAGS - für den Kompiliervorgang 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 das Kommando **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. Diese Partitionen werden normalerweise nur von Sun Workstations benutzt.

Installieren Sie das Paket:

```
make install
mkdir /boot/grub
cp /usr/share/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: 2.6 MB

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

6.46.1. Installation von Gzip

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
```

Und verschieben Sie die Programme in den Ordner /bin:

```
mv /usr/bin/gzip /bin
rm /usr/bin/{gunzip,zcat}
ln -s gzip /bin/gunzip
ln -s gzip /bin/zcat
ln -s gunzip /bin/uncompress
```

6.46.2. Inhalt von Gzip

Installierte Programme: 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

gunzip	Dekomprimiert gzip-Dateien
gzexe	Wird zum Erzeugen von selbstentpackenden ausführbaren Dateien verwendet
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 `.gz`-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—`.Z` zu `.gz`

6.47. Man-1.5o

Man enthält Programme zum Finden und seitenweisen Anzeigen von Hilfeseiten (Man-pages).

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1.9MB

Man ist abhängig von: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make und Sed

6.47.1. Installation von Man

Zuerst nehmen Sie drei Anpassungen an den Quellen vor.

Der erste Patch verhindert ein Problem, wenn Man-pages mit mehr als 80 Zeichen Zeilenlänge im Zusammenhang mit neueren Groff-Versionen formatiert werden:

```
patch -Np1 -i ../man-1.5o-80cols-1.patch
```

Der zweite Patch fügt der Variable `PAGER` die Option *`-R`* hinzu. Dadurch kann Less Escape-Sequenzen korrekt behandeln:

```
sed -i 's@-is@&R@g' configure
```

Der dritte Patch kommentiert die Zeile „MANPATH /usr/man“ in `man.conf` aus. Das verhindert redundante Ergebnisse, wenn Programme wie zum Beispiel **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 diese Option sucht das Programm **man** seine Konfigurationsdatei `man.conf` im Ordner `/etc`.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Anmerkung

Falls Sie SGR-Escape-Sequenzen (Select Graphic Rendition) abschalten möchten, müssen Sie die Datei `man.conf` editieren und das Argument `-c` zu `NROFF` hinzufügen.

Wenn der Zeichensatz 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. Der Grund dafür ist der folgende: 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** davon aus, dass sie nach 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 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 Mehrbyte-Zeichensätzen. Zu dem ersten Problem gibt es derzeit keine Lösung. Und das zweite Problem wird nicht behandelt, weil LFS keine Mehrbyte-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*.

6.47.2. 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. Es liest alle Man-pages 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

6.48. Make-3.80

Das Paket Make enthält Programme zum Kompilieren umfangreicher Pakete.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 8.8 MB

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

6.48.1. Installieren von Make

Bereiten Sie Make zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

6.48.2. 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

6.49. Module-Init-Tools-3.0

Das Paket Module-Init-Tools enthält diverse Programme zur Verwaltung von Kernel-Modulen für Kernelversionen >= 2.5.47.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 650 KB

Module-Init-Tools ist abhängig von: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make und Sed

6.49.1. Installation von Module-Init-Tools

Bereiten Sie Module-Init-Tools zum Kompilieren vor:

```
./configure --prefix="" --enable-zlib
```

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

6.49.2. Inhalt von Module-Init-Tools

Installierte Programme: depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (Link auf insmod), kernelversion, ksyms (Link auf insmod), 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.

genksyms	Erzeugt Symbol-Versionsinformationen
insmod	Installiert ein ladbares Modul in den laufenden Kernel
insmod_ksymoops_clean	Löscht gespeicherte ksyms und Module, auf die seit zwei Tagen nicht zugegriffen wurde
kallsyms	Extrahiert zu Debuggingzwecken alle Kernelsymbole
kernelversion	Gibt die Hauptversionsnummer des laufenden Kernel aus
ksyms	Zeigt die exportierten Kernelsymbole an
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	Entläd ein Modul aus dem laufenden Kernel

6.50. Patch-2.5.4

Das Paket Patch enthält ein Programm zum Modifizieren von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1.9 MB

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

6.50.1. Installieren 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
```

6.50.2. 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.

6.51. Procps-3.2.3

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: 6.2 MB

Procps ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Make und Ncurses

6.51.1. Installation von Procps

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.51.2. 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 die obersten CPU-Prozesse an. 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.

6.52. Psmisc-21.5

Das Paket Psmisc enthält Programme zum Anzeigen von Prozessinformationen.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.2 MB

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

6.52.1. Installation von Psmisc

Bereiten Sie Psmisc zum Kompilieren vor:

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

Die Bedeutung der configure-Option:

`--exec-prefix=""`
: Dadurch werden die Binärdateien in `/bin`, und nicht in `/usr/bin` installiert. Da die Programme aus dem Paket Psmisc häufig in Bootskripten verwendet werden, müssen sie auch verfügbar sein, falls das `/usr`-Dateisystem noch nicht eingehängt ist.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Es gibt keinen Grund, warum `pstree` und `pstree.x11` in `/bin` liegen müssen. Daher verschieben Sie sie nach `/usr/bin`. Ebenso muss `pstree.x11` nicht als separates Programm existieren, daher machen Sie daraus einen symbolischen Link auf **pstree**:

```
mv /bin/pstree* /usr/bin
ln -sf pstree /usr/bin/pstree.x11
```

In der Voreinstellung wird Psmisc's Programm **pidof** nicht installiert. Das ist normalerweise kein Problem weil wir später das Sysvinit Paket 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 -s killall /bin/pidof
```

6.52.2. 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

6.53. Shadow-4.0.4.1

Das Paket Shadow enthält Programme zur sicheren Verwaltung von Kennwörtern.

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 11 MB

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

6.53.1. Installation von Shadow

Bereiten Sie Shadow zum Kompilieren vor:

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

Umgehen Sie ein Problem mit der Internationalisierung von Shadow:

```
echo '#define HAVE_SETLOCALE 1' >> config.h
```

Shadow deklariert die Funktion malloc() falsch, was zu Kompilierproblemen führt. Beheben Sie das Problem:

```
sed -i '/extern char/d' libmisc/xmalloc.c
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Shadow benutzt zwei Dateien zur Einrichtung der Authentifizierungseinstellungen. Installieren Sie diese beiden Konfigurationsdateien:

```
cp etc/{limits,login.access} /etc
```

Sie sollten die voreingestellte *crypt*-Methode zu *MD5* ändern, welche theoretisch sicherer ist. Ausserdem erlaubt sie Passwörter mit mehr als 8 Zeichen. Desweiteren müssen Sie den alten Standort der Benutzermailboxen von `/var/spool/mail` zu `/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 den Befehl auszuführen):

```
cp etc/login.defs.linux /etc/login.defs
sed -i -e 's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
    -e 's@/var/spool/mail@/var/mail@' /etc/login.defs
```

Verschieben Sie ein paar Links/Programme an ihre korrekte Stelle:

```
mv /bin/sg /usr/bin
mv /bin/vigr /usr/sbin
mv /usr/bin/passwd /bin
```

Und verschieben Sie Shadow's dynamische Bibliotheken an eine passendere Stelle:

```
mv /usr/lib/lib{shadow,misc}.so.0* /lib
```

Weil einige Pakete die gerade verschobenen Bibliotheken in `/usr/lib` erwarten, erstellen Sie die folgenden symbolischen Links:

```
ln -sf ../../lib/libshadow.so.0 /usr/lib/libshadow.so
ln -sf ../../lib/libmisc.so.0 /usr/lib/libmisc.so
```

Die Option *-D* zu **useradd** benötigt diesen Ordner um korrekt zu funktionieren:

```
mkdir /etc/default
```

Coreutils hat das Programm **groups** bereits in `/usr/bin` installiert. Wenn Sie möchten, können Sie die von Shadow installierte Version wieder löschen:

```
rm /bin/groups
```

6.53.2. 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. xdm, ftp und pop3 Server), müssen *shadow-konform* sein. Das heißt, sie müssen mit Shadow-Passwörtern umgehen können.

Um Shadow-Passwörter zu aktivieren, benutzen Sie das folgende Kommando:

```
pwconv
```

Und um Shadow-Gruppenpasswörter zu aktivieren, benutzen Sie das folgende 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.

6.53.3. Vergeben des Passworts für root

Wählen Sie ein Kennwort für den Benutzer *root* und setzen Sie es mit dem Kommando:

```
passwd root
```

6.53.4. Inhalt von Shadow

Installierte Programme: chage, chfn, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, 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 benutzt, um den vollständigen Namen und ein paar andere Informationen eines Benutzers zu ändern

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

groups Zeigt die Gruppenzugehörigkeit eines Benutzers an

grpck Prüft die Integrität der Gruppen-Dateien `/etc/group` und `/etc/gshadow`

grpconv Erzeugt oder aktualisiert die group-Datei von Shadow aus der normalen group-Datei

grpunconv	Aktualisiert `/etc/group` aus `/etc/gshadow` 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 Dämon, 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 `/etc/passwd` und `/etc/shadow`
pwconv	Erzeugt oder aktualisiert die Shadow-Passwort-Datei aus der normalen Passwort-Datei
pwunconv	Aktualisiert `/etc/passwd` aus `/etc/shadow` 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
vigr	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

6.54. Sysklogd-1.4.1

Die in Sysklogd enthaltenen Programme dienen zum Aufzeichnen von Systemmeldungen, zum Beispiel die des Kernels.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 0.5 MB

Sysklogd ist abhängig von: Binutils, Coreutils, GCC, Glibc und Make

6.54.1. Installation von Sysklogd

Sysklogd hat Probleme mit den Header-Dateien der 2.6er Kernelserie. Beheben Sie diese Probleme mit diesem Patch:

```
patch -Np1 -i ../sysklogd-1.4.1-kernel_headers-1.patch
```

Ausserdem gibt es eine sog. Race condition in der Logik der Signalbehandlung; das kann manchmal das **sysklogd** Initskript durcheinander bringen. Beheben Sie diesen Fehler mit einem weiteren Patch:

```
patch -Np1 -i ../sysklogd-1.4.1-signal-1.patch
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

6.54.2. Einrichten von Sysklogd

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

```
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 *

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

6.54.3. Inhalt von Sysklogd

Installierte Programme: klogd und syslogd

Kurze Beschreibungen

klogd Ein Systemdämon zum Abfangen und Protokollieren von Kernelnachrichten

syslogd Protokolliert Meldungen, die von Systemprogrammen zum Protokollieren angeboten werden

6.55. Sysvinit-2.85

Das Sysvinit Paket enhält Programme, mit denen Sie das Starten, Ausführen und Beenden des Systems kontrollieren können.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 0.9 MB

Sysvinit ist abhängig von: Binutils, Coreutils, GCC, Glibc und Make

6.55.1. Installation von Sysvinit

Sysvinit-2.85 enthält einen „Pufferüberlauf-Fehler“. Unter bestimmten Umständen ändert es Werte aus Umgebungsvariablen. Beheben Sie das Problem:

```
patch -Np1 -i ../sysvinit-2.85-proclen-1.patch
```

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. Das sieht aber so aus, als ob init diese Signale an alle laufenden Programme sendet. 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
```

6.55.2. Einrichten von Sysvinit

Erstellen Sie die neue Datei `/etc/inittab` indem Sie das folgende Kommando eingeben:

```
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'`* sorgt dafür, dass **agetty** als erstes diese Escape-Sequenz an das Terminal sendet. Diese Sequenz schaltet die Konsole auf einen benutzerdefinierten Zeichensatz um; dieser kann mit **setfont** eingestellt werden. Die Konsole-Initskripte aus dem LFS-Bootskript-Paket 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.

6.55.3. 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 Schalter *-h* auf, ausser wenn der aktuelle Runlevel 0 ist, dann teilt es dem Kernel mit, das System anzuhalten. Vorher notiert es in `/var/log/wtmp`, 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 `/var/log/wtmp` durchsucht. Es kann auch Systemstarts und -stopps sowie Wechsel der Runlevel zeigen.

lastb Zeigt die letzten fehlgeschlagenen Login-Versuche, die in `/var/log/btmp` protokolliert wurden

mesg Kontrolliert, welche anderen Benutzer Nachrichten auf das aktuelle Terminal senden können

pidof Gibt die PIDs eines Programms aus

poweroff Weist den Kernel an, das System anzuhalten und den Computer auszuschalten. Schauen Sie auch nach **halt**.

reboot Weist den Kernel an, das System neu zu starten. Schauen Sie auch nach **halt**.

runlevel Zeigt den vorigen und den aktuellen Runlevel an. Entnimmt die Information aus `/var/run/utmp`.

shutdown Fährt das System sicher herunter, sendet entsprechende Signale an alle Prozesse und benachrichtig alle angemeldeten Benutzer

sulogin Erlaubt *root*, 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

6.56. Tar-1.14

Das Paket Tar enthält ein Archivprogramm.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 10 MB

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

6.56.1. Installieren von Tar

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 das Kommando **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

6.56.2. Inhalt von Tar

Installierte Programme: rmt und tar

Kurze Beschreibungen

rmt Wird zum entfernten Manipulieren von magnetorientierten Bandlaufwerken verwendet und benutzt dafür Interprozesskommunikation

tar Wird zum Erzeugen und Extrahieren von Dateien aus einem Archiv verwendet

6.57. Udev-030

Das Paket Udev enthält Programme zum dynamischen Erzeugen von Gerätedateien.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 5.2 MB

Udev ist abhängig von: Coreutils und Make

6.57.1. Installation von Udev

Kompilieren Sie das Paket:

```
make udevdir=/dev
```

udevdir=/dev
Dieser Parameter gibt an, in welchem Ordner **udev** Gerätedateien erzeugen soll.

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make udevdir=/dev install
```

Udev's Konfigurationsdateien sind alles andere als optimal, installieren Sie daher diese Konfigurationsdateien:

```
cp ../udev-config-2.permissions \
    /etc/udev/permissions.d/25-lfs.permissions
cp ../udev-config-1.rules /etc/udev/rules.d/25-lfs.rules
```

6.57.2. 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)

udevd Ein Daemon, der Hotplug-Ereignisse umsortiert bevor er sie an **udev** weiterreicht. Damit werden bestimmte sog. Race conditions verhindert.

udevsend Übermittelt Hotplut-Ereignisse an **udevd**

udevstart Erzeugt Gerätedateien in `/dev` zu Gerätetreibern, die fest in den Kernel einkompiliert sind. Es erfüllt diese Aufgabe, indem es die Hotplug-Ereignisse simuliert, 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 `sysfs`-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

`/etc/udev` Enthält Konfigurationsdateien, Geräteberechtigungen und Regeln für die Namensvergabe von **udev**

6.58. Util-linux-2.12b

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

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 16 MB

Util-linux ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed und Zlib

6.58.1. 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
```

6.58.2. Installieren von Util-linux

GCC-3.4.1 kompiliert **sfdisk** mit der Standard-Optimierung fehlerhaft. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../util-linux-2.12b-sfdisk-2.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:

```
make HAVE_KILL=yes HAVE_SLN=yes install
```

6.58.3. 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 gennant) benutzt
ipcrm	Entfernt die angegebene IPC-Ressource (Inter-Process Communication)
ipcs	Gibt IPC Status-Informationen aus
isosize	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 ein Dateisystem auf einem Gerät an einem Ordner in der Ordnerstruktur 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
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 einer Binärdatei, der Quellen und der Man-page für ein Kommando an
write	Sendet eine Nachricht an einen Benutzer (sofern der Benutzer den Empfang solcher Nachrichten nicht deaktiviert hat)

6.59. 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. Um einen Eindruck über den von Debugging-Symbolen belegten Speicher zu bekommen, schauen Sie sich dies an:

- 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 im Regelfall zwischen 2 und 5.

Da die meisten Leute vermutlich niemals einen Debugger mit ihrer Systemsoftware einsetzen, kann hier durch das Entfernen der Debugging-Symbole eine Menge Platz gespart werden. 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 einer Anleitung unter *http://www.linuxfromscratch.org/hints/downloads/files/ optimization.txt*.

6.60. Erneutes Stripping

Die meisten werden vermutlich niemals einen Debugger mit der Systemsoftware einsetzen; Sie können hier ca. 200MB Platz sparen, indem Sie die Debugging-Symbole entfernen. Das verursacht keine Probleme, ausser das Sie die Software danach nicht mehr vollständig debuggen können.

Normalerweise gibt es mit dem untenstehenden Kommando keine Probleme. Dennoch könnte man einen Tippfehler machen und dadurch das System unbrauchbar machen. Bevor Sie also das **Strip**-Kommando ausführen, sollten Sie ein Backup anlegen.

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.

6.61. Aufräumen

Wenn Sie von nun an die chroot-Umgebung verlassen und wieder betreten möchten, sollten Sie das folgende, neue chroot-Kommando 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 dem Ordner `/tools` benötigen, Sie können den Ordner nun löschen und die chroot-Umgebung erneut mit dem obigen Kommando betreten. 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

Beim Löschen von `/tools` werden auch die temporären Kopien von Tcl, Expect und DejaGNU gelöscht, welche Sie zum Testen der Toolchain benutzt haben. Wenn Sie diese Programme später noch benutzen möchten, müssen Sie sie neu kompilieren und installieren. Die Installationsanweisungen sind identisch mit denen in Kapitel 5, allerdings müssen Sie den Prefix von `/tools` auf */usr* abändern. Das BLFS-Buch geht einen anderen Weg zur Installation von Tcl, schauen Sie auch hier nach: *http://www.linuxfromscratch.org/blfs/*.

Eventuell möchten Sie die Pakete und Patche aus `/sources` an eine üblichere Stelle verschieben, wie z. B. `/usr/src/packages`. Danach können Sie den Ordner dann löschen. Oder Sie löschen den Ordner sofort, wenn Sie alles auf CD gebrannt haben.

Kapitel 7. Aufsetzen der System-Bootskripte

7.1. Einführung

Dieses Kapitel setzt die Bootskripte auf. Die meisten Skripte funktionieren ohne Anpassungen, aber ein paar benötigen eine Grundeinstellung weil sie zum Beispiel Hardwareabhängig sind.

In diesem Buch werden Init-Skripte im System-V-Stil verwendet, weil sie sehr gebräuchlich sind. Es gibt auch andere Möglichkeiten, lesen Sie z. B. für BSD-Stil-Init unter *http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt* nach. Oder durchsuchen Sie die LFS-Mailinglisten nach „depinit“ um eine andere Möglichkeit zu versuchen.

Wenn Sie sich für etwas ganz anderes entschieden haben, können Sie das nachfolgende Kapitel überspringen und direkt bei Kapitel 8 fortfahren.

7.2. LFS-Bootscripts-2.2.2

Das Paket LFS-Bootskripte enthält einige Boot-Skripte.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 0.3 MB

LFS-Bootscripts 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, ifdown, ifup, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, template und udev

Kurze Beschreibungen

checkfs	Prüft Dateisysteme bevor sie eingehängt werden (mit der Ausnahme von journal- und netzwerkbasierten Dateisystemen)
cleanfs	Entfernt Dateien, die nicht über das Neustarten des Systems hinaus existieren sollten, wie zum Beispiel die in `/var/run/` und `/var/lock/`. Es erzeugt `/var/run/utmp` und entfernt die eventuell vorhandenen Dateien `/etc/nologin`, `/fastboot` und `/forcefsck`.
console	Läd das für Ihre Tastatur eingestellte Tastaturlayout
functions	Enthält Funktionen, die gemeinsam von verschiedenen Skripten genutzt werden, wie z. B. Fehler- oder Statusprüfung
halt	Hält das System an
ifdown	Unterstützt das Netzwerkskript bei Netzwerkgeräten

ifup Unterstützt das Netzwerkskript bei Netzwerkgeräten

localnet Setzt den Hostnamen und das lokale Loopback-Gerät auf

mountfs Hängt alle nicht als *noauto* markierten und nicht netzwerkbasierten Dateisysteme ein

mountkernfs Wird zum Einhängen von Kernel-basierten Dateisystemen, wie z. B. `proc`, verwendet

network Macht Netzwerkschnittstellen wie z. B. Netzwerkkarten verfügbar und richtet - wenn nötig - das Standard-Gatway 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-Dämonen

template Eine Vorlage, die Sie verwenden können, um Ihre eigenen Bootskripte für eigene Dämonen zu schreiben

udev Initialisiert udev und erzeugt die Gerätedateien in `/dev`

7.3. Wie funktionieren diese Bootskripte?

Linux benutzt eine spezielle Booteinrichtung mit dem Namen SysVinit. Es basiert auf dem Konzept der *Runlevel*. Dieses Konzept kann von System zu System stark variieren. Man kann nicht einfach annehmen, weil Dinge in <hier Distributionsnamen einsetzen> funktionieren, tun sie das auch in LFS. LFS hat seinen eigenen Weg, wie diese Dinge funktionieren, aber grundsätzlich respektieren wir die allgemein üblichen Standards.

SysVinit (wir nennen es nun einfach nur „init“) funktioniert nach dem Runlevel-Schema. Es gibt 7 Runlevel (von 0 bis 6), genaugenommen gibt es sogar mehr, aber diese sind für Spezialfälle reserviert und werden üblicherweise nicht benutzt. Die Man-page von init beschreibt diese Details genauer. Jeder Runlevel korrespondiert mit Dingen, die der Computer beim Hochfahren ausführen soll. Der Standard-Runlevel ist 3. Hier ist eine Beschreibung, wie die verschiedenen 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 kd
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 `rcsysinit.d`, welches einige symbolische Links enthält. 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, von 00 bis 99. 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.

Die echten Skripte befinden sich in `/etc/rc.d/init.d`. Sie erledigen die ganze Arbeit, und die ganzen symbolischen Links zeigen auf sie. Stopp- und Startskripte zeigen auf dieselbe Datei in `/etc/rc.d/init.d`. Das funktioniert, weil die Skripte mit

unterschiedlichen Parametern ausgeführt werden können, wie 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 Sinn dahinter ist, dass Sie wohl kaum einen Dienst starten wollen, wenn Sie rebooten oder das System anhalten wollen.

Hier einige 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 auch Ihr eigenes LFS-System). Die Dateien hier sind nur Beispiele, wie man es gut erledigen kann.

7.4. Umgang mit Geräten und Modulen an einem LFS-System

In Kapitel 6 haben Sie das Paket Udev installiert. Bevor wir zu den Details kommen, wie das alles funktioniert, ein kleiner Rückblick, wie früher mit Geräten umgegangen wurde.

Traditionell benutzte Linux eine statische Methode, bei der sehr viele Gerätedateien vorab unter `/dev` installiert wurden (manchmal mehrere tausend). Dabei war es völlig egal, ob die dazugehörige 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 möglichen Geräten erzeugte, die es auf der Welt gibt. 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, werden sie auf einem `ramfs`-Dateisystem gespeichert. Das ist ein Dateisystem, das nur im Arbeitsspeicher existiert und keinen Festplattenplatz verbraucht. Gerätedateien benötigen kaum Platz, so dass nur sehr wenig Arbeitsspeicher verbraucht wird.

7.4.1. Geschichtliches

Im Februar 2000 wurde ein neues Dateisystem mit dem Namen `devfs` in den Kernel 2.3.46 integriert und wurde 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 der Weg, wie 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 zusammenhingen und nicht ohne nennenswerte Änderungen am Kernel geändert werden konnten. Ausserdem wurde es als „missbilligt“ markiert, weil es aktuell nicht mehr gepflegt wurde.

Mit der Enticklung 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 die Anwenderprozesse zu exportieren. Mit dieser aus der Anwenderschicht 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 in einem System weis und welche Gerätenummern verwendet werden müssen. Treiber, die direkt in den Kernel integriert wurden, registrieren sich am `sysfs` sobald sie vom Kernel erkannt werden. Bei Kernel-Modulen geschieht dieser Vorgang beim Laden des Moduls. Sobald `sysfs` (unter `/sys`) in das System eingehängt ist, 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. Das Skript beginnt damit, **/sbin/udev** als Programm zur Verarbeitung von Hotplug-Ereignissen zu registrieren. In dieser Phase sollten noch keine Hotplug-Ereignisse (wird weiter unten erklärt) generiert werden, aber **udev** wird für den Fall der Fälle doch schon mal registriert, falls doch Ereignisse auftreten. 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 Hohen Nummer *7* und der niedrigen Nummer *0* zu erzeugen. Die Rechte eines jeden Gerätes entnimmt **udevstart** den Dateien aus dem Ordner `/etc/udev.d/permissions.d/`. Diese Dateien sind ähnlich durchnummeriert wie die LFS Bootskripte. Falls **udev** für eine Gerätedatei keine Datei mit Berechtigungen finden kann, wird die Voreinstellung *600* und der Besitzer *root:root* festgelegt. Die Namen der Gerätedateien in `/dev` werden durch Regeln aus `/etc/udev/rules.d/` festgelegt.

Sobald die obige Phase abgeschlossen ist, sind alle Geräte die schon existierten und für die Treiber in den Kernel fest eingebaut sind, verfügbar und können benutzt werden. 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 **udev** als Programm zur Verarbeitung von Hotplug-Ereignissen registriert. Wenn also solche Hotplug-Ereignisse erzeugt werden, teilt der Kenel **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 Ihre Einträge in `/etc/modules.conf`. Das Laden eines Moduls wird durch den Zugriff auf die zugehörige Gerätedatei veranlasst. 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.

Beachten Sie, dass manche Treiber sehr viele Gerätedateien erzeugen. Auf langsamen Maschinen kann dieser Vorgang ein paar Sekunden dauern. Das bedeutet, dass einige Geräte nicht sofort verfügbar sind.

7.4.3. Umgang mit Hotplug-Geräten/dynamischen 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 bereits 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 als Modul verfügbar, aber zur Zeit noch nicht geladen ist, dann wird der Kernel nur ein Hotplug-Ereignis erzeugen, welches die Anwenderschicht über das neue Gerät informiert und das es noch keinem Treiber zugeordnet ist. Sonst passiert nichts weiter und das Gerät ist noch nicht benutzbar.

Wenn Sie ein System erstellen, das sehr viele Treiber als Modul vorhält, dann ist die Methode über **S05modules** nicht sehr praktikabel. In diesem Fall filft das Paket Hotplug weiter (zu finden unter *http://linux-hotplug.sourceforge.net/*) Wenn das Hotplug-Paket installiert ist, reagiert es auf die zuvor erwähnten Hotplug-Ereignisse des Bus-Treibers. Hotplug wird die nötigen Module laden und die Geräte durch Erstellen der entsprechenden Gerätedateien zur Verfügung stellen.

7.4.4. Probleme mit dem Erzeugen von Gerätedateien

Es gibt ein paar bekannte Probleme beim automatisierten Erzeugen von Gerätedateien:

1) Ein Kerneltreiber 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. Für dieses Treiber wird schlussendlich keine Gerätedatei erzeugt. Verwenden Sie die Datei `/etc/sysconfig/createfiles` um die 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 meist 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/ R*
- 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*

7.5. Einrichten des setclock-Skripts

Das Skript **setclock** liest die Zeit aus der Hardware-Uhr des Computers (auch bekannt als BIOS- oder CMOS- (Complementary Metal Oxide Semiconductor) 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 das selber einrichten.

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 diese Zeit 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; sie erklärt sehr gut, wie man unter LFS mit der Systemzeit umgeht. Sie erklärt Dinge wie Zeitzonen, UTC und die Umgebungsvariable `TZ`.

7.6. Einrichten der Linux Konsole

Dieser Abschnitt behandelt das **console** Intitskript, 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 das console Initskript einfach nichts.

Das Skript **console** benutzt `/etc/sysconfig/console` als Konfigurationsdatei. Entscheiden Sie, welche Tastaturbelegung und Bildschirmschriftarten Sie benutzen werden. Eine vorgefertigte Version von `/etc/sysconfig/console` mit vielen bekannten Einstellungen für verschiedene Länder wurde bereits mit dem Bootskripts-Paket mitinstalliert. Sie können den relevanten Abschnitt in der Datei einfach auskommentieren, falls Ihr Land bereits unterstützt wird. Falls Sie Bedenken haben, schauen Sie in `/usr/share/kbd` nach gültigen Tastaturbelegungen und Schriftarten. Lesen Sie die Man-pages zu **loadkeys** und **setfont** und bestimmten 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 Spanische Anwender, die das Euro-Symbol benutzen (erreichbar mit AltGr+E), diese Einstellungen richtig:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="es euro2"
FONT="lat9-16 -u iso01"
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**-Initskript nicht das zugehörige Programm aus.

Bei manchen Tastaturbelegungen senden die Backspace- und Löschen-Tasten andere Tastenkodes 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 Kodeschnipsel um das Problem zu lösen:

```
mkdir -p /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 Kodeschnipsel nach dem Laden der Haupt-Tastaturbelegungstabelle zu laden:

```
cat >>/etc/sysconfig/console <<"EOF"
KEYMAP_CORRECTION="/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-2.6.8.1“. Damit stellen Sie sicher, dass die Tastatur immer richtig funktioniert, auch 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. Eigentlich sollte das allerdings keine Probleme verursachen, weil man im Wartungsmodus üblicherweise nicht auf Sonderzeichen angewiesen ist.

Da der Kernel die Tastaturzuweisungstabellen läd, 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 jeden Kernel die Tastaturzuweisungstabelle fest einkompilieren möchten.

7.7. Erstellen der Datei /etc/inputrc

Die Datei `/etc/inputrc` hat mit der Zuweisung von Tasten in bestimmten Situationen zu tun. Sie ist die erste von Readline benutzte Datei. Readline ist eine Bibliothek, die Eingabe-Funktionen für Bash und die meisten anderen Shells zur Verfügung stellt.

Weitere Informationen finden Sie in der info-Seite zu Bash, Abschnitt *Readline Init File*. Die info-Seite zu Readline ist ebenfalls eine gute Informationsquelle.

Globale Einstellungen werden in `/etc/inputrc` vorgenommen. Benutzerdefinierte Einstellungen können in `~/.inputrc` gemacht werden. Änderungen in `~/.inputrc` überschreiben die Einstellungen der globalen Datei. In kürze wird Bash so eingerichtet, dass sie `/etc/inputrc` benutzt, falls die persönliche Datei `.inputrc` nicht existiert während `/etc/profile` eingelesen wird (üblicherweise beim Anmelden). Damit das System beide Dateien benutzt, sollte sinnvollerweise für neue Benutzer eine Standard-`.inputrc` in `/etc/skel` abgelegt werden.

Unten finden Sie eine Basisversion von `/etc/inputrc`, zusammen mit erklärenden Kommentaren zu den verschiedenen Optionen. Beachten Sie bitte, dass sich Kommentare nicht in der gleichen Zeile wie Kommandos befinden dürfen.

Um die Datei `.inputrc` in `/etc/skel` zu erzeugen, leiten Sie die Ausgabe des Kommandos bitte nach `/etc/skel/.inputrc` um und stellen anschließend die richtigen Rechte ein. Kopieren Sie diese Datei dann nach `/etc/inputrc` und in die Persönlichen Ordner der bereits bestehenden Benutzer. Benutzen Sie die Option *-p* zu **cp** um die Rechte mitzukopieren und stellen Sie die Besitzer und Gruppen korrekt ein.

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Make sure we don't output everything on the 1 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
```

7.8. Die Startdateien von Bash

Das Shell-Programm **/bin/bash** (weiterhin nur als „shell“ bezeichnet) benutzt einige Startdateien zum Einrichten der Benutzerumgebung. Jede Datei hat einen bestimmten Zweck und beeinflusst Login- und Interaktiv-Umgebungen unterschiedlich. Die Dateien in `/etc` enthalten globale Einstellungen. Wenn eine gleichwertige Datei auch im Persönlichen Ordner des Benutzers existiert, überschreibt diese 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.

Weiter Informationen finden Sie mit **info bash** - Abschnitte: Bash Startup Files und 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 Kassifizierung von Zeichen in Buchstaben, Zahlen und andere Klassen. Die Bash benötigt diese Einstellungen, um Sonderzeichen in Befehlszeilen in nicht-englischen Locales verarbeiten zu können.
- Korrekte alphabetische Sortierung für jedes Land
- Passende Papiergröße
- Korrekte Formatierung von Währungs-, Zeit- und Datumswerten

Dieses Skript setzt auch die Variable `INPUTRC`. Wenn diese Variable gesetzt ist, benutzen Bash und Readline die vorher erzeugte Datei `/etc/inputrc`.

Ersetzen Sie *[ll]* mit dem zweistelligen Länderkode für die gewünschte Sprache (z. B. „de“) und *[CC]* mit dem zweistelligen Kode für das gewünschte Land (z. B. „DE“ oder „AT“). Evtl. ist es sogar nötig (und das ist die bevorzugte Schreibweise), die Kodierung des Zeichensatzes nach einem Punkt anzufügen (z. B. „de_DE.iso8859-15“). Geben Sie das folgende Kommando ein um mehr Informationen zu erhalten:

```
man 3 setlocale
```

Mit dem folgenden Kommando erhalten Sie eine Liste aller von Glibc unterstützten Locales:

```
locale -a
```

Wenn Sie die korrekten Locale-Einstellungen herausgefunden haben, erstellen Sie die Datei `/etc/profile`:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

export LC_ALL=[ll]_[CC]
export LANG=[ll]_[CC]
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-basierte Locales) erfordern weitere Schritte und Patches, weil viele Anwendungen dazu neigen, unter diesen Bedingungen nicht richtig zu funktionieren. Diese Schritte und Patches sind nicht Teil des LFS-Buches und diese Locales werden grundsätzlich von LFS nicht unterstützt.

7.9. Einrichten des sysklogd-Skript

Das `sysklogd`-Skript startet das Programm **syslogd** mit der Option *`-m 0`*. Diese Option schaltet die periodische Marke ab, die **syslogd** in der Voreinstellung alle 20 Minuten in die Logdateien schreibt. Wenn Sie diese Zeitmarke einschalten wollen, editieren Sie das Skript `sysklogd` entsprechend. Weitere Informationen erhalten Sie mit **`man syslogd`**.

7.10. Einrichten des localnet-Skript

Eine Teilaufgabe des localnet-Skript ist das Einstellen des Hostnamen. Dies muss in `/etc/sysconfig/network` eingestellt 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 Domänennamen) ein. Diese Information wird erst später in `/etc/hosts` eingetragen.

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` einstellen. 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 172.31.0.0
   C     192.168.0.0 bis 192.168.255.0
```

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).

Auch wenn Sie keine Netzwerkkarte verwenden, brauchen Sie dennoch einen voll qualifizierten Domänennamen. Das ist nötig, damit einige Programme korrekt arbeiten können.

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-Skript

Dieser Arbschnitt ist nur interessant, wenn Sie eine Netzwerkkarte einrichten möchen.

Wenn Sie keine Netzwerkkarte haben, brauchen Sie höchstwahrscheinlich keine Konfigurationsdateien bezüglich Netzwerkkarten einrichten. Falls das der Fall ist, müssen Sie alle symbolischen Links mit Namen `network` aus allen 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 in `/etc/sysconfig/network-devices` ab. Dieser Ordner sollte Dateien der Form `ifconfig.xyz` enthalten, wobei „xyz“ der Name eines Netzwerkgerätes ist (wie zum Beispiel eth0 oder eth0:1)

Wenn Sie den Ordner `/etc/sysconfig/network-devices` umbenennen oder verschieben möchten, aktualisieren Sie auch in der Datei `/etc/sysconfig/rc` den Pfad zu „network_devices“.

Nun erzeugen Sie neue Dateien. Das folgende Kommando erzeugt die Beispielhafte `ipv4`-Datei für *eth0*:

```
cd /etc/sysconfig/network-devices &&
mkdir 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 der 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 stellt ein, wie die IP-Adresse vergeben wird. Die LFS-Bootskripte sind in Bezug auf IP-Adressen Zuweisung 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 DHCP zu nutzen (wird im BLFS-Buch beschrieben).

Die Variable GATEWAY sollte die IP-Adresse des Standard-Gateway enthalten. Wenn Sie kein Standard-Gateway haben, setzen Sie ein Kommentarzeichen vor die Zeile.

Die Variable PREFIX muss die Anzahl der verwendeten Bits in der Netzwerkmaske enthalten. Jedes Oktett hat acht Bit. Wenn die Netzwerkmaske 255.255.255.0 ist, dann werden die ersten drei Oktette benutzt (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 entsprechend Ihrem Subnetz an.

7.12.2. Erstellen der Datei /etc/resolv.conf

Wenn Sie mit dem Internet verbunden sind, brauchen Sie höchstwahrscheinlich DNS-Namensauflösung um Internet Domänennamen 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`, erstellen einen neuen Kernel für Ihr LFS-System und 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 Programm benutzt, um festzustellen, wo und in welcher Reihenfolge Partitionen eingehängt werden sollen und welche Dateisysteme geprüft werden müssen. Erstellen Sie eine neue Dateisystemtabelle:

```
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**.

Wenn Sie eine reiserfs-Partition verwenden, sollten Sie *`1 1`* am Ende der Zeile durch *`0 0`* ersetzen, weil eine solche Partition nicht geprüft werden muss.

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 sehr wenig Software POSIX shared memory verwendet. 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).

8.3. Linux-2.6.8.1

Das Linux-Paket enthält den Kernel und die Header-Dateien.

Geschätzte Kompilierzeit: 4.20 SBU
Ungefähr benötigter Festplattenplatz: 181 MB

Linux 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 nur ein paar Schritte—Konfigurieren, kompilieren und installieren. Wenn Sie die Methode der Installation in diesem Buch nicht mögen, schauen Sie in der `README`-Datei im Kernel Quellordner nach alternativen Methoden.

Bereiten Sie den Kompiliervorgang 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.

Ausserdem müssen Sie sicherstellen, dass der Kernel keine Hotplug-Ereignisse in die Anwenderschicht sendet, bevor diese nicht bereit ist:

```
sed -i 's@/sbin/hotplug@/bin/true@' kernel/kmod.c
```

Wenn Sie sich in Abschnitt 7.6, „Einrichten der Linux Konsole“ entschieden haben, die Tastaturzuweisungstabelle in den Kernel einzukompilieren, dann führen Sie dieses Kommando aus:

```
loadkeys -m /usr/share/kbd/keymaps/[Pfad zu keymap] > \
    drivers/char/defkeymap.c
```

Wenn Sie zum Beispiel eine holländische Tastatur haben, würden Sie `/usr/share/kbd/keymaps/i386/ qwerty/nl.map.gz` benutzen.

Richten Sie den Kernel mit der menügeführten Benutzeroberfläche ein:

```
make menuconfig
```

make oldconfig könnte in einigen Fällen besser geeignet sein. Schauen Sie in die Datei README um mehr Informationen zu erhalten.

Anmerkung

Wenn Sie den Kernel einrichten, schalten Sie „Support for hotpluggable devices“ im Menü „General Setup“ ein. Dadurch werden die von **udev** benötigten Hotplug-Ereignisse gesendet um den Ordner /dev mit Gerätedateien zu füllen.

Wenn Sie möchten, können Sie die Kernelkonfiguration überspringen und einfach die Kernel-Konfigurationsdatei .config von Ihrem Host-System nach linux-2.6.8.1 kopieren (falls sie verfügbar ist). Das wird allerdings nicht empfohlen, Sie sind besser dran, wenn Sie alle Konfigurationsmenüs durchsehen und Ihre eigene Kernelkonfiguration frisch einrichten.

Um Unterstützung für POSIX shared memory zu haben, müssen Sie im Kernel die Option „Virtual memory file system support“ einschalten. Diese finden Sie im Menü „File systems“ und ist üblicherweise eingeschaltet.

Die LFS-Bootskripte gehen davon aus, dass entweder „Support for Host-side USB“ und „USB device filesystem“ direkt in den Kernel einkompiliert wurden, oder nichts von beidem kompiliert wurde. Die Bootskripte werden nicht richtig funktionieren, wenn die Unterstützung als Modul kompiliert wurde (usbcore.ko).

Anmerkung

NPTL setzt voraus, dass der Kernel mit GCC 3.x kompiliert wird, in diesem Fall 3.4.1. Es ist bekannt, dass das Kompilieren mit 2.95.x Fehler in der Glibc Test-suite hervorruft, daher wird nicht empfohlen, den Kernel mit gcc 2.95.x zu kompilieren.

Kompilieren Sie das Kernel-Abbild und die Module:

```
make
```

Wenn Sie Kernel-Module verwenden, brauchen Sie die Datei /etc/modules.conf. Informationen betreffend Module und Kernelkonfiguration im allgemeinen finden Sie in der

Kernel-Dokumentation im Ordner `linux-2.6.8.1/Documentation`. Die Man-page von *modules.conf* könnte für Sie auch von Interesse sein.

Seien Sie vorsichtig, wenn Sie andere Dokumentationen lesen, 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 läd. 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
```

Wenn Sie viele Module aber dafür wenig Festplattenspeicher haben, können Sie die Module strippen und komprimieren. Für die meisten ist das Komprimieren den Aufwand nicht wert, aber wenn Sie wirklich Platzprobleme haben, dann schauen Sie unter *http://www.linux-mips.org/archives/linux-mips/2002-04/msg00031.html*.

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 zu der Kerneldatei variiert, abhängig von der benutzten Plattform auf der Sie arbeiten. Geben Sie das folgende Kommando ein, um den Kernel zu installieren:

```
cp arch/i386/boot/bzImage /boot/lfskernel-2.6.8.1
```

`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 System.map /boot/System.map-2.6.8.1
```

`.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 .config /boot/config-2.6.8.1
```

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 hat dann auch Schreibrechte darauf.

Unter diesem Aspekt möchten Sie vielleicht **chown -R 0:0** auf den Ordner `linux-2.6.8.1` anwenden damit alle Dateien dem Benutzer *root* gehören.

8.3.2. Inhalt von Linux

Installierte Dateien: Kernel, Kernel-Header und System.map

Kurze Beschreibungen

`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.

`Kernel Header` Definieren die Schnittstelle zu den Diensten des Kernels. Die Header in dem Ordner `include` Ihres Systems sollten *immer* diejenigen sein, mit denen die Glibc kompiliert wurde und sollten daher bei einem Kernelupgrade *nicht* ersetzt werden.

`System.map` 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. Als eines der letzten Dinge müssen Sie sicherstellen, dass es booten kann. Die untenstehende Anleitung gilt nur für Computern 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.

Das Booten kann ein 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).

Bereits einige Schritte vorher haben Sie den Grub Bootloader in Vorbereitung zu diesem Schritt installiert. Jetzt müssen ein paar Grub-Dateien an spezielle Orte auf der Festplatte kopiert werden. Bevor Sie dazu kommen, sollten Sie eine Grub-Boot-Diskette 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 ein eigenes Schema der Form *(hdn,m)* zur Benennung von Festplatten und Partitionen, wobei *n* die Nummer der Festplatte, und *m* die Nummer der Partition ist. Beide Werte starten 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 jetzigen 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, welche das Grub Bootmenü definiert:

```
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 6.0
root (hd0,3)
kernel /boot/lfskernel-2.6.8.1 root=/dev/hda4
EOF
```

Vielleicht möchten Sie einen weiteren Eintrag für Ihr Host-System vornehmen. Dieser könnte so aussehen:

```
cat >> /boot/grub/menu.lst << "EOF"
title Red Hat
root (hd0,2)
kernel /boot/kernel-2.4.20 root=/dev/hda3
initrd /boot/initrd-2.4.20
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/*.

Kapitel 9. Das Ende

9.1. Das Ende

Herzlichen Glückwunsch! Sie sind fertig mit der Installation Ihres eigenen LFS-Systems. Wir wünschen Ihnen viel Freude mit Ihrem brandneuen selbstgebauten Linux-System.

Es könnte sinnvoll sein, die Datei `/etc/lfs-release` zu erstellen. Mit dieser Datei 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 6.0 > /etc/lfs-release
```

9.2. Lassen Sie sich zählen

Möchten Sie nun, nachdem Sie das Buch durchgearbeitet haben, als LFS-Benutzer gezählt werden? Dann 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

Lassen Sie uns Ihr System neu starten. Als erstes verlassen Sie die chroot-Umgebung:

```
logout
```

Hängen Sie die virtuellen Dateisysteme aus:

```
umount $LFS/dev/pts
umount $LFS/dev/shm
umount $LFS/dev
umount $LFS/proc
umount $LFS/sys
```

Und hängen Sie das LFS-Dateisystem aus:

```
umount $LFS
```

Falls Sie sich zu Beginn für mehrere Partitionen entschieden haben, müssen Sie auch die anderen Partitionen aushängen, bevor Sie die Hauptpartition aushängen:

```
umount $LFS/usr
umount $LFS/home
umount $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 *LFS 6.0* automatisch booten.

Nach dem Neustart ist Ihr LFS-System bereit, Sie können es nun benutzen und damit beginnen, weitere eigene Software zu installieren.

9.4. Was nun?

Vielen Dank, dass Sie das LFS-Buch gelesen haben. Wir hoffen, dass Sie das Buch nützlich fanden und viel über den Entwicklungsprozess 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.

- 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. Schauen Sie für weitere Informationen unter Kapitel 1 - Mailinglisten nach.

- 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 (Wechseln des Basisordners)

CMOS Complementary Metal Oxide Semiconductor

COS Class Of Service

CPU Central Processing Unit

CRC Cyclic Redundancy Check (Zyklische Redundanzprüfung)

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 (Ende der Datei)

EQN Gleichung

EVMS Enterprise Volume Management System

ext2 second extended-Dateisystem

FAQ Frequently Asked Questions (Häufig gestellte Fragen)

FHS Filesystem Hierarchy Standard

FIFO First-In, First Out

FQDN Fully Qualified Domain Name (Vollständiger Domänen-Name)

FTP File Transfer Protocol

GB Gibabyte

GCC GNU Compiler Collection

GID Group Identifier (Gruppen-ID)

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 (Eingabe/Ausgabe)

IP Internet Protocol

IPC Inter-Process Communication

IRC Internet Relay Chat

ISO International Organization for Standardization

ISP Internet Service Provider

KB Kilobyte

LED Light Emitting Diode

LFS Linux From Scratch

LSB Linux Standards Base

MB Megabyte

MBR Master Boot Record

MD5 Message Digest 5

NIC Network Interface Card (Netzwerkkarte)

NLS Native Language Support

NNTP Network News Transport Protocol

NPTL Native POSIX Threading Library

OSS Open Sound System

PCH Pre-Compiled Headers (Vorkompilierte Header)

PCRE Perl Compatible Regular Expression (Perl-kompatible Reguläre Ausdrücke)

PID Process Identifier (Prozess-ID)

PLFS Pure Linux From Scratch

PTY pseudo terminal

QA Quality Assurance (Qualitätssicherung)

QOS Quality Of Service

RAM Random Access Memory

RPC Remote Procedure Call

RTC Real Time Clock

SBU Static Binutils Unit

SCO Die 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 (Benutzer-ID)

umask Dateierzeugungsmaske

USB	Universal Serial Bus
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
VC	Virtual Console (Virtuelle Konsole)
VGA	Video Graphics Array
VT	Virtual Terminal (Virtuelles Terminal)

Anhang B. Danksagungen

Wir möchten uns bei allen nachfolgenden Personen und Organisationen für ihr Mitwirken und ihre Beiträge zum Projekt Linux From Scratch bedanken.

Derzeitige Projektmitglieder

- *Gerard Beekmans* <gerard@linuxfromscratch.org> – Linux From Scratch Initiator, LFS-Projektbetreuer
- *Christine Barczak* <theladyskye@linuxfromscratch.org> – LFS Buchautorin
- *Matthew Burgess* <matthew@linuxfromscratch.org> – LFS Co-Maintainer, allgemeine Paketbetreuung, LFS Buchautor
- *Craig Colton* <meerkats@bellsouth.net> – LFS, Automated Linux From Scratch (ALFS), BLFS und Ersteller des Logos für das Hint-Projekt
- *Nathan Coulson* <nathan@linuxfromscratch.org> – Betreuer der LFS Bootskripte
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Website-Entwickler, Betreuer der FAQ
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – LFS Qualitätssicherung Teamleiter, BLFS Buchautor
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – LFS XML/XSL Betreuer
- *Jim Gifford* <jim@linuxfromscratch.org> – LFS Buchautor, Betreuer der Patches
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Wiki Betreuer
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Betreuer der Website-Skripte
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – LFS NNTP Gateway Betreuer
- *Ryan Oliver* <ryan@linuxfromscratch.org> – Teamleiter des Test-Teams, Betreuer der Toolchain, Mitbegründer von Pure LFS (PLFS)
- *Alexander Patrakov* <semzx@newmail.ru> – Früherer LFS Buchautor

- *James Robertson* <jwrober@linuxfromscratch.org> – Bugzilla Betreuer, Wiki Entwickler, LFS Buchautor
- *Tushar Teredesai* <tushar@linuxfromscratch.org> – BLFS Buchautor, Betreuer des Hints und Patches Projekt
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – LFS Buchautor, Bugzilla Betreuer, Betreuer der LFS Bootskripte, Co-Administrator des LFS-Servers.
- *Zack Winkles* <zwinkles@gmail.com> – Früherer LFS Buchautor
- 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.org> – 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

- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.org
- *Andres Meggiotto* <sysop@mesi.com.ar> – lfs.mesi.com.ar
- *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

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

Index

Pakete

Programme

Bibliotheken

Skripte

Sonstige