Linux From Scratch

Version 5.1.1

Gerard Beekmans

Linux From Scratch: Version 5.1.1

von Gerard Beekmans

Dieses Buch beschreibt die genaue Vorgehensweise zum Installieren eines Linux-Systems von Grund auf, ausschliesslich unter Verwendung der Quellen aller benötigter Programme.

Widmung

Ich widme dieses Buch meiner mich liebenden und sehr unterstützenden Frau *Beverly Beekmans*.

Inhaltsverzeichnis

Einleitung

Vorwort

Ich habe schon einige Linux-Distributionen benutzt, aber mit keiner war ich vollkommen zufrieden. Ich mochte die Zusammenstellung der Bootskripte nicht, Programme waren nicht nach meinem Geschmack vorkonfiguriert. Viele Dinge dieser Art haben mich gestört. Wenn ich vollends mit meinem Linux zufrieden sein wollte, musste ich es -- nur mit Hilfe der Quellen -- selbst von Grund auf erstellen. Ich beschloss, weder vorkompilierte Pakete noch eine CD-ROM oder Bootdiskette für die Installation der Basiswerkzeuge zu benutzen. Ich würde mein gerade laufendes Linux-System verwenden, um mein eigenes Linux zu entwickeln.

Die Umsetzung dieser Idee schien zu diesem Zeitpunkt sehr schwierig und oft erschien sie sogar unmöglich. Nachdem ich jedoch eine Vielzahl von Problemen wie Abhängigkeiten und Kompilierfehler aus der Welt geschafft hatte, war mein eigenes Linux-System gebaut und voll funktionsfähig. Ich nannte es Linux From Scratch, kurz: LFS.

Ich hoffe, Sie haben viel Freude beim Erstellen Ihres eigenen LFS!

--
Gerard Beekmans
gerard@linuxfromscratch.org

Die Zielgruppe

Wer dieses Buch wahrscheinlich lesen möchte

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 Frage, die viele Leute stellen, 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 IhrSystem 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 Konfiguration 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 laufen zu lassen mit gerade mal 8 Mb belegtem Festplattenspeicher. 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.

Wir können noch eine weitere Analogie zum Vergleich heranziehen. Vergleichen wir 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.

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.

Wer dieses Buch wahrscheinlich nicht lesen möchte

Es gibt sicherlich einige, die - aus welchen Gründen auch immer - dieses Buch nicht lesen wollen. Wenn Sie Ihr Linux-System nicht von Grund auf selbst bauen möchten, ist dieses Buch vermutlich die falsche Lektüre. Unser Ziel ist es, Ihnen zu helfen, ein vollständiges, lauffähiges und grundsolides System zu erstellen. Wenn Sie nur interessiert, was genau beim Hochfahren Ihres Computers geschieht, dann empfehlen wir das „From Power Up To Bash“-HOWTO. 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.

Wenn Sie sich entscheiden, was Sie lesen möchten, halten Sie sich einfach Ihr Ziel vor Augen. Wenn Sie ein komplettes Linux installieren wollen und nebenbei ein bisschen dazulernen wollen, dann ist dieses Buch vermutlich die beste Wahl. Wenn Ihr Ziel aber eher die reine Übung ist und Sie dann später keine weitere Verwendung für das fertige Linux-System haben, dann ist das „From Power Up To Bash“-HOWTO wahrscheinlich die bessere Wahl.

Das „From Power Up To Bash Prompt“-HOWTO finden Sie unter http://axiom.anu.edu.au/~okeefe/p2b/ oder auf der Website des Linux Documentation Project unter http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html.

Voraussetzungen

In diesem Buch gehen wir davon aus, 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

  Dies ist ein umfangreiches Handbuch zum Erstellen und Installieren „allgemeiner“ UNIX-Softwarepakete unter Linux. Das HOWTO ist erhältlich unter http://www.tldp.org/HOWTO/Software-Building-HOWTO.html.

- The Linux Users' Guide

  Dieses Handbuch behandelt die Verwendung ausgewählter Linux-Software und ist zu finden unter http://espc22.murdoch.edu.au/~stewart/guide/guide.html.

- The Essential Pre-Reading Hint

  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. Sie ist verfügbar unter http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

Konventionen in diesem Buch

Zum besseren Verständnis gibt es einige 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 und für Eingabebeispiele.

http://www.linuxfromscratch.org/

Diese Textform wird für Links benutzt, sowohl innerhalb des Buches als auch zu externen Seiten wie HOWTOs, Downloads 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 zu hier zu lesen ist.

Danksagungen

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

Derzeitige Projektmitglieder

- Gerard Beekmans <gerard@linuxfromscratch.org> -- Linux From Scratch Initiator, LFS-Projektbetreuer.
- Matthew Burgess <matthew@linuxfromscratch.org> -- LFS Co-Maintainer, allgemeine Paketbetreuung, LFS-Buchautor.
- Craig Colton <meerkats@bellsouth.net> -- LFS, ALFS, BLFS und Ersteller des Logos für das Hint-Projekt.
- Nathan Coulson <nathan@linuxfromscratch.org> -- LFS-Bootskript-Betreuer.
- Jeroen Coumans <jeroen@linuxfromscratch.org> -- Website-Entwickler, Betreuung der FAQ.
- Bruce Dubbs <bdubbs@linuxfromscratch.org> -- Kopf des LFS-Qualitätssicherungsteams, BLFS-Buchautor.
- Manuel Canales Esparcia <manuel@linuxfromscratch.org> -- LFS-Buchautor (XML).
- Alex Groenewoud <alex@linuxfromscratch.org> -- LFS-Buchautor.
- Mark Hymers <markh@linuxfromscratch.org> -- Betreuung des CVS, Ersteller des BLFS-Buches, ehemaliger LFS-Buchautor.
- James Iwanek <iwanek@linuxfromscratch.org> -- Mitglied des Teams für Systemadministration.
- Nicholas Leippe <nicholas@linuxfromscratch.org> -- Wiki-Betreuer.
- Anderson Lizardo <lizardo@linuxfromscratch.org> -- Ersteller und Betreuer der Website-Skripte.
- Bill Maltby <bill@linuxfromscratch.org> -- LFS Projekt-Organisator.
- Alexander Patrakov <alexander@linuxfromscratch.org> -- LFS-Buchautor (Internationalisierung/Lokalisierung).
- Scot Mc Pherson <scot@linuxfromscratch.org> -- Betreuer des NNTP-Gateway.
- Ryan Oliver <ryan@linuxfromscratch.org> -- Kopf des Test-Teams, Betreuung der Toolchain, Mitbegründer des PLFS.
- James Robertson <jwrober@linuxfromscratch.org> -- Bugzilla-Betreuer, Wiki-Entwickler, LFS-Buchautor.
- Greg Schafer <greg@linuxfromscratch.org> -- Toolchain-Betreuer, ehemaliger LFS-Buchautor, Mitbegründer des PLFS.
- Tushar Teredesai <tushar@linuxfromscratch.org> -- BLFS-Buchautor, Betreuer der Hint und Patches Projekte.
- Jeremy Utley <jeremy@linuxfromscratch.org> -- LFS-Buchautor, Bugzilla-Betreuer, Betreuung der LFS-Bootskripte, Co-Administrator des LFS-Servers.
- Zack Winkles <winkie@linuxfromscratch.org> -- LFS-Buchautor, Co-Betreuer der LFS-Bootskripte.
- 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 Spiegel
- Mikhail Pastukhov <miha@xuy.biz> -- lfs.130th.net Spiegel.
- Frank Mancuso <crash4o4@gameover.com> -- lfs.crash404.com Spiegel.
- William Astle <lost@l-w.net> -- ca.linuxfromscratch.org Spiegel.
- Jeremy Polen <jpolen@rackspace.com> -- us2.linuxfromscratch.org Spiegel.
- Tim Jackson <tim@idge.net> -- linuxfromscratch.idge.net Spiegel.
- Jeremy Utley <jeremy@linux-phreak.net> -- lfs.linux-phreak.net Spiegel.

Südamerikanische Spiegel

- Manuel Canales Esparcia <manuel@linuxfromscratch.org> -- lfsmirror.lfs-es.org Spiegel.
- Andres Meggiotto <sysop@mesi.com.ar> -- lfs.mesi.com.ar Spiegel.
- Eduardo B. Fonseca <ebf@aedsolucoes.com.br> -- br.linuxfromscratch.org Spiegel.

Europäische Spiegel

- Barna Koczka <barna@siker.hu> -- hu.linuxfromscratch.org Spiegel.
- UK Mirror Service -- linuxfromscratch.mirror.ac.uk Spiegel.
- Martin Voss <Martin.Voss@ada.de> -- lfs.linux-matrix.net Spiegel.
- Unknown -- mirror.vtx.ch Spiegel
- Guido Passet <guido@primerelay.net> -- nl.linuxfromscratch.org Spiegel.
- Bastiaan Jacques <baafie@planet.nl> -- lfs.pagefault.net Spiegel
- Roel Neefs <lfs-mirror@linuxfromscratch.rave.org> -- linuxfromscratch.rave.org Spiegel.
- Justin Knierim <justin@jrknierim.de> -- www.lfs-matrix.de Spiegel
- Stephan Brendel <stevie@stevie20.de> -- lfs.netservice-neuss.de Spiegel.
- Unknown -- linuxfromscratch.je-zi.de Spiegel
- Unknown -- linuxfromscratch.tuxcenter.net Spiegel
- Hagen Herrschaft <hrx@hrxnet.de> -- de.linuxfromscratch.org Spiegel.
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> -- at.linuxfromscratch.org Spiegel.
- Fredrik Danerklint <fredan-lfs@fredan.org> -- se.linuxfromscratch.org Spiegel.
- Parisian sysadmins <archive@doc.cs.univ-paris8.fr> -- www2.fr.linuxfromscratch.org Spiegel.
- Alexander Velin <velin@zadnik.org> -- bg.linuxfromscratch.org Spiegel.
- Dirk Webster <dirk@securewebservices.co.uk> -- lfs.securewebservices.co.uk Spiegel
- Thomas Skyt <thomas@sofagang.dk> -- dk.linuxfromscratch.org Spiegel.
- Simon Nicoll <sime@dot-sime.com> -- uk.linuxfromscratch.org Spiegel.

Asiatische Spiegel

- Pui Yong <pyng@spam.averse.net> -- sg.linuxfromscratch.org Spiegel.
- Stuart Harris <stuart@althalus.me.uk> -- lfs.mirror.intermedia.com.sg Spiegel
- Unknown -- lfs.mirror.if.itb.ac.id Spiegel

Australische Spiegel

- Jason Andrade <jason@dstc.edu.au> -- au.linuxfromscratch.org Spiegel.

Spender

- Dean Benson <dean@vipersoft.co.uk> für etliche Geldspenden.
- DREAMWVR.COM für das ehemalige Sponsoring diverser Ressourcen zu LFS und dazugehörigen Unterprojekten.
- Hagen Herrschaft <hrx@hrxnet.de> für die Spende eines 2,2 GHz P4 Systems, welches nun unter dem Namen *lorien* läuft.
- O'Reilly für die gespendeten Bücher zu SQL und PHP.
- VA Software die, im Namen von Linux.com, eine VA Linux 420 (ehem. StartX SP2) Workstation gespendet haben.
- Mark Stone für die Spende von *shadowfax*, dem ersten linuxfromscratch.org Server, einem 750 MHz P3 mit 512 MB RAM und zwei 9 GB SCSI Festplatten. Als der Server umgezogen wurde, wurde er in *belgarath* umbenannt.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> für die Spende eines Yamaha CDRW 8824E CD-Brenners.
- Zahllose weitere Menschen aus den verschiedenen LFS-Mailinglisten, die dieses Buch mit Ihren Vorschlägen, Fehlerberichten und Kritiken besser machen.

Ausgeschiedene Team-Mitglieder und Beitragende

- Timothy Bauscher <timothy@linuxfromscratch.org> -- LFS-Buchautor, Betreuung des Hint-Projektes.
- Robert Briggs für die Spende der *linuxfromscratch.org* und *linuxfromscratch.com* Domain Namen.
- Ian Chilton <ian@ichilton.co.uk> für die Betreuung des Hint-Projektes.
- Marc Heerdink <gimli@linuxfromscratch.org> -- LFS-Buchautor.
- Seth W. Klein <sklein@linuxfromscratch.org> -- Erschaffer der LFS-FAQ.
- Garrett LeSage <garrett@linuxart.com> -- Schöpfer des ursprünglichen LFS-Banners.
- Simon Perreault <nomis80@videotron.ca> -- Betreuer des Hint-Projektes.
- Geert Poels <Geert.Poels@skynet.be> -- Schöpfer des ursprünglichen BLFS-Banner; basierend auf dem LFS-Banner von Garrett LeSage.
- Frank Skettino <bkenoah@oswd.org> für das ursprüngliche Design der alten Website -- schauen Sie unter http://www.oswd.org/.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> für das vorübergehende Hosten des linuxfromscratch.org Servers, dass Beantworten zahlloser Fragen im IRC und für seine unendliche Geduld.

Aufbau

Dieses Buch ist in die folgenden Abschnitte unterteilt:

Teil I - Einführung

Teil I erläutert einige wichtige Dinge zur Installation und schafft Grundlagen zu allgemeinen Dingen des Buches (Version, Änderungsprotokoll, Danksagungen, zugehörige Mailinglisten und so weiter).

Teil II - Vorbereitungen zur Installation

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

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 im Verlauf des Buches installiert wurden.

Teil I. Einführung

Kapitel 1. Einführung

Der Ablauf im Überblick

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.

In Chapter 2[p.9] erstellen Sie als erstes eine neue Linux-Partition und ein Dateisystem, auf dem Ihr neues LFS-System kompiliert und installiert wird. Dann laden Sie in Chapter 3[p.14] alle für LFS notwendigen Pakete und Patches herunter und speichern sie auf dem neuen Dateisystem. In Chapter 4[p.20] erstellen Sie sich dann eine gute Arbeitsumgebung für die weiteren Schritte.

Chapter 5[p.26] beschreibt dann die Installation einiger Pakete für die grundlegende Entwicklungsumgebung (im weiteren Verlauf des Buches *Toolchain* genannt), die benötigt wird, um dann das eigentliche System in Chapter 6[p.?] zu erstellen. Einige dieser Pakete werden benötigt, um rekursive Abhängigkeiten aufzulösen -- zum Beispiel benötigen Sie einen Compiler, um einen Compiler zu kompilieren.

Als erstes erstellen Sie im Chapter 5[p.26] eine erste Version der Basiswerkzeuge, bestehend aus Binutils und GCC. Die Programme aus diesen Paketen werden statisch verlinkt, damit sie unabhängig vom Host-System benutzt werden können. Im zweiten 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 Chapter 5[p.26] 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 Chapter 5[p.26], inklusive einiger Hinweise auf die Unterschiede zwischen statisch und dynamisch verlinkter Programme.

In Chapter 6[p.66] wird das eigentliche LFS-System erstellt. Wir benutzen das Programm chroot (change root, wechseln der Wurzel), um eine Shell in einer virtuellen Umgebung zu starten, in der das root-Verzeichnis 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 Chapter 7[p.161] die Boot-Skripte eingerichtet, der Kernel und der Bootloader werden in Chapter 8[p.171] konfiguriert, und Chapter 9[p.177] 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.

Bitte lesen Sie Chapter 4[p.20] sehr genau, es erklärt einige sehr wichtige Dinge, über die Sie sich im klaren sein sollten bevor Sie mit Chapter 5[p.26] und den folgenden beginnen.

Änderungsprotokoll

Dies ist das Linux From Scratch Buch in der Version 5.1.1 vom June 5th, 2004. Wenn dieses Buch älter als zwei Monate ist, gibt es vielleicht bereits eine neuere, bessere Version. Das können Sie überprüfen, indem Sie einen unserer Software-Spiegel aus der Liste von http://www.linuxfromscratch.org/ besuchen.

Unten finden Sie eine Liste der Änderungen die seit der vorherigen Buchversion vorgenommen wurden. Erst eine Zusammenfassung, dann ein detailliertes Protokoll.

- Aktualisierung auf:
  - autoconf-2.59
  - automake-1.8.4
  - coreutils-5.2.1
  - e2fsprogs-1.35
  - expect-5.41.0
  - file-4.09
  - gcc-3.3.3
  - gettext-0.14.1
  - glibc-2.3.3-lfs-5.1
  - grub-0.94
  - kbd-1.12
  - less-382
  - lfs-bootscripts-2.0.5
  - libtool-2.5.6
  - linux-2.4.26
  - man-pages-1.66
  - modutils-2.4.27
  - ncurses-5.4
  - perl-5.8.4
  - procps-3.2.1
  - psmisc-21.4
  - sed-4.0.9
  - shadow-4.0.4.1
  - tar-1.13.94
  - tcl-8.4.6
  - texinfo-4.7
  - util-linux-2.12a
- Hinzugefügt:
  - iana-etc-1.00
  - inetutils-1.4.2-no_server_man_pages-1.patch
  - make_devices-1.2
  - mktemp-1.5 + mktemp-1.5-add-tempfile.patch

- Entfernt:
  - gcc-3.3.1-suppress-libiberty.patch
  - lfs-utils-0.5
  - MAKEDEV-1.7
  - man-1.5m2-manpath.patch
  - man-1.5m2-pager.patch
  - ncurses-5.3-etip-2.patch
  - ncurses-5.3-vsscanf.patch
  - perl-5.8.0-libc-3.patch
  - procps-3.1.11-locale-fix.patch
  - shadow-4.0.3-newgrp-fix.patch
  - zlib-1.1.4-vsnprintf.patch
- June 2, 2004 [matt]: Prologue - acknowledgments, Added Thomas Reitelbach as the German translator
- May 30, 2004 [matt]: Chapter 6 - vim, corrected the optional command for invoking the testsuite
- May 23, 2004 [matt]: Chapter 6 - kbd, removed the hardcoded path to the kernel source directory
- May 19, 2004 [matt]: Chapter 6 - mktemp, added instruction to install tempfile wrapper
- May 18, 2004 [manuel]: Chapter 3 - Updated the list of mirrors for Glibc package. Fixed several textual bugs.
- May 17th, 2004 [winkie]: Chapter 5 - Pass „AUTOCONF=no“ to the Glibc build. This prevents autoconf from causing us problems.
- May 16th, 2004 [jeremy]: Chapter 9 - Added a brief paragraph to the rebooting system page to discuss packages which might be useful to add prior to rebooting to the new system
- May 15th, 2004 [matt]: Chapter 6 - Added a clearer warning that make_devices needs to be customised
- May 14th, 2004 [matt]: Chapter 3 - Added glibc's md5sum
- May 14th, 2004 [matt]: Chapters 5 & 6 - Upgraded to glibc-2.3.3-lfs-5.1
- May 11th, 2004 [jeremy]: Prologue - Updated the list of active staff in the project.
- May 9th, 2004 [winkie]: Chapter 6 - Removed unused and broken entries from `nsswitch.conf`.
- May 7th, 2004 [matt]: Merged Manuel's lfs-xsl-0.9 patches
- May 7th, 2004 [matt]: Fixed README error regarding invocation of `make`
- May 3rd, 2004: LFS 5.1-pre2 released
- May 2nd, 2004 [matt]: Quoted chroot commands in chapter 6 (bug #818).
- May 2nd, 2004 [matt]: Removed description of the now non-existent part IV from the structure section in the prologue.
- May 1st, 2004 [jeremy]: Added creation of the /media and /srv directories, as well as 2 directories under /media for floppy and cdrom, as per FHS - fixes bugzilla bug #785 and #819.
- April 14th, 2004 [jeremy]: Updated to lfs-bootscripts-2.0.3, no textual changes needed
- March 24th, 2004 [jeremy]: Chapter 7 - Updated to the new lfs-bootscripts-2.0.2, and all necessary changes to the bootscript configuration
- March 21st, 2004 [winkie]: Chapter 6 - Replaced Lfs-Utils with Iana-Etc and Mktemp.
- February 27th, 2004 [jeremy]: Upgraded to Procps-3.2.0.
- February 27th, 2004 [jeremy]: Upgraded to Lfs-utils-0.5 - fixes a possible symlink attack in iana-get.
- February 27th, 2004 [jeremy]: Chapter 6 - Altered the instructions for Findutils to be FHS-compliant.

- February 26th, 2004 [jeremy]: Removed the creation of the /usr/etc directory to conform with FHS - closes bug 775.
- February 26th, 2004 [jeremy]: Upgraded to Linux-2.4.25.
- February 23rd, 2004 [alex]: Chapters 6 + 9 - Cleaned up the Revision of chroot and Reboot sections.
- February 22nd, 2004 [alex]: Moved the stripping of the final system from chapter 9 to the end of chapter 6.
- February 22nd, 2004 [alex]: Chapter 6 - Coreutils and E2fsprogs: Clarified the prerequisites for running the tests.
- February 19th, 2004 [alex]: Chapter 5 - Stripping: Removed an unnecessary „{,share/}“ from the documentation's **rm** command.
- February 14th, 2004 [jeremy]: Chapter 6 - Upgraded to Less-382.
- February 14th, 2004 [jeremy]: Chapters 5 + 6 - Upgraded to Ncurses-5.4, and removed references to the etip patch.
- February 12th, 2004 [jeremy]: Chapter 6 - Removed explicit paths from the pwconv and grpconv commands, since /usr/sbin is part of the default path.
- February 9th, 2004 [alex]: Chapter 6 - Moved the Bootscripts installation section to chapter 7.
- February 8th, 2004 [matt]: Chapter 6 - Updated to Man-pages-1.66.
- February 7th, 2004 [alex]: Chapter 1 - Moved the Conventions and Acknowledgments sections to the Preface.
- February 7th, 2004 [alex]: Chapter 6 - Creating devices: replaced the MAKEDEV script with the make_devices script. Contributed by Matthias Benkmann.
- February 5th, 2004 [alex]: Chapter 6 - Simplified the final install of the kernel headers to just copying them from the temporary tools directory.
- February 4th, 2004 [alex]: Chapters 5 + 6 - Moved the Mounting of proc and devpts to before Chrooting, dropped Util-linux from the tools, and added a little arch script for Perl.

Erscheinen der Version 5.1-pre1 am 01. Februar 2004.

Ressourcen

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/.

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* Port 6667. Der Support-Chatraum heist #LFS-support.

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.

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*.

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.

Referenzen

Wenn Sie noch detailliertere Informationen zu Paketen brauchen, werden Sie auf dieser Seite hilfreiche Links finden: http://www.109bean.org.uk/LFS-references.html.

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.

Kontakt

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

Wie Sie um Hilfe bitten können

Wenn Sie beim Lesen des Buches auf ein Problem stoßen, sollten Sie als erstes in der FAQ unter http://www.linuxfromscratch.org/faq/ 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 („Ressourcen“[p.6]) 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.

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 (dies ist die Version 5.1.1),
- 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.

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

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.

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 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
```

Viele Leute kopieren leider nur den unteren Teil:

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

und das darauf folgende. 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.

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.

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.

Kapitel 2. Vorbereiten einer neuen Partition

Einführung

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

Erstellen einer neuen Partition

Um das neue Linux-System zu installieren, brauchen wir etwas Platz: eine leere Partition. Wenn Sie keine freie Partition und keinen unpartitionierten Platz auf Ihrer Festplatte haben, können Sie LFS auch auf der Partition installieren, auf der bereits Ihre gerade installierte Distribution läuft. Dieses Vorgehen empfehlen wir nicht, wenn Sie das erste Mal ein LFS installieren. Aber wenn Sie wenig Plattenplatz haben und sich etwas zutrauen, schauen Sie sich die Anleitung unter http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt an.

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.

Man hat fast nie genug Arbeitsspeicher, deshalb 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 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.

Erstellen eines Dateisystems auf der neuen Partition

Nun haben wir 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 wir ein ext2-Dateisystem erstellen. Anleitungen für andere Dateisysteme können Sie unter http://www.linuxfromscratch.org/blfs/view/stable/postlfs/filesystems.html finden.

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

```
mke2fs /dev/xxx
```

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

Wenn Sie eine (neue) 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:

```
mkswap /dev/yyy
```

Bitte ersetzen Sie `yyy` mit dem Namen Ihrer Swap-Partition.

Einhängen (mounten) der neuen Partition

Nachdem wir nun ein Dateisystem erzeugt haben, möchten wir auch darauf zugreifen können. Dazu müssen wir erst einen Mountpunkt wählen und es dann dort einhängen (mounten). In diesem Buch nehmen wir an, 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
```

Nun erstellen 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 Bezeichnungen ersetzen.

Sie sollten sicherstellen, 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 Dateisysteme eingehängt sind. Wenn Sie nosuid, nodev oder noatime sehen, müssen Sie Ihre Partition erneut einhängen.

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

Teil II. Vorbereitungen zur Installation

Kapitel 3. Das Material: Pakete und Patches

Einführung

Die untenstehende Liste enthält alle Pakete, die Sie für ein minimales Linux-System herunterladen müssen. Die Versionsnummern entsprechen Softwareversionen von denen wir *wissen*, dass Sie funktionieren, und das Buch basiert darauf. Wenn Sie wenig Erfahrung mit LFS haben, empfehlen wir dringend, 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.

Wir können nicht garantieren, dass die Download-Adressen immer verfügbar sind. Falls sich eine Download Adresse nach Erscheinen des Buches geändert haben sollte, googlen Sie bitte nach dem entsprechenden Paket. Sollten Sie auch hier erfolglos sein, schauen Sie bitte auf die Korrekturseiten unter http://www.linuxfromscratch.org/lfs/print/, oder noch besser, Sie probieren eine alternative Download-Methode aus. Das Vorgehen wird auf der Seite 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 bequemenen 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 was Sie benötigen immer auf der LFS-Partition abgelegt und in allen Arbeitsschritten des Buches verfügbar.

Wir empfehlen Ihnen daher, folgendes Kommando als *root* auszuführen, bevor Sie mit dem Herunterladen der Pakete beginnen:

```
mkdir $LFS/sources
```

Machen Sie diesen Ordner für normale Benutzer schreibbar (und sticky) -- denn Sie werden das Herunterladen der Pakete nicht als *root* durchführen. Wir schlagen folgendes Kommando vor:

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

Alle Pakete

Laden Sie die folgenden Pakete herunter:

Autoconf (2.59) - 903 KB:
http://freshmeat.net/projects/autoconf/

Automake (1.8.4) - 644 KB:
http://freshmeat.net/projects/automake/

Bash (2.05b) - 1,910 KB:
http://freshmeat.net/projects/gnubash/

Binutils (2.14) - 10,666 KB:
http://freshmeat.net/projects/binutils/

Bison (1.875) - 796 KB:
http://freshmeat.net/projects/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/

Ed (0.2) - 182 KB:
http://freshmeat.net/projects/ed/

Expect (5.41.0) - 510 KB:
http://freshmeat.net/projects/expect/

File (4.09) - 356 KB: -- *(Siehe Hinweis 1 weiter unten)*
http://freshmeat.net/projects/file/

Findutils (4.1.20) - 760 KB:
http://freshmeat.net/projects/findutils/

Flex (2.5.4a) - 372 KB:
ftp://ftp.gnu.org/gnu/non-gnu/flex/

Gawk (3.1.3) - 1,596 KB:
http://freshmeat.net/projects/gnuawk/

GCC (2.95.3) - 9,618 KB:
http://freshmeat.net/projects/gcc/

GCC-core (3.3.3) - 11,283KB:
http://freshmeat.net/projects/gcc/

GCC-g++ (3.3.3) - 2,026 KB:
http://freshmeat.net/projects/gcc/

GCC-testsuite (3.3.3) - 1,051 KB:

http://freshmeat.net/projects/gcc/

Gettext (0.14.1) - 6,397 KB:
http://freshmeat.net/projects/gettext/

Glibc (2.3.3-lfs-5.1) - 13,101 KB: -- *(Siehe Hinweis 2 weiter unten)*
http://freshmeat.net/projects/glibc/

Grep (2.5.1) - 545 KB:
http://freshmeat.net/projects/grep/

Groff (1.19) - 2,360 KB:
http://freshmeat.net/projects/groff/

Grub (0.94) - 902 KB:
ftp://alpha.gnu.org/pub/gnu/grub/

Gzip (1.3.5) - 324 KB:
ftp://alpha.gnu.org/gnu/gzip/

Iana-Etc (1.00) - 161 KB:
http://freshmeat.net/projects/iana-etc/

Inetutils (1.4.2) - 1,019 KB:
http://freshmeat.net/projects/inetutils/

Kbd (1.12) - 617 KB:
http://freshmeat.net/projects/kbd/

Less (382) - 259 KB:
http://freshmeat.net/projects/less/

LFS-Bootscripts (2.0.5) - 32 KB:
http://downloads.linuxfromscratch.org/

Libtool (1.5.6) - 2,602 KB:
http://freshmeat.net/projects/libtool/

Linux (2.4.26) - 30,051 KB:
http://freshmeat.net/projects/linux/

M4 (1.4) - 310 KB:
http://freshmeat.net/projects/gnum4/

Make (3.80) - 899 KB:
http://freshmeat.net/projects/gnumake/

Make_devices (1.2) - 20 KB:
http://downloads.linuxfromscratch.org/

Man (1.5m2) - 196 KB:
http://freshmeat.net/projects/man/

Man-pages (1.66) - 1,582 KB:
http://freshmeat.net/projects/man-pages/

Mktemp (1.5) - 69 KB:
http://freshmeat.net/projects/mktemp/

Modutils (2.4.27) - 229 KB:
http://freshmeat.net/projects/modutils/

Ncurses (5.4) - 2,019 KB:

http://freshmeat.net/projects/ncurses/

Net-tools (1.60) - 194 KB:
http://freshmeat.net/projects/net-tools/

Patch (2.5.4) - 182 KB:
http://freshmeat.net/projects/patch/

Perl (5.8.4) - 9,373 KB:
http://freshmeat.net/projects/perl/

Procinfo (18) - 24 KB:
http://freshmeat.net/projects/procinfo/

Procps (3.2.1) - 260 KB:
http://freshmeat.net/projects/procps/

Psmisc (21.4) - 375 KB:
http://freshmeat.net/projects/psmisc/

Sed (4.0.9) - 751 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 (1.13.94) - 1,025 KB:
ftp://alpha.gnu.org/gnu/tar/

Tcl (8.4.6) - 3,363 KB:
http://freshmeat.net/projects/tcltk/

Texinfo (4.7) - 1,385 KB:
http://freshmeat.net/projects/texinfo/

Util-linux (2.12a) - 1,814 KB:
http://freshmeat.net/projects/util-linux/

Vim (6.2) - 3,193 KB:
http://freshmeat.net/projects/vim/

Zlib (1.2.1) - 277 KB:
http://freshmeat.net/projects/zlib/

Gesamtgröße der Pakete: 134 MB

Anmerkung

Anmerkung 1) Wenn Sie das hier lesen ist (4.09) 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://gaosu.rave.org/pub/linux/lfs/.

Anmerkung

Anmerkung 2) Als dieses Buch geschrieben wurde, haben die Betreuer von Glibc entschieden, neue Glibc-Versionen nicht als Tarball zum Download bereitzustellen. Daher hat das LFS-Toolchain-Team

einen eigenen Tarball aus dem CVS erzeugt und, sofern nötig, Patches bereits eingespielt.

Der Tarball ist auf den LFS-Softwarespiegeln verfügbar:

ftp://gaosu.rave.org/pub/linux/lfs/packages/conglomeration/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://lfs.mirror.intermedia.com.sg/pub/lfs/lfs-packages/conglomeration/glibc-2.3.3-lfs-5.1.tar.bz2
http://packages.lfs-es.org/glibc/glibc-2.3.3-lfs-5.1.tar.bz2
http://mirror.averse.net/lfs-packages/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://mirror.averse.net/pub/lfs-packages/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://ftp.lfs-matrix.de/lfs-packages/conglomeration/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://ftp.sg.linuxfromscratch.org/pub/lfs-packages/glibc-2.3.3-lfs-5.1.tar.bz2
http://ftp.sg.linuxfromscratch.org/glibc-2.3.3-lfs-5.1.tar.bz2

Falls Sie die Integrität des Tarballs überprüfen möchten: Dies ist die MD5 Prüfsumme: `cd11fabdf5162ad68329e7b28b308278`. Sie kann mit dem Programm **md5sum** verglichen werden.

Erforderliche Patches

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

Bash Patch - 7 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/bash-2.05b-2.patch

Bison Attribute Patch - 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/bison-1.875-attribute.patch

Coreutils Hostname Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/coreutils-5.2.1-hostname-1.patch

Coreutils Uname Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/coreutils-5.2.1-uname-1.patch

Ed Mkstemp Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/ed-0.2-mkstemp.patch

Expect Spawn Patch - 6 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/expect-5.41.0-spawn-1.patch

GCC No-Fixincludes Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/gcc-3.3.3-no_fixincludes-1.patch

GCC Specs Patch - 11 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/gcc-3.3.3-specs-1.patch

GCC-2 Patch - 16 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/gcc-2.95.3-2.patch

GCC-2 No-Fixincludes Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/gcc-2.95.3-no-fixinc.patch

GCC-2 Return-Type Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/gcc-2.95.3-returntype-fix.patch

Inetutils No-Server-Man-Pages Patch - 4 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/inetutils-1.4.2-no_server_man_pages-1.patch

Kbd More-Programs Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/kbd-1.12-more-programs-1.patch

Man 80-Columns Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/man-1.5m2-80cols.patch

Mktemp Tempfile Patch - 3 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/mktemp-1.5-add-tempfile.patch

Net-tools Mii-Tool-Gcc33 Patch - 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/net-tools-1.60-miitool-gcc33-1.patch

Perl Libc Patch - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.1.1/perl-5.8.4-libc-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

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 gesetzt sein, den Sie für Ihre LFS-Partition gewählt haben. Überprüfen Sie nochmals mit dem folgenden Kommando, dass die 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).

Erstellen des Ordners $LFS/tools

Alle in Chapter 5[p.26] kompilierten Programme werden unter`$LFS/tools` installiert. Dadurch trennen wir sie von den Programmen, die in Chapter 6[p.66] installiert werden. Die hier kompilierten Programme sind nur übergangsweise Hilfsmittel und werden kein Teil des endgültigen LFS-Systems sein. Wenn wir 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 Chapter 5[p.26] leicht passieren), was wirklich schlecht wäre.

Falls Sie später Ihre ausführbaren Dateien des Systems durchsuchen möchten, um zum Beispiel herauszufinden welche Dateien sie benutzen oder wogegen sie verlinkt sind, können Sie die Suche vereinfachen indem Sie einen eindeutigen Namen vergeben. Statt dem einfachen „tools“ können Sie etwas wie „tools-fuer-lfs“ benutzen. Dann müssen Sie allerdings im gesamten Buch sehr sorgfältig alle Referenzen auf „tools“ entsprechend anpassen -- inklusive aller Patches, wie z. B. dem GCC Specs Patch.

Erstellen Sie den Ordner mit diesem Kommando:

```
mkdir $LFS/tools
```

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

```
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 wir immer noch einige Programme vom Host-System benutzen) *als auch* im nächsten Kapitel (wenn wir in die LFS-Partition „chroot'ed“ haben) funktionieren werden (weil wir immer den gleichen gültigen Pfad benutzen).

Hinzufügen des Benutzers lfs

Als *root* Benutzer eingeloggt können kleinste Fehler Ihr System beschädigen oder gar zerstören. Deshalb empfehlen wir, 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 wir dazu den Benutzer *lfs* erstellen und diesen während des ganzen Installationsvorgangs benutzen. Bitte führen Sie als *root* dieses Kommando aus, um den neuen Benutzer zu erzeugen:

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

Die Bedeutung der Parameter:

- **`-s /bin/bash`**: Dies macht die **`bash`** zur voreingestellten Shell für den Benutzer *lfs*.
- **`-m`**: Dies erzeugt den Persönlichen Ordner für *lfs*.
- **`-k /dev/null`**: Dieser Parameter verhindert das mögliche Kopieren von Dateien aus einem skeleton-Ordner (Voreinstellung ist `/etc/skel`), indem der Quellpfad dieser Dateien auf das spezielle Null-Gerät eingestellt wird.

Wenn Sie sich als *lfs* einloggen möchten, müssen Sie für *lfs* ein Passwort vergeben:

```
passwd lfs
```

und *lfs* vollen Zugriff auf den Ordner `$LFS/tools` geben. 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 „**-**“ sorgt dafür, dass **su** eine neue Shell startet.

Vorbereiten der Arbeitsumgebung

Um Ihre Arbeitsumgebung für die weiteren Schritte vorzubereiten, erstellen wir 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. *non-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 Verzeichnisse zu vermeiden. Jedoch müssen wir 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 auf 022 stellt sicher, dass neu erzeugte Dateien nur durch ihren Besitzer beschreibbar sind, aber für alle anderen les- und ausführbar.

Die LFS-Variable 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.

Wir stellen `/tools/bin` an den Anfang der PATH Variable, so dass wir beim Durcharbeiten dieses Kapitels die erstellten Werkzeuge und Programme auch automatisch benutzt, sobald sie verfügbar sind.

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

```
source ~/.bash_profile
```

Über SBUs

Die meisten Leute möchten gerne vorher wissen, wie lange es ungefähr dauert, die einzelnen Pakete zu kompilieren und installieren. "Linux From Scratch" wird aber auf so unterschiedlichen Systemen gebaut, dass es unmöglich ist, echte Zeiten anzugeben, die auch nur annähernd akkurat wären: Das grösste Paket (Glibc) braucht auf schnellen Maschinen nicht einmal 20 Minuten, aber auf langsamen Maschinen drei Tage oder mehr -- das ist kein Scherz. 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 Chapter 5[p.26]. Die Zeit die Sie benötigen, um dieses Paket zu kompilieren, 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 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-3.3.3 ist dafür bekannt, ca. 30% langsamer zu sein.

Bitte beachten Sie auch, dass die SBU-Angaben auf Mehrprozessormaschinen nicht gut anwendbar sind. Aber wenn Sie das Glück haben eine solche Maschine zu besitzen, wird der Unterschied höchstwahrscheinlich so gering sein, dass es Ihnen egal sein kann.

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

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.

Anmerkung

Die Erfahrung hat uns gezeigt, dass man nicht viele Vorteile durch das Durchlaufen der Testsuites in Chapter 5[p.26] 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 Chapter 5[p.26] erzeugten Werkzeuge sind nur temporär und werden ohnehin später wieder gelöscht. Wir empfehlen Ihnen, die Testsuites in Chapter 5[p.26]*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. Wir werden das später in Chapter 5[p.26] ausführlicher behandeln.

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

Einführung

In diesem Kapitel werden wir ein minimales Linux-System kompilieren und installieren. Das System wird gerade genug Werkzeuge beinhalten, um mit dem Erstellen des endgültigen LFS-Systems im nachfolgenden Kapitel 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 wir eine brandneue Host-unabhängige Toolchain (Compiler, Assembler, Linker und Bibliotheken). Dann benutzen wir 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 *lfs* Benutzer 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 über den Bildschirm laufen sehen. Das ist normal und kann einfach ignoriert werden. Das sind wirklich nur Warnungen -- meistens über missbilligte (aber dennoch korrekte) 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 auch mögliche Fehlkonfigurationen, wenn ein Paket später nochmalig 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.

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 erstmal die ersten Pakete installiert haben. Scheuen Sie sich nicht, zwischendurch noch einmal in diesem Abschnitt nachzulesen.

Das Ziel von Chapter 5[p.26] ist es, eine gut funktionierende temporäre Arbeitsumgebung zu erschaffen, in die wir uns später abkapseln und von wo aus wir in Chapter 6[p.66] ohne Schwierigkeiten ein sauberes endgültiges LFS-System erstellen können. Wir werden uns so weit wie möglich vom Host-System abschotten und so eine in sich geschlossene Toolchain erzeugen. Bitte beachten Sie, dass der gesamte Prozess so 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. Ein einfacher Weg sein Ziel-Tripplet herauszufinden besteht darin, das Skript `config.guess` auszuführen, das mit den Quellen vieler Pakete mitgeliefert wird. Entpacken Sie die Binutils-Quellen, führen Sie das Skript aus: **`./config.guess`** und 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 of binary> | 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 Hinweise zum Kompilierprozess in Chapter 5[p.26]:

- Der Kompilierprozess 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 wir, dass Programme nur gegen die gewünschten Bibliotheken gelinkt werden.
- Vorsichtiges Anpassen von **gcc**'s *specs* Datei um dem Compiler mitzuteilen, 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 konfigurierter 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, und 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 Verzeichnisse 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** man 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, den Binärwerkzeugen 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 `glibc-build` Ordner nach den Details durchsuchen. Sie werden ein paar interessante Dinge finden, wie zum Beispiel *CC="gcc -B/tools/bin/"* zum Kontrollieren der verwendeten Binärwerkzeuge, 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 nicht von Toolchain-Vorgaben abhängig.

Nach der Installation von Glibc nehmen wir noch ein paar Anpassungen vor;damit stellen wir sicher, dass Suchen und Verlinken nur innerhalb unseres Prefix `/tools` stattfindet. Wir installieren einen angepassten **ld**, welcher einen fest angegebenen Suchpfad auf `/tools/lib` hat. Dann bearbeiten wir 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 hier an kompilierte Programm bis zum Ende des Kapitels unseren neuen Dynamischen Linker in `/tools/lib` benutzt.

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

Während des zweiten Durchlaufs der Binutils können wir 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 Chapter 5[p.26] 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 wir nach dem Eintreten in die chroot-Umgebung in Chapter 6[p.66] installieren. Wenn die Glibc erstmal nach`/usr` installiert ist, werden wir schnell ein paar Voreinstellungen in der Toolchain ändern und dann schreiten wir mit dem Erstellen des endgültigen LFS-Systems fort.

Bemerkungen 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 all 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 der Standard 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 linken wir die ersten beiden Pakete in diesem Kapitel dann statisch? 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 wir durch diesen Schritt noch einen Schritt unabhängiger vom Host-System werden. Das bedeutet, diese Programme können unabhängig vom Host-System eingesetzt werden. Natürlich könnten wir auch dann noch ein gut funktionierendes LFS-System erstellen, wenn diese Pakete dynamisch gelinkt werden.

Binutils-2.14 - Durchlauf 1

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

```
Approximate build time:  1.0 SBU
Required disk space:     170 MB
```

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

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 daraufhin erst diverse Funktionalitäten einschalten.

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 nun Binutils zum Kompilieren vor:

```
../binutils-2.14/configure --prefix=/tools --disable-nls
```

Die Bedeutung der configure-Parameter:

- **`--prefix=/tools`**: Dies teilt dem configure-Skript mit, die Installation der Binutils-Programme in den Ordner `/tools` vorzubereiten.
- **`--disable-nls`**: Dies deaktiviert die Internationalisierung (oft auch als i18n abgekürzt). Wir brauchen keine Internationalisierung für unsere statischen Programme und *nls* verursacht häufig Probleme beim statischen Verlinken von Programmen.

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. Wir benutzen diese Option zur Umgehung dieses Problems.
- **`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 dennoch 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-Bibliothekssuchpfad 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 „Inhalt von Binutils“[p.85].

GCC-3.3.3 - Durchlauf 1

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

```
Approximate build time:  4.4 SBU
Required disk space:     411.7 MB
```

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

Installieren von GCC

Entpacken Sie nur den GCC-core Tarball; wir 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-3.3.3/configure --prefix=/tools \
    --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 möchten wir mögliche Einflüsse aus dem Host-System vermeiden, 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 im zweiten Durchlauf selbst damit zu kompilieren. Darauf folgt ein weiterer 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 wir 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 „Inhalt von GCC“[p.87].

Linux-2.4.26 Header

```
Approximate build time:  0.1 SBU
Required disk space:     192.5 MB
```

Installation der Kernel-Header

Da einige Pakete die Kernel Header referenzieren, entpacken wir 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 Kernelsauszuführen. 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 symlinks
```

Installieren Sie die plattformspezifischen Header-Dateien:

```
mkdir /tools/include/asm
cp include/asm/* /tools/include/asm
cp -R include/asm-generic /tools/include
```

Installieren Sie die Multiplattform-Headerdateien:

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

Glibc-2.3.3-lfs-5.1

Glibc ist 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.). Die C-Bibliothek wird von allen dynamisch gelinkten Programmen verwendet.

```
Approximate build time:  11.8 SBU
Required disk space:     734.2 MB
```

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

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-2.3.3-lfs-5.1/configure --prefix=/tools \
    --disable-profile --enable-add-ons=linuxthreads \
    --with-binutils=/tools/bin --with-headers=/tools/include \
    --without-gd --without-cvs
```

Die Bedeutung der configure-Parameter:

- **`--disable-profile`**: Dies sorgt dafür, dass die Bibliotheken ohne Profiling-Informationen erzeugt werden. Lassen Sie diese Option weg, wenn Sie mit den erzeugten Bibliotheken Profiling betreiben möchten.
- **`--enable-add-ons=linuxthreads`**: Dies aktiviert Zusätze, die zu Glibc installiert wurden, in unserem Fall Linuxthreads.
- **`--with-binutils=/tools/bin`** und **`--with-headers=/tools/include`**: Genaugenommen werden diese Optionen nicht benötigt. Aber sie stellen sicher, dass in Bezug auf die Kernel-Header und Binutils-Programme beim Kompilieren der Glibc nichts schiefgehen kann.
- **`--without-gd`**: Diese Option stellt sicher, dass wir nicht das **`memusagestat`** Programm erzeugen, das seltsamerweise immer gegen die Host-Bibliotheken (libgd, libpng, libz und so weiter) verlinkt wird.
- **`--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, um eine Warnung bezüglich des fehlenden Programmes **`autoconf`** zu verhindern.

Während dieser Phase sehen Sie möglicherweise die folgende 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.

Kompilieren Sie das Paket:

```
make AUTOCONF=no
```

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 check
```

Die Glibc Testsuite ist sehr stark von einigen Funktionen Ihres Host-Systems abhängig, insbesondere vom Kernel. Darüberhinaus können in diesem Kapitel einige Tests von der Umgebung Ihres Host-Systems negativ beeinflusst werden. Diese werden natürlich kein Problem mehr sein, wenn wir später die Testsuite von Glibc in der chroot-Umgebung in Chapter 6[p.66] ausführen. Grundsätzlich erwarten wir, 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. Auch andere Dateisystemeigenschaften können hier Einfluss haben.
- 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 Timeouts fehlschlagen.

Machen Sie sich keine allzugrossen Gedanken, wenn einige Glibc-Tests in diesem Kapitel fehlschlagen. Die Glibc aus Chapter 6[p.66] ist diejenige, die wir endgültig verwenden werden. Erst dort ist es wirklich wichtig, dass die Tests erfolgreich durchlaufen. Aber denken Sie daran, selbst in Chapter 6[p.66] können immer noch Fehler auftreten -- beim *math*-Test zum Beispiel. 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 mitloggen, damit Sie später die Logdatei nach den aufgetretenen Fehlern durchsuchen können.

Auch wenn es nur eine harmlose Meldung ist, die Installationsphase von Glibc wird sich über das Fehlen von `/tools/etc/ld.so.conf` beschweren. Verhindern Sie diese störende Meldung:

```
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 komplexeKonventionen, wie zum Beispiel die dort gesprochene Sprache. Die „Internationalisierung“ von GNU-Programmen funktioniert mit Hilfe der sogenannten Locales. Wir installieren 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. Wir 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 „Inhalt von Glibc“[p.79].

Die Glibc "integrieren"

Jetzt wo 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 wir den Linker und die Specs-Datei des Compilers anpassen. Manche Leute meinen das untenstehende wäre *„Schwarze Magie“*, aber in Wirklichkeit ist es ganz einfach.

Installieren Sie zuerst den angepassten Linker (die Anpassung haben Sie am Schluss des ersten Binutils-Durchlaufs durchgeführt) in dem Sie folgendes Kommando im Ordner `binutils-build` 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, seien Sie unbesorgt. Es ist noch nicht alles verloren. Ignorieren Sie das obige Kommando einfach. Die Folge davon 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, da der angepasste Linker installiert ist, müssen Sie die Binutils-Ordner *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:

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

Wir empfehlen, das obige Kommando nicht abzutippen sondern mittels Kopieren und Einfügen auszuführen. Sie können 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“.

Wichtig

Wenn Sie auf einer Plattform arbeiten, auf der der Name des dynamischen Linkers anders lautet als `ld-linux.so.2`, *müssen* Sie natürlich statt `ld-linux.so.2` den korrekten Namen des Linkers für Ihre Plattform einsetzen. Falls nötig, schauen Sie nochmal im Abschnitt „Technische Anmerkungen zur Toolchain“[p.27] 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 Problem zu umgehen:

```
rm -f /tools/lib/gcc-lib/*/*/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 wir 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

ist:

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

Achten Sie besonders darauf, dass `/tools/lib` der Prefix zu ihrem dynamischen Linker ist.

Wenn Sie keine oder eine andere als die obige Ausgabe Ausgabe erhalten haben, ist etwas ernsthaft schiefgelaufen. Sie müssen alle Ihre Schritte noch einmal überprüfen und den Fehler finden und korrigieren. Machen Sie nicht weiter, 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 „GCC-3.3.3 - Durchlauf 1“[p.32] und reparieren Sie die Verknüpfung. 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 „Vorbereiten der Arbeitsumgebung“[p.23] ist etwas schiefgelaufen. Es könnte auch etwas beim Anpassen der Specs-Datei fehlgeschlagen sein. In diesem Fall wiederholen Sie die Anpassung und benutzen Sie Kopieren und Einfügen, um das Kommando auszuführen, tippen Sie es nicht ab.

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

```
rm dummy.c a.out
```

Tcl-8.4.6

Das Tcl Paket enthält die sog. Tool Command Language.

```
Approximate build time:  0.9 SBU
Required disk space:     22.7 MB
```

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

Installieren von Tcl

Dieses und die nächsten beiden Pakete werden nur installiert, damit wir 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 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 Tcl Testsuite 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 Durchlauchs der Testsuite auf Coordinated Universal Time (UTC), auch als Greenwhich Mean Time (GMT) bekannt. Dadurch werden zeitbezogene Tests korrekt ausgewertet. Mehr Informationen zu der TZ Umgebungsvariable finden Sie später in Chapter 7[p.161].

Installieren Sie das Paket:

```
make install
```

Warnung

Sie sollten den `tcl8.4.6` Quellordner 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
```

Inhalt von Tcl

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

Installierte Bibliothek: libtcl8.4.so

Kurze Beschreibung

tclsh8.4 ist die Tcl Kommando-Shell.

libtcl8.4.so ist die Tcl-Bibliothek.

Expect-5.41.0

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

```
Approximate build time:  0.1 SBU
Required disk space:     3.9 MB
```

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

Installieren von Expect

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

```
patch -Np1 -i ../expect-5.41.0-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 wir 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 unbedingt 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.)

Und installieren Sie:

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

Die Bedeutung des make-Parameters:

- **`SCRIPTS=""`**: Dies verhindert die Installation der mitgelieferten Expect-Skripte, wir brauchen sie hier nicht.

Sie können nun die Quellordner von Tcl und Expect entfernen.

Inhalt von Expect

Installiertes Programm: expect

Installierte Bibliothek: libexpect5.41.0.a

Kurze Beschreibung

expect „spricht“ mit anderen interaktiven Programmen und benutzt dazu ein anpassbares Skript.

DejaGnu-1.4.4

Das Paket DejaGnu enthält ein Grundgerüst zum Testen anderer Programme.

```
Approximate build time:  0.1 SBU
Required disk space:     6.1 MB
```

Dejagnu ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installieren von DejaGnu

Bereiten Sie DejaGnu zum Kompilieren vor:

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

Kompilieren und installieren Sie das Paket:

```
make install
```

Inhalt von DejaGnu

Installiertes Programm: runtest

Kurze Beschreibung

runtest ist das Wrapper-Skript, das die korrekte expect-Shell findet und DejaGnu ausführt.

GCC-3.3.3 - Durchlauf 2

```
Approximate build time:  11.0 SBU
Required disk space:     332.7 MB
```

Neuinstallation von GCC

Die Hilfsmittel zum Testen von GCC und Binutils sind nun installiert (Tcl, Expect und DejaGnu). Wir können GCC und Binutils nun erneut installieren, sie gegen die neue Glibc verlinken und testen. Es gibt eine Sache, die noch beachtet werden muss: 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"
```

Wenn Sie diese Meldung erhalten:

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

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 wir sowohl den C- als auch den C++-Compiler. Entpacken Sie die GCC core- und g++- Tarballs (und -testsuite, falls Sie die Test durchlaufen lassen möchten). Die Archive entpacken sich in einen einzigen Unterordner namens `gcc-3.3.3/`.

Zuerst korrigieren Sie ein Problem und führen eine wichtige Anpassung durch:

```
patch -Np1 -i ../gcc-3.3.3-no_fixincludes-1.patch
patch -Np1 -i ../gcc-3.3.3-specs-1.patch
```

Der erste Patch schaltet das GCC „fixincludes“-Skript ab. Wir haben das vorher bereits kurz erwähnt; hier wollen wir eine nähere Erklärung dazu geben. Unter normalen Umständen durchsucht das fixincludes-Skript von GCC Ihr System nach Header-Dateien die repariert werden müssen. 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 Chapter 6[p.66], nachdem wir 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 wir 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 GCC Include-Suchpfad. 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. Vergessen Sie nicht, sie zu installieren.

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

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

Bevor Sie mit dem Kompilieren von GCC beginnen denken Sie daran, alle Umgebungsvariablen zurückzusetzen, die die Standard Optimierungen überschreiben würden.

Bereiten Sie nun GCC zum Kompilieren vor:

```
../gcc-3.3.3/configure --prefix=/tools \
    --with-local-prefix=/tools \
    --enable-clocale=gnu --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-languages=c,c++
```

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 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 C++ shared libraries und C++-Programme die interoperabel mit anderen Linux-Distributionen sind.
- **`--enable-languages=c,c++`**: Diese Option wird benötigt, damit sowohl der C- als auch der C++-Compiler erzeugt werden.

Kompilieren Sie das Paket:

```
make
```

Diesmal müssen Sie nicht das *bootstrap*-Target verwenden, weil wir 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 für GCC dennoch laufen lassen möchten, führen Sie dieses Kommando aus:

```
make -k check
```

Der Schalter *-k* lässt die Testsuite bis zum Ende durchlaufen, auch 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-3.3.3/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-3.3.3 auf i686-pc-linux-gnu aussehen sollte finden Sie unter http://gcc.gnu.org/ml/gcc-testresults/2004-01/msg00826.html.

Beachten Sie, dass das Ergebnis folgendes enthält:

```
* 1 XPASS (unexpected pass) for g++
* 1 FAIL (unexpected failure) for gcc
* 24 XPASS's for libstdc++
```

Der erfolgreiche Durchlauf für g++ ist unerwartet, weil wir *--enable-__cxa_atexit* benutzt haben. Offensichtlich unterstützen nicht alle von GCC unterstützten Plattformen "__cxa_atexit" in ihren C-Bibliotheken, daher wird das erfolgreiche Durchlaufen dieses Tests als unerwartet betrachtet.

Die 24 unerwartet erfolgreichen Durchläufe für libstdc++ sind begründet durch die Option *--enable-clocale=gnu*, das die korrekte Wahl Systeme ist die auf Glibc Version 2.2.5 oder höher basieren. Die zugrunde liegende Locale-Unterstützung in der GNU C-Bibliothek ist besser als das ansonsten gewählte Modell *generic* (welches anwendbar wäre, wenn Sie zum Beispiel Newlibc, Sun-libc oder eine sonstige libc verwenden würden). Die libstdc++ Testsuite erwartet anscheinend das Modell *generic*, daher ist das erfolgreiche Absolvieren dieser Tests unerwartet.

Ein paar unerwartete Fehler lassen sich oftmals gar 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.

Schlussendlich installieren Sie das Paket:

```
make install
```

Anmerkung

An diesem Punkt empfehlen wir dringend, die Gesamtprüfung, die wir früher in diesem Kapitel gemacht haben, noch einmal durchzuführen. Schlagen Sie im „Die Glibc "integrieren"“[p.38] 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 „Inhalt von GCC“[p.87].

Binutils-2.14 - Durchlauf 2

```
Approximate build time:  1.5 SBU
Required disk space:     35.6 MB
```

Neuinstallation von Binutils

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

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

Bereiten Sie nun Binutils zum Kompilieren vor:

```
../binutils-2.14/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 Bibliothekssuchpfad des Linkers als **`/tools/lib`** vorzugeben. Wir möchten im Standard Bibliothekssuchpfad keine Ordner unseres Host-Systems haben, daher geben wir den gewünschten Pfad vor.

Bevor Sie mit dem Kompilieren von Binutils beginnen denken Sie daran, alle Umgebungsvariablen zu entfernen, die die Standard-Optimierungen überschreiben würden.

Kompilieren Sie das Paket:

```
make
```

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 für Binutils dennoch laufen lassen möchten, führen Sie dieses Kommando aus:

```
make check
```

Es sollten keine unerwarteten Fehler auftreten, erwartete Fehler sind in Ordnung. Leider gibt es hier, anders als im GCC-Paket, keine einfache Möglichkeit die Testergebnisse zusammenfassend anzuzeigen. Nichtsdestotrotz, wenn ein Fehler auftritt, sollte er leicht zu erkennen sein. Die Ausgabe zeigt dann etwas wie:

```
make[1]: *** [check-binutils] Error 2
```

Und 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. Wir brauchen sie im jetzigen Zustand noch im nächsten Kapitel.

Details zu diesem Paket finden Sie in „Inhalt von Binutils“[p.85].

Gawk-3.1.3

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

```
Approximate build time:  0.2 SBU
Required disk space:     16.9 MB
```

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

Installieren von Gawk

Bereiten Sie Gawk zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Und installieren Sie:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Gawk“[p.98].

Coreutils-5.2.1

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

```
Approximate build time:  0.9 SBU
Required disk space:     69 MB
```

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

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 dieKompatibilität 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.)

Und installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Coreutils“[p.89].

Bzip2-1.0.2

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. Bei Textdateien erreichen Sie eine wesentlich bessere Kompressionsrate als das traditionelle Kommando **gzip**.

```
Approximate build time:  0.1 SBU
Required disk space:     2.5 MB
```

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

Installieren von Bzip2

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

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

Details zu diesem Paket finden Sie in „Inhalt von Bzip2“[p.126].

Gzip-1.3.5

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

```
Approximate build time:  0.1 SBU
Required disk space:     2.6 MB
```

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

Installation von Gzip

Bereiten Sie Gzip zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Und installieren Sie:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Gzip“[p.136].

Diffutils-2.8.1

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

```
Approximate build time:  0.1 SBU
Required disk space:     7.5 MB
```

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

Installieren von Diffutils

Bereiten Sie Diffutils zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Und installieren Sie:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Diffutils“[p.128].

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).

```
Approximate build time:  0.2 SBU
Required disk space:     7.5 MB
```

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

Installieren von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Und installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Findutils“[p.97].

Make-3.80

Das Paket Make enthält Programme zum Kompilieren umfangreicher Pakete.

```
Approximate build time:  0.2 SBU
Required disk space:     8.8 MB
```

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

Installieren von Make

Bereiten Sie Make zum Kompilieren vor:

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

Kompilieren Sie das Programm:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Nun installieren Sie Make und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Make“[p.140].

Grep-2.5.1

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

```
Approximate build time:  0.1 SBU
Required disk space:     5.8 MB
```

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

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 die Programme:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Dann installieren Sie sie und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Grep“[p.134].

Sed-4.0.9

Das Paket Sed enthält einen Stream-Editor.

```
Approximate build time:  0.2 SBU
Required disk space:     5.9 MB
```

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

Installieren von Sed

Bereiten Sie Sed zum Kompilieren vor:

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

Kompilieren Sie das Programm:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Nun installieren Sie Make und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Sed“[p.108].

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 Meldungen in der Sprache des Anwenders ausgegeben werden.

```
Approximate build time:  0.5 SBU
Required disk space:     67.6 MB
```

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

Installieren von Gettext

Bereiten Sie Gettext zum Kompilieren vor:

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

Kompilieren Sie die Programme:

```
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) und ist nicht als kritisch einzustufen. Deshalb empfehlen wir, diesen Schritt zu überspringen. Ausserdem 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 verfügbar).)

Und installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Gettext“[p.110].

Ncurses-5.4

Das Paket Ncurses enthält Bibliotheken für den terminalunabhängigen Zugriff auf Textbildschirme.

```
Approximate build time:  0.7 SBU
Required disk space:     27.8 MB
```

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

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 die Programme und Bibliotheken:

```
make
```

Dann installieren Sie sie und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Ncurses“[p.99].

Patch-2.5.4

Das Paket Patch enthält ein Programm zum Modifizieren von Dateien.

```
Approximate build time:  0.1 SBU
Required disk space:     1.9 MB
```

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

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=/tools
```

Kompilieren Sie das Programm:

```
make
```

Nun installieren Sie Make und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Patch“[p.142].

Tar-1.13.94

Das Paket Tar enthält ein Archivprogramm.

```
Approximate build time:  0.2 SBU
Required disk space:     10.3 MB
```

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

Installieren von Tar

Bereiten Sie Tar zum Kompilieren vor:

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

Kompilieren Sie die Programme:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Dann installieren Sie sie und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Tar“[p.153].

Texinfo-4.7

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

```
Approximate build time:  0.2 SBU
Required disk space:     16.3 MB
```

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

Installieren von Texinfo

Bereiten Sie Texinfo zum Kompilieren vor:

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

Kompilieren Sie die Programme:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **`make check`**.)

Dann installieren Sie sie und die dazugehörige Dokumentation:

```
make install
```

Details zu diesem Paket finden Sie in „Inhalt von Texinfo“[p.118].

Bash-2.05b

Das Paket Bash enthält die Bourne-Again-SHell.

```
Approximate build time:  1.2 SBU
Required disk space:     27 MB
```

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

Installieren von Bash

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

```
patch -Np1 -i ../bash-2.05b-2.patch
```

Bereiten Sie Bash nun zum Kompilieren vor:

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

Kompilieren Sie das Programm:

```
make
```

(Wenn Sie die Testsuite durchlaufen lassen wollen, führen Sie dieses Kommando aus: **`make tests`**.)

Nun installieren Sie Make und die dazugehörige Dokumentation:

```
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 „Inhalt von Bash“[p.123].

Util-linux-2.12a

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

```
Approximate build time:  0.2 SBU
Required disk space:     16 MB
```

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

Installieren von Util-linux

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

```
cp configure configure.backup
sed "s@/usr/include@/tools/include@g" configure.backup > configure
```

Bereiten Sie Util-linux zum Kompilieren vor:

```
./configure
```

Kompilieren Sie einige unterstützende Routinen:

```
make -C lib
```

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

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

Nun kopieren wir diese Programme in unseren temporären Ordner tools:

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

Details zu diesem Paket finden Sie in „Inhalt von Util-linux“[p.154].

Perl-5.8.4

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

```
Approximate build time:  0.8 SBU
Required disk space:     74 MB
```

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

Installieren von Perl

Zuerst müssen Sie ein paar festeingestellte Pfade zur C-Bibliothek anpassen:

```
patch -Np1 -i ../perl-5.8.4-libc-1.patch
```

Perl besteht darauf, zum Feststellen der Plattform das Programm **arch** zu benutzen. Erzeugen Sie ein kleines Skript, um dieses Kommando nachzuahmen:

```
echo "uname -m" > /tools/bin/arch
chmod 755 /tools/bin/arch
```

Bereiten Sie Perl zum Kompilieren vor (passen Sie auf, dass Sie das '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
```

Dann kopieren Sie die Werkzeuge und ihre Bibliotheken an die richtige Stelle:

```
cp perl pod/pod2man /tools/bin
mkdir -p /tools/lib/perl5/5.8.4
cp -R lib/* /tools/lib/perl5/5.8.4
```

Details zu diesem Paket finden Sie in „Inhalt von Perl“[p.116].

Stripping

Die Schritte in diesem Abschnitt sind optional. Wenn Ihre LFS-Partition sehr klein ist werden Sie froh sein, dass Sie einige unnötige Dinge loswerden 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 allen restlichen Paketen keine Probleme haben.

Teil III. Installation des LFS-Systems

Kapitel 6. Installieren der grundlegenden System-Software

Einführung

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

Die Installation der ganzen Software ist recht einfach. Vielleicht sind Sie der Meinung das es einfacher wäre, wenn wir hier eine generelle Installationsanleitung geben würden und nur bei davon abweichenden Paketen eine vollständige Erklärung geben. Auch wenn wir dieser Überlegung im Grunde zustimmen haben wir uns entschlossen, für jedes Paket eine vollständige Anleitung zu geben, einfach um die Fehlerwahrscheinlichkeit so gering wie möglich zu halten.

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, die offizielle Download-Adresse (falls Sie Pakete updaten möchten) 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/cvs/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.

Einhängen der Dateisysteme proc- und devpts

Damit bestimmte Programme richtig funktionieren, müssen die Dateisysteme *proc* und *devpts* in der chroot-Umgebung verfügbar sein. Das proc-Dateisystem ist das Pseudo-Prozessinfo-Dateisystem. Der Kernel stellt mit diesem Dateisystem Informationen über den Status des Systems zur Verfügung. Und das devpts-Dateisystem ist die heutzutage übliche Methode, Pseudo-Terminals (PTYs) zu implementieren. Seit Kernel 2.4 kann ein Dateisystem so oft und an so vielen Stellen eingehängt sein, wie Sie möchten. Daher ist es auch kein Problem, dass diese Dateisysteme auch auf Ihrem Host-System bereits eingehängt sind -- besonders, weil es sich bei diesen beiden um virtuelle Dateisysteme handelt.

Erst müssen Sie *root* werden, denn nur *root* kann Dateisysteme an ungewöhnlichen Stellen einhängen. Prüfen Sie anschliessend, ob die Umgebungsvariable LFS korrekt gesetzt ist. Benutzen Sie dazu das Kommando **`echo $LFS`**, und stellen Sie sicher, dass Sie den Pfad zum Mountpunkt Ihrer LFS-Partition enthält. Das sollte `/mnt/lfs` sein, wenn Sie unserem Beispiel gefolgt sind.

Erzeugen Sie die Mountpunkte für die Dateisysteme:

```
mkdir -p $LFS/{proc,dev/pts}
```

Das *proc*-Dateisystem wird mit dem folgenden Kommando eingehängt:

```
mount proc $LFS/proc -t proc
```

Hängen Sie das *devpts*-Dateisystem mit diesem Kommando ein:

```
mount devpts $LFS/dev/pts -t devpts
```

Falls dieser Befehl mit einer Meldung wie dieser fehlschlägt:

```
filesystem devpts not supported by kernel
```

dann ist der wahrscheinlichste Grund dafür, dass der Kernel des Host-Systems ohne Unterstützung für das Dateisystem devpts kompiliert wurde. Sie können mit diesem Kommando überprüfen, welche Dateisysteme Ihr Kernel unterstützt: **cat /proc/filesystems**. Einige PTYs werden benötigt, um die Testsuites für Binutils und GCC ausführen zu können. Wenn devpts nicht aufgelistet wird, machen Sie sich keine Sorgen, es gibt noch einen anderen Weg, die PTYs in der chroot-Umgebung ans laufen zu bekommen. Wir behandeln das Thema später im Abschnitt Make_devices[p.73].

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. Ansonsten werden Sie höchstwahrscheinlich Probleme bekommen.

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 wir 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.

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 (und im Buch gehen wir davon aus, dass Sie dies tun) 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.

Erstellen der Ordner

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

```
mkdir -p /{bin,boot,dev/{pts,shm},etc/opt,home,lib,mnt,proc}
mkdir -p /{root,sbin,srv,tmp,usr/local,var,opt}
mkdir -p /media/{floppy,cdrom}
mkdir /usr/{bin,include,lib,sbin,share,src}
ln -s share/{man,doc,info} /usr
mkdir /usr/share/{doc,info,locale,man}
mkdir /usr/share/{misc,terminfo,zoneinfo}
mkdir /usr/share/man/man{1,2,3,4,5,6,7,8}
mkdir /usr/local/{bin,etc,include,lib,sbin,share,src}
ln -s share/{man,doc,info} /usr/local
mkdir /usr/local/share/{doc,info,locale,man}
mkdir /usr/local/share/{misc,terminfo,zoneinfo}
mkdir /usr/local/share/man/man{1,2,3,4,5,6,7,8}
mkdir /var/{lock,log,mail,run,spool}
mkdir -p /var/{tmp,opt,cache,lib/misc,local}
mkdir /opt/{bin,doc,include,info}
mkdir -p /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. Wir nehmen zwei Änderungen vor: eine für den Persönlichen Ordner von *root* und eine weitere für die Ordner für temporäre Dateien.

```
chmod 0750 /root
chmod 1777 /tmp /var/tmp
```

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 in der Bit Maske 1777.

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.

Erstellen notwendiger symbolischer Links

Einige Programme haben fest eingestellte Pfade zu Programmen, die hier aber noch nicht existieren. Deshalb erstellen wir 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
```

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:
EOF
```

Die erzeugten Gruppen sind nicht Teil irgendeines Standards -- es sind Gruppen, die das Skript **make_devices** im nächsten Abschnitt benutzt. Neben der Gruppe „root“ 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 wir eine neue Shell. Die Auflösung von Benutzer- und Gruppennamen funktioniert sofort nach dem Erstellen von `/etc/passwd` und `/etc/group`, weil wir in Chapter 5[p.26] 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 wir aber frisch installierte Programme sofort nach der Installation an ihrem neuen Ort benutzen möchten, schalten wir 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 wir die Logdateien und vergeben die richtigen Rechte:

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chmod 644 /var/run/utmp /var/log/{btmp,lastlog,wtmp}
```

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.

Erstellen der Gerätedateien mit Make_devices-1.2

Das Make_devices Paket enthält ein Skript zum Erzeugen von Gerätedateien.

```
Approximate build time:  1 SBU
Required disk space:     160 KB
```

Make_devices ist abhängig von: Bash, Bzip2, Coreutils.

Erstellen von Gerätedateien

Beachten Sie, dass beim Entpacken von `make_devices-1.2.bz2` kein neuer Ordner erstellt wird, in den Sie wechseln könnten, da das Paket nur ein Shell-Skript enthält.

Installieren Sie das Skript `make_devices`:

```
bzcat make_devices-1.2.bz2 > /dev/make_devices
chmod 754 /dev/make_devices
```

Gerätedateien sind spezielle Dateien: Sie können Daten erzeugen oder empfangen. Üblicherweise gehören sie zu einem physikalischen Teil Hardware. Gerätedateien können mit dem folgenden Kommando erzeugt werden: **mknod -m mode name type major minor**. In diesem Kommando entspricht *mode* den oktal angegebenen Rechten (Lesen/Schreiben/Ausführen), und *name* ist der Name der zu erzeugenden Gerätedatei. Es mag überraschend erscheinen, aber der Name der Gerätedatei ist frei wählbar, abgesehen davon, dass viele Programme sich darauf verlassen, dass Gerätedateien wie `/dev/null` ihren üblichen Namen besitzen. Die drei verbleibenden Parameter teilen dem Kernel mit, welches Gerät die Datei denn nun tatsächlich referenziert. *type* ist ein Buchstabe, entweder b oder c, und gibt an, ob das Gerät in Blöcken (wie zum Beispiel Festplatten) oder zeichenweise angesprochen wird (wie z. B. die Konsole). *major* und *minor* sind Nummern, die zusammen einen eindeutig identifizierbaren Kode für das Gerät ergeben. Eine Liste der zur Zeit zugewiesenen Nummern für Linux finden Sie in der Datei `devices.txt` im Unterordner `Documentation` in den Kernelquellen.

Beachten Sie, dass eine Major/Minor-Kombinationen üblicherweise sowohl einem Block- als auch einem Zeichenorientierten Gerät zugeordnet ist. Es handelt sich jedoch um vollkommen unterschiedliche Geräte, die nicht einfach vertauscht werden können. Ein Gerät wird durch alle drei Werte type/major/minor identifiziert und nicht nur durch major/minor. Wenn Sie also eine Gerätedatei erstellen, achten Sie darauf, den korrekten *Typ* anzugeben.

Weil das Nachschlagen von Typ, Major- und Minor-Nummern mittels **mknod** eine typische Fehlerquelle ist, wurde das Skript `make_devices` erstellt. Es enthält eine komplette Serie von **mknod**-Kommandos; eines pro Gerät, inklusive empfohlenem Namen, Rechten und Gruppenzuordnungen. Es wurde so erstellt, dass nur die üblichen Geräte aktiviert sind. Die restlichen Zeilen sind auskommentiert. Sie sollten `make_devices` mit einem Editor öffnen und an Ihre Bedürfnisse anpassen. Dies braucht seine Zeit, aber es ist sehr einfach. Wenn Sie zufrieden sind, führen Sie das Skript aus, um die Gerätedateien anzulegen:

Warnung

Ein Fehler beim Bearbeiten von **make_devices** (zum Beispiel bei der Anzahl der Partitionen) kann zu Fehlern beim Booten führen.

```
cd /dev
./make_devices
```

Falls Sie in „Einhängen der Dateisysteme proc- und devpts“[p.67] keine Schwierigkeiten mit dem Einhängen des Dateisystems devpts hatten, können Sie diesen Abschnitt überspringen und im nächsten Abschnitt weitermachen. Wenn Sie devpts nicht einhängen konnten, müssen Sie einige statische ptyXX und ttyXX Gerätedateien erzeugen. Dafür öffnen Sie `make_devices` in Ihrem Editor, suchen den Abschnitt „Pseudo-TTY masters“ und aktivieren einige ptyXX-Geräte -- eine Handvoll reicht, um die Testsuites zufriedenzustellen, aber wenn Sie einen Kernel ohne devpts-Unterstützung wünschen, brauchen Sie wahrscheinlich weitaus mehr (jedes xterm, ssh-Verbindung, telnet-Sitzung und so weiter benutzt ein Pseudo-Terminal). In dem direkt darauf folgenden Abschnitt „Pseudo-TTY slaves“ aktivieren Sie bitte noch die zugehörigen ttyXX-Geräte. Wenn Sie fertig sind führen Sie **./make_devices** in `/dev` erneut aus, dadurch werden die neuen Geräte erzeugt.

Inhalt von Make_devices

Installiertes Skript: make_devices

Kurze Beschreibung

make_devices ist ein Skript zum Erzeugen der grundlegenden statischen Gerätedateien im Ordner `/dev`.

Linux-2.4.26 Header

```
Approximate build time:  0.1 SBU
Required disk space:     186 MB
```

Installation der Kernel-Header

Wir werden jetzt noch keinen neuen Kernel kompilieren -- das erledigen wir, wenn wir die Installation aller Pakete abgeschlossen haben. Die im nächsten Abschnitt installierten Bibliotheken benötigen die Kernel-Header, da sie direkt mit dem Kernel arbeiten. Anstatt die Kernelquellen erneut zu entpacken, die Versionsdatei und symbolische Verknüpfungen zu erstellen u.s.w, kopieren wir einfach in einem Rutsch die Header aus dem temporären Tools-Ordner:

```
cp -a /tools/include/{asm,asm-generic,linux} /usr/include
```

Einige Kernel Header-Dateien benutzen die Header-Datei `autoconf.h`. Da wir den Kernel jetzt aber noch nicht konfigurieren und Compilerfehler vermeiden möchten, müssen wir die Datei selber erstellen. Erstellen Sie eine leere Datei `autoconf.h`:

```
touch /usr/include/linux/autoconf.h
```

Warum wir die Kernel-Header kopieren und nicht symbolisch linken

Früher war es gängige Praxis, den Ordner `/usr/include/{linux,asm}` nach `/usr/src/linux/include/{linux,asm}` symbolisch zu verlinken. Das war aber *schlechte* Praxis, wie der folgende Ausschnitt aus einem Posting von Linus Torvalds auf der Linux Kernel-Mailingliste zeigt:

```
I would suggest that people who compile new kernels should:

 - not have a single symbolic link in sight (except the one that the
   kernel build itself sets up, namely the „linux/include/asm“
   symlink that is only used for the internal kernel compile itself)

And yes, this is what I do. My /usr/src/linux still has the old 2.2.13
header files, even though I haven't run a 2.2.13 kernel in a _loong_
time. But those headers were what Glibc was compiled against, so those
headers are what matches the library object files.

And this is actually what has been the suggested environment for at
least the last five years. I don't know why the symlink business keeps
on living on, like a bad zombie. Pretty much every distribution still
has that broken symlink, and people still remember that the linux
sources should go into „/usr/src/linux“ even though that hasn't
been true in a _loong_ time.
```

Der wichtige Teil ist der, in dem Linus sagt, dass die *Header-Dateien diejenigen sein sollen, mit denen die Glibc kompiliert wurde*. Das sind die Header-Dateien, die zum späteren Kompilieren von Paketen verwendet werden sollten, weil nur diese exakt auf die Objektkodedateien der Bibliotheken passen. Durch das Kopieren der Header stellen wir sicher, dass sie verfügbar bleiben, falls Sie später den Kernel updaten.

Beachten Sie, dass es vollkommen in Ordnung ist, die Kernelquellen in `/usr/src/linux` liegen zu haben, solange Sie keine symbolischen Links `/usr/include/{linux,asm}` haben.

Man-pages-1.66

Das Paket Man-pages enthält über 1200 Hilfetexte.

```
Approximate build time:  0.1 SBU
Required disk space:     15 MB
```

Man-pages ist abhängig von: Bash, Coreutils, Make.

Installation der Man-pages

Installieren Sie die Man-pages durch Ausführen von:

```
make install
```

Inhalt von Man-pages

Installierte Dateien: verschiedene Hilfeseiten

Kurze Beschreibung

Man-pages enthält beispielsweise die Hilfeseiten zu allen C- und C++-Funktionen, wichtigen Geräte- und Konfigurationsdateien.

Glibc-2.3.3-lfs-5.1

Glibc ist 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.). Die C-Bibliothek wird von allen dynamisch gelinkten Programmen verwendet.

```
Approximate build time:  12.3 SBU
Required disk space:     784 MB
```

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

Installieren von Glibc

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. Wir können die Specs-Datei und den Linker nicht vor der Installation von Glibc modifizieren, weil die Glibc Autoconf-Tests dann falsche Resultate ergeben würden.

Bevor Sie mit dem Kompilieren von Glibc beginnen, denken Sie daran, alle Umgebungsvariablen zurückzusetzen, die die Standard-Optimierungen überschreiben würden.

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

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

Bereiten Sie nun Glibc zum Kompilieren vor:

```
../glibc-2.3.3-lfs-5.1/configure --prefix=/usr \
    --disable-profile --enable-add-ons=linuxthreads \
    --libexecdir=/usr/lib --with-headers=/usr/include \
    --without-cvs
```

Die Bedeutung der neuen configure-Optionen:

- **`--libexecdir=/usr/lib`**: Das wird das Programm `pt_chown` in `/usr/lib` anstelle von `/usr/libexec` installieren. Die Verwendung von *libexec* wird als nicht-FHS-konform betrachtet, weil FHS diesen Ordner noch nicht einmal erwähnt.
- **`--with-headers=/usr/include`**: Das stellt sicher, dass die Kernel-Header in `/usr/include` zum Kompilieren benutzt werden. Wenn Sie diese Option nicht angeben, werden die Header aus `/tools/include` benutzt, was nicht ideal wäre (auch wenn Sie eigentlich identisch sein sollten). Diese Option hat auch den Vorteil, dass Sie sofort merken, wenn Sie vergessen haben sollten, die Kernel-Header in `/usr/include` zu 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 Anmerkungen zur Testsuite aus „Glibc-2.3.3-lfs-5.1“[p.35] gelten natürlich auch hier. Schlagen Sie dort nach, falls Sie irgendwelche Zweifel haben.

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
```

Und 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
```

Schlussendlich erzeugen wir die Linxthreads-Manpages:

```
make -C ../glibc-2.3.3-lfs-5.1/linuxthreads/man
```

Und installieren diese:

```
make -C ../glibc-2.3.3-lfs-5.1/linuxthreads/man install
```

Konfigurieren von Glibc

Wir müssen die Datei `/etc/nsswitch.conf` erstellen, denn obwohl Glibc bei einer fehlenden oder kaputten Datei Standardwerte vorgibt, funktionieren diese Standardwerte nicht gut in Netzwerken. Außerdem müssen wir 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/Canada/Eastern /etc/localtime
```

Die Bedeutung der Option:

- **`--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.

Anstelle von *Canada/Eastern* müssen Sie natürlich den Namen der Zeitzone einsetzen, den Ihnen **tzselect** ausgeben hat.

Konfigurieren 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 Verzeichnisse in `/etc/ld.so.conf` eintragen, damit der dynamische Lader diese finden kann. Zwei Ordner, die dafür bekannt sind weitere Bibliotheken zu enthalten, sind `/usr/local/lib` und `/opt/lib`, also fügen wir 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
```

Inhalt von Glibc

Installierte Programme: catchsegv, gencat, getconf, getent, glibcbug, 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], libc_nonshared.a, 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 Beschreibung

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.

glibcbug erzeugt einen Fehlerbericht und verschickt ihn per E-Mail an die Bug-E-Mailadresse.

iconv führt Zeichensatzkonvertierungen durch.

iconvconfig erzeugt schnellladende iconv-Modul Konfigurationsdateien.

ldconfig konfiguriert die Laufzeitbindungen des dynamischen Linkers.

ldd gibt aus, welche gemeinsamen Bibliotheken von einem Programm oder einer Bibliothek benötigt werden.

lddlibc4 unterstützt ldd bei Objektdateien.

locale ist ein Perl-Programm, das im Compiler die Verwendung von POSIX-Locales für eingebaute Operationen ein- bzw. ausschaltet.

localedef erzeugt Locale-Spezifikationen.

mtrace...

nscd ist der "name service cache daemon"; dieser stellt einen Zwischenspeicher für die meisten namensbasierten Anfragen zur Verfügung.

nscd_nischeck prüft, ob der sichere Modus für NIS+-Anfragen 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 wird zum Erzeugen von symbolischen Verknüpfungen benutzt. Das Programm ist statisch verlinkt, daher kann es zum Erzeugen symbolischer Verknüpfungen auf dynamische Bibliotheken verwendet werden, selbst wenn das System zum dynamischen Linken aus irgendwelchen Gründen nicht funktioniert.

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 aufzulösen.

libSegFault behandelt Signale zu Speicherzugriffsfehlern.

libanl ist eine Bibliothek zum asynchronen Nachschlagen von Namen.

libbsd-compat ermöglicht einigen BSD-Programmen unter Linux zu laufen.

libc ist die C-Bibliothek -- eine Sammlung von häufig genutzten Funktionen.

libcrypt ist die Kryptographie-Bibliothek.

libdl ist eine Schnittstellenbibliothek zum dynamischen Linker.

libg ist eine Laufzeitbibliothek für g++.

libieee ist die IEEE-Fließkommabibliothek.

libm ist eine Mathematik-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* sind 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 ist 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.

Erneutes Anpassen der Toolchain

Nun, da die neue C Bibliothek installiert ist, muss die Toolchain erneut angepasst werden. Wir modifizieren sie so, dass alle weiteren kompilierten Programme gegen die neue C-Bibliothek gelinkt werden. Im Grunde ist das genau das gleiche, was wir im vorigen Kapitel beim Anpassen der Glibc schonmal gemacht haben, auch wenn es aussieht, als wäre es genau umgekehrt: Im vorigen Kapitel haben wir die Toolchain von `/{,usr/}lib` auf dem Host in das neue Verzeichnis `/tools/lib` umgelenkt. Nun lenken wir die Toolchain von diesem Verzeichns `/tools/lib` um nach `/{,usr/}lib` in unserem LFS-System.

Als erstes wird der Linker angepasst. Aus diesem Grunde haben wir die Quell- und Kompilierordner aus dem zweiten Durchlauf von Binutils bestehen lassen. Installieren Sie den angepassten Linker aus dem `binutils-build`-Ordner:

```
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` 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 passen Sie die Specs-Datei von GCC an, so dass sie auf den neuen dynamischen Linker verweist. Wie schon zuvor benutzen wir dazu sed:

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

Auch hier empfehlen wir, den Befehl zu kopieren und einzufügen. Und auch hier ist es wieder sinnvoll, die Specs-Datei darauf zu überprüfen, ob die Änderungen tatsächlich erfolgreich 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 „Technische Anmerkungen zur Toolchain“[p.27] nach.

Achtung

Es ist an diesem Punkt zwingend notwendig, die grundlegenden Funktionen (Kompilieren und Linken) der angepassten Toolchain zu überprüfen. Aus diesem Grund führen wir 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 `/lib` nun 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
```

Binutils-2.14

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

```
Approximate build time:  1.4 SBU
Required disk space:     167 MB
```

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

Installieren von Binutils

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

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

Wenn Sie die Nachricht:

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

erhalten, sind die PTYs in Ihrer chroot-Umgebung nicht korrekt konfiguriert. In dem Fall macht es keinen Sinn, die Tests für Binutils und GCC laufen zu lassen, solange Sie das Problem nicht behoben haben. Schlagen Sie bitte im Abschnitt „Einhängen der Dateisysteme proc- und devpts“[p.67] und Make_devices[p.73] nach und führen Sie die empfohlenen Schritte durch, um das Problem zu beseitigen.

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
```

Bereiten Sie nun Binutils zum Kompilieren vor:

```
../binutils-2.14/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
```

Die Anmerkungen zur Testsuite aus dem Abschnitt „Binutils-2.14 - Durchlauf 2“[p.46] sind hier immer noch gültig.

Schlagen Sie nach, falls Sie irgendwelche Bedenken oder Zweifel haben.

Installieren Sie das Paket:

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

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

```
cp ../binutils-2.14/include/libiberty.h /usr/include
```

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 Beschreibung

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 erzeugt, manipuliert und extrahiert aus Archiven. Ein Archiv ist eine einzelne Datei, die eine strukturierte Sammlung weiterer Dateien enthält.

as ist 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 ist ein Linker. Er verbindet mehrere Objektdateien und Archivdateien zu einer einzigen Datei, replaziert ihre Daten und verbindet ihre Symbolreferenzen.

nm listet alle Symbole auf, die in einer Objektdatei vorkommen.

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 ist die Bibliothek für Binärdateibezeichner.

libopcodes ist eine Bibliothek zur Behandlung von Obcodes. Sie wird zum Erzeugen von Werkzeugen wie z. B. objdump benutzt. Obcodes sind die **Text**-Versionen der Prozessorinstruktionen.

GCC-3.3.3

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

```
Approximate build time:  11.7 SBU
Required disk space:     294 MB
```

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

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/stable/general/gcc.html.

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

```
patch -Np1 -i ../gcc-3.3.3-no_fixincludes-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 nun GCC zum Kompilieren vor:

```
../gcc-3.3.3/configure --prefix=/usr \
    --enable-shared --enable-threads=posix \
    --enable-__cxa_atexit --enable-clocale=gnu \
    --enable-languages=c,c++
```

Kompilieren Sie das Paket:

```
make
```

Wichtig

Die GCC-Testsuite in diesem Abschnitt wird als *absolut kritisch* betrachtet. Wir raten Ihnen, sie unter keinen Umständen zu überspringen.

Testen Sie das Ergebnis, aber halten Sie bei Fehlern nicht an (Sie erinnern sich an die paar bekannten):

```
make -k check
```

Die Anmerkungen zur Testsuite aus dem „GCC-3.3.3 - Durchlauf 2“[p.43] gelten auch hier noch. Schlagen Sie dort nach, falls Sie irgendwelche Zweifel haben.

Installieren Sie das Paket:

```
make install
```

Einige Pakete erwarten, dass der C-Präprozessor im Ordner `/lib` installiert ist. Um diesen Paketen Rechnung zu tragen, erzeugen Sie 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 wir einen weiteren symbolischen Link:

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

Anmerkung

An dieser Stelle ist es wichtig, den „Gesundheitscheck“, den wir schon früher durchgeführt haben, erneut laufen zu lassen. Schlagen Sie im „Erneutes Anpassen der Toolchain“[p.82] nach und wiederholen Sie den Test. Wenn das Ergebnis negativ ist, haben Sie möglichweise versehentlich den GCC-Specs-Patch aus Chapter 5[p.26] angewendet.

Inhalt von GCC

Installierte Programme: c++, cc (Link auf gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug, und gcov

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

Kurze Beschreibung

cpp ist der C-Präprozessor. Er wird von dem Compiler benutzt, um #include und #define und ähnliche Anweisungen im Quellkode durch ihren endgültigen Kode zu erweitern.

g++ ist der C++-Compiler.

gcc ist der C-Compiler. Er wird verwendet, um den Quellkode eines Programmes in Assemblerkode umzuwandeln.

gccbug ist ein Shellskript, mit dem man gute Fehlerberichte erzeugen kann.

gcov ist 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++ ist die Standard C++-Bibliothek. Sie enthält viele häufig genutzte Funktionen.

libsupc++ stellt Unterstützungsroutinen für die Programmiersprache C++ zur Verfügung.

Coreutils-5.2.1

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

```
Approximate build time:  0.9 SBU
Required disk space:     69 MB
```

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

Installieren von Coreutils

Die Funktion von **uname** ist 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-1.patch
```

Wir möchten nicht, dass Coreutils seine Version von **hostname** installiert, weil sie schlechter ist als die von Net-tools bereitgestellte. Verhindern Sie die Installation mit dem folgenden Patch:

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

Bereiten Sie Coreutils zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Diese Testsuite trifft einige Annahmen in Hinsicht auf die Existenz von Benutzern und Gruppen, die in in unserem LFS-System noch nicht vorhanden sind. Wir müssen noch einige Dinge einrichten, bevor wir die Tests laufen lassen können. Falls Sie diese Testsuite nicht ausführen möchten, fahren Sie mit „Installieren Sie das Paket“ fort.

Damit die Testsuite vollständig durchlaufen kann, wird das Programm **su** benötigt. Wir haben uns aber nicht die Mühe gemacht es in Chapter 5[p.26] zu installieren, weil es root-Rechte benötigt. Wir holen dies nun nach:

```
make install-root
```

Erstellen Sie die „Tabelle der eingebundenen Dateisysteme“ mit:

```
touch /etc/mtab
```

Und 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
```

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

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

Die verbleibenden Tests werden als Benutzer *dummy* ausgeführt:

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

Danach entfernen Sie die dummy Gruppen und Benutzer:

```
sed -i.bak '/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,install,ln,ls} /bin
mv /usr/bin/{mkdir,mknod,mv,pwd,rm,rmdir,sync} /bin
mv /usr/bin/{sleep,stty,su,test,touch,true,uname} /bin
mv /usr/bin/chroot /usr/sbin
```

Wir benutzen das Programm `kill` aus dem Procps-Paket (welches unter `/bin/kill` später in diesem Kapitel installiert wird). Entfernen Sie die installierte Version von Coreutils:

```
rm /usr/bin/kill
```

Schliesslich erstellen Sie noch zwei symbolische Links, um FHS-kompatibel zu sein:

```
ln -s test /bin/[
ln -s ../../bin/install /usr/bin
```

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, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami und yes

Kurze Beschreibung

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 ein Kommando mit dem angegebenen Pfad als / Ordner aus. Das Kommando kann eine interaktive Shell sein. Auf den meisten Systemen darf das nur `root`.

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 zeigt die aktuelle Zeit im angegebenen Format an oder setzt die Systemzeit.

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 ist identisch mit ls.

dircolors gibt Kommandos zum Setzen der LS_COLOR Umgebungsvariable aus, um damit das Farbschema von ls zu ändern.

dirname entfernt den nicht-ordnerspezifischen Teil eines Dateinamens.

du gibt den verwendeten Festplattenspeicher aus, der vom aktuellen Ordner, den Unterordnern und Dateien oder einer einzelnen Datei verbraucht wird.

echo gibt die 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 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 zusammen, die identische join-Felder haben.

link erzeugt einen harten Link von der angegebenen Datei zu einer Datei.

ln erzeugt einen harten oder weichen Link zwischen Dateien.

logname gibt den Login-Namen des aktuellen Benutzers aus.

ls zeigt den Inhalt der angegebenen Ordner an. In der Voreinstellung werden Dateien und Ordner alphabetisch sortiert.

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 numeriert 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 ist 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 aktuelle Arbeitsumgebung 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 den Wert 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 unüblichen 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.

stty setzt Terminal-Einstellungen oder zeigt sie an.

su startet eine Shell mit anderer Benutzer- und/oder Gruppen-ID.

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.

uptime gibt aus, wie lange ein System bereits läuft, wieviele Benutzer eingeloggt sind und wie hoch die Systemlast ist.

users gibt die Namen der eingeloggten Benutzer aus.

vdir ist das gleiche wie ls -l.

wc gibt die Anzahl Zeilen, Wörter und Bytes einer Datei aus. Und eine Summe, falls mehrere Dateien angegeben wurden.

who 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.

Zlib-1.2.1

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

```
Approximate build time:  0.1 SBU
Required disk space:     1.5 MB
```

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

Installation von Zlib

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
```

Erzeugen Sie nun die nicht-gemeinsame (statische) Bibliothek:

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

Um erneut 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, während `/usr` möglicherweise noch nicht verfübar ist, trotzdem Zugriff zu diesen Bibliotheken.

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

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

Der symbolische Link `/usr/lib/libz.so` zeigt nun auf eine Datei, die nicht mehr existiert, weil wir sie gerade verschoben haben. Erstellen Sie den symbolischen Link neu, so dass er auf den neuen Standort der Bibliothek zeigt:

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

Inhalt von Zlib

Installierte Bibliotheken: libz[a,so]

Kurze Beschreibung

libz* enthält Funktionen zum Komprimieren und Dekomprimieren, die von einigen Programmen genutzt werden.

Mktemp-1.5

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

```
Approximate build time:  0.1 SBU
Required disk space:     317 KB
```

Die Installationsabhängigkeiten zu Mktemp wurden leider noch nicht überprüft.

Installation von Mktemp

Viele Skripte verwenden 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.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-Systembibliothek.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

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

Inhalt von Mktemp

Installierte Programme: mktemp, tempfile

Kurze Beschreibung

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 Gründern der Rückwärtskompatibilität installiert.

Iana-Etc-1.00

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

```
Approximate build time:  0.1 SBU
Required disk space:     641 KB
```

Die Installationsabhängigkeiten zu Iana-Etc wurden leider noch nicht überprüft.

Installation von Iana-Etc

Auffinden der Daten:

```
make
```

Installieren Sie es:

```
make install
```

Inhalt von Iana-Etc:

Installierte Dateien: protocols, services

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).

```
Approximate build time:  0.2 SBU
Required disk space:     7.5 MB
```

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

Installieren von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

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

Die obige localstatedir-Anweisung ändert den Standort der Locate-Datenbank wie vom FHS-Standard verlangt nach /var/lib/misc.

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

Inhalt von Findutils

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

Kurze Beschreibung

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.

Gawk-3.1.3

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

```
Approximate build time:  0.2 SBU
Required disk space:     17 MB
```

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

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
```

Inhalt von Gawk

Installierte Programme: awk (Link auf gawk), gawk, gawk-3.1.3, grcat, igawk, pgawk, pgawk-3.1.3, und pwcat

Kurze Beschreibung

gawk ist ein Programm zur Manipulation von Textdateien. Es ist die GNU-Implementierung von awk.

grcat zeigt die Gruppendatenbank **/etc/group** an.

igawk ermöglicht gawk, Dateien einzubinden.

pgawk ist die Profiling-Version von gawk.

pwcat zeigt die Passwortdatenbank **/etc/passwd** an.

Ncurses-5.4

Das Paket Ncurses enthält Bibliotheken für den terminalunabhängigen Zugriff auf Textbildschirme.

```
Approximate build time:  0.6 SBU
Required disk space:     27 MB
```

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

Installation von Ncurses

Bereiten Sie Ncurses zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make install
```

Setzen Sie die Ausführungsrechte für die Ncurses-Bibliothek:

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

Und 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
```

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], libpanel.[a,so]

Kurze Beschreibung

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 ist 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 terminalabhängigen 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.

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.

Vim-6.2

Das Paket Vim enthält einen sehr mächtigen Texteditor.

```
Approximate build time:  0.4 SBU
Required disk space:     34 MB
```

Vim ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 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/stable/postlfs/editors.html, dort finden Sie einige Installationshinweise.

Installation von Vim

Ändern Sie den Standardpfad von `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
```

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.

Konfigurieren von Vim

In der Voreinstellung läuft **vim** im **vi**-Kompatibilitätsmodus. Einige Leute mögen das so, aber wir würden vim lieber im **vim**-Modus ausführen (sonst hätten wir in diesem Buch nicht vim installiert, sondern gleich **vi**). Wir haben die Einstellung "nocompatible" gesetzt, um hervorzuheben, dass der neue Modus benutzt wird. Ausserdem erinnert es diejenigen, die zurück zum "compatible" Modus möchten, daran, dass diese Einstellung als erstes gemacht wird, weil sie andere Einstellungen überschreibt. Erzeugen Sie eine Standard vim-Konfigurationsdatei mit diesem Kommando:

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

set nocompatible
set backspace=2
syntax on

" End /etc/vimrc
```

```
EOF
```

set nocompatible versetzt **vim** in einen nützlicheren Betriebsmodus als den vi-kompatiblen Modus. Entfernen Sie das "no" falls Sie das alte **vi**-Verhalten nutzen möchten. *backspace=2* erlaubt das sogenannte backspacing über Zeilenumbrüche hinweg, automatisches Einrücken und das Starten von Einrückungen. *syntax on* aktiviert **vim**s Hervorheben von Syntax.

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 Beschreibung

efm_filter.pl ist 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 ist ein Skript, das vim mit less.vim startet.

mve.awk bearbeitet vim Fehler.

pltags.pl erzeugt eine Markup-Datei für Perl-Code, die mit vim benutzt werden kann.

ref prüft die Schreibweise von Argumenten.

rview ist eine eingeschränkte Version von view: es gibt keine Shell-Kommandos und view kann nicht angehalten werden.

rvim ist 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.

vim ist der Editor.

vim132 startet vim in einem Terminal mit 132-Spalten-Modus.

vim2html.pl konvertiert vim-Dokumentation zu HTML.

vimdiff editiert zwei oder drei Versionen einer Datei und zeigt die Unterschiede an.

vimm aktiviert das DEC Locator-Eingabemodell auf einem entfernten Terminal.

vimspell.sh erzeugt Syntax-highlighting-Aussagen für eine Datei, die in vim benutzt werden.

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.

M4-1.4

M4 enthält einen Makroprozessor.

```
Approximate build time:  0.1 SBU
Required disk space:     3.0 MB
```

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

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.

Und installieren Sie das Paket:

```
make install
```

Inhalt von M4

Installiertes Programm: m4

Kurze Beschreibung

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.

Bison-1.875

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

```
Approximate build time:  0.6 SBU
Required disk space:     10.6 MB
```

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

Installation von Bison

Zuerst wenden wir einen Patch an, der aus dem CVS zurückportiert wurde, um ein kleines Problem beim Kompilieren einiger Pakete zu beheben:

```
patch -Np1 -i ../bison-1.875-attribute.patch
```

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
```

Inhalt von Bison

Installierte Programme: bison und yacc

Installierte Bibliothek: liby.a

Kurze Beschreibung

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 ist ein Wrapper zu bison. Er wird benutzt, weil immer noch viele Programm yacc anstelle von bison aufrufen. Bison wird dann mit der -y Option aufgerufen.

liby.a ist 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.

Less-382

Less ist ein Textanzeigeprogramm.

```
Approximate build time:  0.1 SBU
Required disk space:     3.4 MB
```

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

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 es:

```
make install
```

Inhalt von Less

Installierte Programme: less, lessecho und lesskey

Kurze Beschreibung

less ist 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.

Groff-1.19

Groff enthält verschiedene Programme zur Verarbeitung und Formatierung von Text.

```
Approximate build time:  0.5 SBU
Required disk space:     43 MB
```

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

Installation von Groff

Groff erwartet, dass die Umgebungsvariable PAGE die Standardpapiergröße enthält. Für alle in den Vereinigten Staaten ist das untenstehende Kommando so korrekt. Wenn Ihr Standort woanders ist, ersetzen Sie besser PAGE=letter durch PAGE=A4.

Bereiten Sie Groff zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
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
```

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 Beschreibung

addftinfo liest eine troff-Schriftdatei und fügt einige font-metrische Informationen hinzu, die vom groff-System benutzt werden.

afmtodit erzeugt eine Schriftdatei für die Verwendung mit groff und grops.

eqn kompiliert Beschreibungen von Gleichungen, die in groff Eingabedateien enthalten sind, zu Kommandos, die groff versteht.

eqn2graph konvertiert eine EQN-Gleichung zu einem beschnittenen Bild.

grn ist ein groff-Präprozessor für gremlin-Dateien.

grodvi ist ein Treiber für groff, der das TeX dvi-Format erzeugt.

groff ist eine Benutzerschnittstelle für das groff-Dokumentenformatierungssystem. Normalerweise führt es das troff-Programm und einen für das Ausgabegerät passenden Postprozessor aus.

groffer zeigt groff-Dateien und Man-pages unter X und im tty an.

grog liest Dateien und errät, welche der groff Optionen -e, -man, -me, -mm, -ms, -p, -s, und -t benötigt werden und gibt das Kommando mit diesen Optionen aus.

grolbp ist ein groff-Treiber für Canon CAPSL-Drucker (Laserdrucker der LBP-4 und LBP-8 Serie).

grolj4 ist 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 ist die GNU-Implementation von tbl.

hpftodit erzeugt aus einer HP-tagged Schriftmetrik-Datei eine Schriftdatei zur Verwendung mit groff -Tlj4.

indxbib erzeugt einen invertierten Index für die bibliographischen Datenbanken, eine spezielle Datei für die 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 ist ein einfacher Präprozessor für groff.

neqn formatiert Gleichungen für die ascii-Ausgabe.

nroff ist ein Skript, das nroff-Kommandos mit groff emuliert.

pfbtops übersetzt eine Postscript-Schrift in .pfb Format zu ASCII.

pic kompiliert Beschreibungen von Bildern, die in groff oder TeX Eingabedateien vorhanden sind, zu Kommandos die von TeX oder troff verwendet werden können.

pic2graph konvertiert ein PIC-Diagramm zu einem beschnittenen Bild.

pre-grohtml übersetzt die Ausgabe von GNU-troff zu html.

post-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 Beschreibungen von Tabellen, die in troff Eingabedateien eingebettet sind, 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 groff Kommando aufgerufen, welches auch Präprozessoren und Postprozessoren in der richtigen Reihenfolge und mit den richtigen Optionen aufruft.

zsoelim ist die GNU-Implementierung von soelim.

Sed-4.0.9

Das Paket Sed enthält einen Stream-Editor.

```
Approximate build time:  0.2 SBU
Required disk space:     5.2 MB
```

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

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
```

Inhalt von Sed

Installiertes Programm: sed

Kurze Beschreibung

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

Flex-2.5.4a

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

```
Approximate build time:  0.1 SBU
Required disk space:     3.4 MB
```

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

Installation von Flex

Bereiten Sie Flex zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Zum Durchlaufen der Testsuite können Sie dieses Kommando benutzen: **`make bigcheck`**.

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 seinen Vorgänger **lex** aufzurufen. Um diesen Programmen dennoch gerecht zu werden erzeugen Sie ein kleines Shell-Skript mit dem Namen `lex`, welches **flex** im 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
```

Inhalt von Flex

Installierte Programme: flex, flex++ (Link auf flex), und lex

Installierte Bibliothek: libfl.a

Kurze Beschreibung

flex ist 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. Man nimmt für solche Aufgaben Flex, weil es einfacher ist die Muster anzugeben, als das Mustersuchprogramm selber zu schreiben.

flex++ startet eine Version von flex die exklusiv für C++-Scanner verwendet wird.

libfl.a ist die Flex-Bibliothek.

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 Meldungen in der Sprache des Anwenders ausgegeben werden.

```
Approximate build time:  0.5 SBU
Required disk space:     55 MB
```

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

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
```

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, project-id, team-address, trigger, urlget, user-email, und xgettext

Installierte Bibliotheken: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so], und libgettextsrc[a,so]

Kurze Beschreibung

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 Sprache 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 vergleich zwei `.po`-Dateien um zu prüfen das 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 Umgebung des Benutzers.

msgmerge kombiniert zwei rohe Übersetzungen in eine einzige Datei.

msgunfmt macht 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 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 ist 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 msgcomm, msgcmp, msgattrib und msgen).

libgettextsrc ist eine private Bibliothek, die die allgemeinen Routinen der verschiedenen gettext-Programme enthält. Sie sind nicht zur normalen Verwendung gedacht.

Net-tools-1.60

Die Net-tools sind eine Sammlung von grundlegenden Programmen für das Netzwerk unter Linux.

```
Approximate build time:  0.1 SBU
Required disk space:     9.4 MB
```

Net-tools ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Make.

Installation von Net-tools

Wenn Sie nicht wissen, was Sie während dem Ausführen von **make config** auf die Fragen antworten sollen, dann akzeptieren Sie einfach die Voreinstellungen. Diese sind in den meisten Fällen richtig. Sie werden gefragt werden, welche Netzwerkprotokolle Sie im Kernel aktiviert haben. Die Standardantworten aktivieren die Programme mit den gängigen Protokollen TCP, PPP und einigen anderen. Sie müssen diese Protokolle dann noch im Kernel aktivieren -- was Sie hier tun ist nur die Vorbereitung der Programme damit Sie diese Protokolle später benutzen können, aber es ist immer noch Sache des Kernels, diese Protokolle auch wirklich verfügbar zu machen.

Beheben Sie zuerst ein kleines Problem in den Quellen des **mii-tool** Hilfsprogramms:

```
patch -Np1 -i ../net-tools-1.60-miitool-gcc33-1.patch
```

Bereiten Sie nun Net-tools zum Kompilieren vor (wenn Sie generell die Standardwerte annehmen möchten die Ihnen von make config vorgeschlagen werden, dann können Sie stattdessen auch **`yes "" | make config`** ausführen):

```
make config
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make update
```

Inhalt von Net-tools

Installierte Programme: arp, dnsdomainname Llink auf hostname), domainname (Link auf hostname), hostname, ifconfig, nameif, netstat, nisdomainname (Link auf hostname), plipconfig, rarp, route, slattach, und ypdomainname (Link auf hostname)

Kurze Beschreibung

arp wird zum Manipulieren des ARP-Cache des Kernels verwendet. Normalerweise zum Hinzufügen oder Entfernen eines Eintrages oder um den Cache auszugeben.

dnsdomainname zeigt den DNS-Domänennamen des Systems an.

domainname setzt den NIS/YP-Domänennamen des Systems oder zeigt ihn an.

hostname setzt den Hostnamen des Systems oder zeigt ihn an.

ifconfig ist das Hauptwerkzeug zum Konfigurieren von Netzwerkschnittstellen.

nameif bennent Netzwerkgeräte basierend auf ihrer MAC-Adresse.

netstat zeigt Netzwerkverbindungen, Routingtabellen und Gerätestatistiken an.

nisdomainname hat die gleiche Funktion wie domainname.

plipconfig wird zur Feinkonfiguration eines PLIP-Gerätes benutzt.

rarp wird benutzt, um die RARP-Tabelle des Kernels zu manipulieren.

route wird zum Manipulieren von IP-Routingtabellen benutzt.

slattach bindet ein Netzwerkgerät an eine serielle Schnittstelle. So kann man normale Terminalverbindungen für Punkt-zu-Punkt Verbindungen mit anderen Computern benutzen.

ypdomainname hat die gleiche Funktion wie domainname.

Inetutils-1.4.2

Inetutils enthält verschiedene Programme zur grundlegenden Netzwerkunterstützung.

```
Approximate build time:  0.2 SBU
Required disk space:     11 MB
```

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

Installation von Inetutils

Wir 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. Wir 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 wir 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/stable/basicnet/inetutils.html. Beachten Sie, dass es für fast alle dieser Netzwerkserver einen besseren Ersatz gibt.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

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

```
mv /usr/bin/ping /bin
```

Inhalt von Inetutils

Installierte Programme: ftp, ping, rcp, rlogin, rsh, talk, telnet, und tftp

Kurze Beschreibung

ftp ist das ARPANET Dateiübertrangsprogramm.

ping sendet echo-request-Pakete und berichtet, wie lange die Antwort braucht.

rcp kopiert entfernte Dateien.

rlogin führt einen entfernten Login durch.

rsh führt eine entfernte Shell aus.

talk wird zum Unterhalten mit anderen Benutzern verwendet.

telnet ist eine Schnittstelle zum TELNET-Protokoll.

tftp ist das Triviale Dateiübertragungsprogramm.

Perl-5.8.4

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

```
Approximate build time:  2.9 SBU
Required disk space:     143 MB
```

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

Installieren von Perl

Wenn Sie die vollständige Kontrolle über die Art haben möchten, wie Perl sich selbst zum Installieren konfiguriert, dann können Sie stattdessen das interaktive **Configure**-Skript benutzen. Wenn Sie mit den (sinnvollen) Voreinstellungen zufrieden sind die Perl automatisch erkennt, dann 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
```

Und installieren Sie das Paket:

```
make install
```

Inhalt von Perl

Installierte Programme: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.4 (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: (zu viele um sie einzeln zu nennen)

Kurze Beschreibung

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 nach perl.

h2ph konvertiert .h C-Header Dateien zu .ph Perl-Header-Dateien.

h2xs konvertiert .h C-Header Dateien zu Perl-Erweiterungen.

libnetcfg kann zum Konfigurieren von libnet benutzt werden.

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

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 ist die Perl Installations-prüfprozedur. Damit wird geprüft, ob Perl und seine Bibliotheken korrekt installiert wurden.

piconv ist die Perl Version des Zeichensatz-Konverters **iconv**.

pl2pm ist ein Hilfsmittel zum konvertieren von Perl4 .pl Dateien zu Perl5 .pm Modulen.

pod2html konvertiert pod-Dateien in das Html-Format.

pod2latex konvertiert Dateien im Pod-Format nach LaTeX.

pod2man konvertiert Pod-Daten zu formatiertem *roff-input.

pod2text konvertiert Pod-Daten in formatierten ASCII-Text.

pod2usage gibt Benutzungshinweise aus eingebetteten Pod-Dokumenten in Dateien aus.

podchecker prüft die Syntax einer Pod-Dokumentation.

podselect zeigt ausgewählte Bereiche einer Pod-Dokumentation an.

psed ist die Perl-Version des Stream-Editors **sed**.

pstruct gibt C-Strukuten aus, die von „cc -g -S“ erzeugt wurden.

s2p übersetzt sed zu perl.

splain wird zur Analyse von Warnungen in Perl benutzt.

xsubpp konvertiert Perl XS-Kode zu C-Kode.

Texinfo-4.7

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

```
Approximate build time:  0.2 SBU
Required disk space:     17 MB
```

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

Installieren von Texinfo

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 die Komponenten einer TeX-Installation mitinstallieren:

```
make TEXMF=/usr/share/texmf install-tex
```

Die Bedeutung des make-Parameters:

- **`TEXMF=/usr/share/texmf`**: Die TEXMF Makefile-Variable enthält den Standort Ihrer TeX-Ordnerstruktur, falls Sie zum Beispiel planen später ein TeX-Paket zu installieren.

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
```

Inhalt von Texinfo

Installierte Programme: info, infokey, install-info, makeinfo, texi2dvi, und texindex

Kurze Beschreibung

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 tar und info tar.

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.

Autoconf-2.59

Autoconf erstellt Shell-Skripte, die automatisch Quelltexte konfigurieren.

```
Approximate build time:  0.5 SBU
Required disk space:     7.7 MB
```

Autoconf ist abhängig von: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

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
```

Inhalt von Autoconf

Installierte Programme: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, und ifnames

Kurze Beschreibung

autoconf ist ein Werkzeug zum Erzeugen von Shell-Skripten, die automatisch Quellcode-Pakete konfigurieren um sie an unterschiedliche Unix-System anzupassen. Die erzeugten configure-Skripte sind eigenständig und können auch dann ausgeführt werden, wenn autoconf nicht installiert ist.

autoheader ist ein Werkzeug zum Erzeugen von Vorlagedateien für C-#define-Anweisungen, die configure benutzen soll.

autom4te ist ein Wrapper zu dem M4-Makroprozessor.

autoreconf ist sehr praktisch, wenn viele autoconf-generierte configure-Skripte existieren. Das Programm ruft (wenn nötig) autoconf und autoheader immer wieder auf um so die configure-Skripte und Header-Vorlagen in einer bestimmten Ordnerstruktur neu zu erzeugen.

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 konfiguriert ist, kann dieses kleine Werkzeug helfen, herauszufinden welche Tests **configure** durchführen muss. Es kann einige Lücken in autoscan-generierten `configure.in`-Dateien füllen.

Automake-1.8.4

Automake generiert Makefile-Dateien, die danach von Autoconf benutzt werden können.

```
Approximate build time:  0.2 SBU
Required disk space:     6.8 MB
```

Automake ist abhängig von: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

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
```

Inhalt von Automake

Installierte Programme: acinstall, aclocal, aclocal-1.8, automake, automake-1.8, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree, ylwrap

Kurze Beschreibung

acinstall ist ein Skript, welches M4-Dateien im aclocal-Stil installiert.

aclocal erzeugt, basierend auf dem Inhalt von `configure.in`-Dateien, entsprechende `aclocal.m4`-Dateien.

automake ist 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.

compile ist ein Wrapper für Compiler.

config.guess ist ein Skript. Es versucht, kanonische Tripplets für das Build, den Host oder die Zielarchitektur zu erraten.

config.sub ist ein Sub-Skript zum Validieren der Konfiguration.

depcomp ist ein Skript zum Kompilieren eines Programmes, so das nicht nur die gewünschte Ausgabe erzeugt wird, sondern auch Informationen zu Abhängigkeiten.

elisp-comp kompiliert Emacs-Lisp-Kode.

install-sh ist ein Skript, welches ein Programm, ein Skript oder eine Datendatei installiert.

mdate-sh ist ein Skript, welches den Änderungszeitstempel einer Datei oder eines Ordners ausgibt.

missing ist ein Skript, welches fehlende GNU-Programme während der Installation ersetzt.

mkinstalldirs ist ein Skript zum Erzeugen einer Ordnerstruktur.

py-compile kompiliert ein Python-Programm.

symlink-tree ist ein Skript zum Erzeugen einer Symlink-Version einer Ordnerstruktur.

ylwrap ist ein Wrapper für lex und yacc.

Bash-2.05b

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, Sed.

Installieren von Bash

Bash hat ein paar Fehler die manchmal zu unerwünschten Effekten führen. Beheben Sie das Problem mit diesem Patch:

```
patch -Np1 -i ../bash-2.05b-2.patch
```

Bereiten Sie Bash nun zum Kompilieren vor:

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

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** (dies ersetzt die gerade laufende Version):

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

Beachten Sie, dass die Parameter die Sitzung zu einer interaktiven Login-Shell machen (/etc/profile wird eingelesen, falls die Datei existiert, und die entsprechend zuerst gefundene Datei ~/.bash_profile, ~/.bash_login oder ~/.profile), ausserdem wird das Hashing abgeschaltet, damit neu installierte Programme sofort gefunden werden können.

Inhalt von Bash

Installierte Programme: bash, sh (Link auf bash), und bashbug

Kurze Beschreibung

bash ist 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 ist ein Shell-Skript, welches dem Benutzer helfen soll, einen Fehlerbericht zur Bash in einem standardisierten Format zu Erstellen und per Email zu versenden.

sh ist 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 zu nachzuahmen und bleibt dabei trotzdem POSIX-Konform.

File-4.09

File ist ein kleines Werkzeug zum Identifizieren von Dateitypen.

```
Approximate build time:  0.1 SBU
Required disk space:     6.3 MB
```

File ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Zlib.

Installation von File

Bereiten Sie File zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

Inhalt von File

Installiertes Programm: file

Installierte Bibliothek: libmagic.[a,so]

Kurze Beschreibung

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 file-Programm verwendet.

Libtool-1.5.6

GNU-Libtool ist ein Skript zur Unterstützung von Bibliotheken. Libtool versteckt die Komplexität von gemeinsam benutzten Bibliotheken hinter einer konsistenten und portablen Schnittstelle.

```
Approximate build time:  1.5 SBU
Required disk space:     20 MB
```

Libtool ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

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
```

Inhalt von Libtool

Installierte Programme: libtool und libtoolize

Installierte Bibliotheken: libltdl.[a,so].

Kurze Beschreibung

libtool stellt vereinheitlichte Dienste zum Erstellen von Bibliotheken zur Verfügung.

libtoolize stellt einen Standardweg zur Verfügung um einem Paket libtool-Unterstützung hinzuzufügen.

libltdl versteckt die verschiedenen Schwierigkeiten mit Bibliotheken die dlopen verwenden.

Bzip2-1.0.2

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. Bei Textdateien erreichen Sie eine wesentlich bessere Kompressionsrate als das traditionelle Kommando **gzip**.

```
Approximate build time:  0.1 SBU
Required disk space:     3.0 MB
```

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

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
```

Wenn 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
```

Und 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}
mv /usr/bin/{bzip2recover,bzless,bzmore} /bin
ln -s bzip2 /bin/bunzip2
ln -s bzip2 /bin/bzcat
```

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 Beschreibung

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.

bzip2 komprimiert Dateien mit dem Burrows-Wheeler Blocksortierendem 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* ist die Bibliothek, die verlustlose blocksortierende Datenkompression mit Hilfe des Burrows-Wheeler-Algorythmus implementiert.

Diffutils-2.8.1

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

```
Approximate build time:  0.1 SBU
Required disk space:     7.5 MB
```

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

Installieren von Diffutils

Bereiten Sie Diffutils zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

Inhalt von Diffutils

Installierte Programme: cmp, diff, diff3, und sdiff

Kurze Beschreibung

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 sich die Dateien unterscheiden.

diff3 vergleicht drei Dateien Zeile für Zeile.

sdiff führt interaktiv zwei Dateien zusammen und gibt das Ergebnis aus.

Ed-0.2

Ed enthält einen recht spartanischen Zeileneditor.

```
Approximate build time:  0.1 SBU
Required disk space:     3.1 MB
```

Ed ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation von Ed

Anmerkung

Ed wird nicht von vielen Leuten benutzt. Ed wird installiert, weil er von dem Programm Patch verwendet wird wenn Sie einen Ed-basierten Patch installieren möchten. Das passiert allerdings sehr selten, heutzutage werden fast ausschliesslich diff-basierte Patches bevorzugt.

Ed verwendet die *mktemp*-Funktion um temporäre Dateien in `/tmp` zu erstellen, doch diese Funktion ist verwundbar (schauen Sie in die Sektion über temporäre Dateien in http://en.tldp.org/HOWTO/Secure-Programs-HOWTO/avoid-race.html). Der folgende Patch lässt Ed die *mkstemp*-Funktion verwenden, das ist der bevorzugte Weg um temporäre Dateien zu erzeugen:

```
patch -Np1 -i ../ed-0.2-mkstemp.patch
```

Bereiten Sie Ed nun zum Kompilieren vor:

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

Die Bedeutung der configure-Option:

- **`--exec-prefix=""`**: Dies erzwingt eine Installation der Programme nach **`/bin`**. Die Programme dort zu installieren ist sinnvoll für den Fall, dass `/usr` mal nicht verfügbar sein sollte.

Kompilieren Sie das Paket:

```
make
```

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

Installieren Sie das Paket:

```
make install
```

Inhalt von Ed

Installierte Programme: ed und red (Link auf ed)

Kurze Beschreibung

ed ist ein zeilenorientierter Texteditor. Er kann zum Erzeugen, Anzeigen, Verändern oder sonstigem Manipulieren von Textdateien verwendet werden.

red ist ein beschränker ed -- er kann nur Dateien im aktuellen Ordner bearbeiten und keine Shell-Kommandos ausführen.

Kbd-1.12

Kbd enthält die Dateien für das Tastaturlayout und entsprechende Werkzeuge dazu.

```
Approximate build time:  0.1 SBU
Required disk space:     12 MB
```

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

Installation von Kbd

In der Voreinstellung werden einige von Kbd's Hilfprogrammen (**setlogcons**, **setvesablank** und **getunimap**) nicht installiert. Aktivieren Sie erst die Installation dieser Programme:

```
patch -Np1 -i ../kbd-1.12-more-programs-1.patch
```

Bereiten Sie Kbd zum Kompilieren vor:

```
./configure
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

Konfigurieren der Tastatur

Es gibt nichts störenderes, als ein Linux zu benutzen auf dem ein falsches Tastaturlayout geladen ist. Wenn Sie eine Standard-US-Tastatur haben, können Sie diesen Abschnitt überspringen, denn das US-Layout wird automatisch geladen wenn Sie es nicht ändern.

Um das voreingestellte Tastaturlayout zu ändern, erstellen Sie mit dem folgenden Kommando den symbolischen Link `/usr/share/kbd/keymaps/defkeymap.map.gz`:

```
ln -s path/to/keymap /usr/share/kbd/keymaps/defkeymap.map.gz
```

Natürlich müssen Sie `pfad/zum/tastaturlayout` mit dem Pfad und Dateinamen Ihres Tastaturlayouts ersetzen. Wenn Sie zum Beispiel eine holländische Tastatur haben würden Sie `i386/qwerty/nl.map.gz` benutzen.

Eine andere Möglichkeit, dass Tastaturlayout zu setzen, ist, die keymap in den Kernel einzukompilieren. Das stellt sicher, dass Ihre Tastatur immer wie gewünscht funktioniert, selbst dann wenn Sie in den Wartungsmodus booten (indem Sie den `init=/bin/sh' Kernelparameter angeben), denn dann wird das Bootskript zum setzen des Tastaturlayouts normalerweise nicht ausgeführt.

Wenn Sie in Chapter 8[p.171] zum Kompilieren des Kernel bereit sind, führen Sie das folgende Kommando aus wenn Sie die jetzige Keymap in den Kernel patchen wollen. Sie müssten dieses Kommando allerdings jedesmal ausführen, wenn Sie einen neuen Kernel entpacken:

```
loadkeys -m /usr/share/kbd/keymaps/defkeymap.map.gz > \
    [unpacked sources dir]/linux-2.4.26/drivers/char/defkeymap.c
```

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 Beschreibung

chvt ändert das vordergründige Virtuelle Terminal.

deallocvt zieht zugewiesene unbenutzte Virtuelle Terminals zurück.

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 ist ein veraltetes Programm, das benutzerdefinierte Zeichenausgabezuweisungstabellen in den Konsoletreiber lädt. Dies wird nun durch setfont erledigt.

openvt startet ein Programm in einem neuen Virtuellen Terminal (VT).

psf* ist 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/VGA-Schriften in der Konsole.

setkeycodes lädt scancode-zu-keycode Zuweisungstabellen des Kernel. Nützlich, wenn Sie ein paar unübliche Tasten auf Ihrer Tastatur haben.

setleds setzt Tastaturoptionen und LED's. Einige Leute finden es nützlich, "Num Lock" in der Voreinstellung eingeschaltet zu haben. Mit setleds +num kann man dies erreichen.

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 aktutelle EGA/VGA-Konsole-Schrift an.

showkey zeigt Scancode, Keycode und ASCII-Code der auf der Tastatur gedrückten Taste an.

unicode_start versetzt Tastatur und Konsole in den Unicode-Modus.

unicode_stop schaltet den Unicode-Modus von Tastatur und Konsole wieder aus.

E2fsprogs-1.35

E2fsprogs stellt die Dateisystemwerkzeuge für die Benutzung des ext2-Dateisystems zur Verfügung. Auch ext3 wird unterstützt; das ist ein Journaling Dateisystem.

```
Approximate build time:  0.6 SBU
Required disk space:     48.4 MB
```

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

Installation von E2fsprogs

Es wird empfohlen, E2fsprogs ausserhalb des Quellordners zu kompilieren:

```
mkdir ../e2fsprogs-build
cd ../e2fsprogs-build
```

Bereiten Sie E2fsprogs zum Kompilieren vor:

```
../e2fsprogs-1.35/configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs
```

Die Bedeutung der configure-Parameter:

- **`--with-root-prefix=""`**: Bestimmte Programme (so 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.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie das Ergebnis testen möchten, stellen Sie sicher, dass die Datei mtab existiert. Benutzen Sie den Befehl **`touch /etc/mtab`**, dies verhindert das Fehlschlagen vieler Tests. Ausserdem "erschwindeln" Sie noch das Vorhandensein eines sehr altertümlichen Pagers um weitere Tests am Fehlschlagen zu hindern: **`ln -s /tools/bin/cat /bin/more`**. Dann können Sie den Test mit diesem Kommando starten: **`make check`**.

Installieren Sie das Meiste aus dem Paket:

```
make install
```

Und installieren Sie auch die gemeinsamen Bibliotheken:

```
make install-libs
```

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 Beschreibung

badblocks durchsucht ein Gerät (üblicherweise eine Festplatte) nach defekten Blöcken.

blkid ist ein Kommandozeilenprogramm zum Auffinden und Ausgeben der Eigenschaften eines Blockgerätes.

chattr ändert Dateiattribute auf second extended (ext2) Dateisystemen.

compile_et ist ein Fehlertabellen-Compiler. Er konvertiert eine Tabelle mit Fehlercode-Namen und Meldungen in eine C-Quelldatei, die dann mit der com_err Bibliothek verwendet werden kann.

debugfs ist 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 Reparieren von ext2- und ext3-Dateisystemen benutzt.

e2image wird zum Speichern von kritischen ext2-Dateisystemdaten in eine Datei verwendet.

e2label zeigt oder verändert das Label eines Dateisystems auf dem angegebenen Gerät.

findfs findet ein Dateisystem mit Hilfe des Label oder einer UUID.

fsck wird zum Prüfen und Reparieren von Dateisystemen verwendet. In der Voreinstellung prüft es alle Dateisysteme in `/etc/fstab`

logsave speichert die Ausgabe eines Kommandos in eine Logdatei.

lsattr listet Dateiattribute auf einem ext2-Dateisystem 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 second extended Dateisystems auf einem Gerät verwendet.

mklost+found wird benutzt um einen `lost+found`-Ordner auf einem second extended Dateisystem zu erzeugen. Es führt eine Vorzuweisung von Disk Blocks 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 second extended 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 ist 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 verwendet.

libuuid enthält Routinen zum Erzeugen von einmaligen Bezeichnern für Objekte, die hinter dem lokalen System verfügbar sein könnten.

Grep-2.5.1

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

```
Approximate build time:  0.1 SBU
Required disk space:     5.8 MB
```

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

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
```

Inhalt von Grep

Installierte Programme: egrep (Link auf grep), fgrep (Link auf grep), und grep

Kurze Beschreibung

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.

Grub-0.94

Das Paket Grub enthält den GRand Unified Bootloader.

```
Approximate build time:  0.2 SBU
Required disk space:     10 MB
```

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

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 nun Grub zum Kompilieren vor:

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

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
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.

Inhalt von Grub

Installierte Programme: grub, grub-install, grub-md5-crypt, grub-terminfo, und mbchk

Kurze Beschreibung

grub ist 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 unübliches Terminal haben.

mbchk prüft das Format eines Multiboot-Kernel.

Gzip-1.3.5

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

```
Approximate build time:  0.1 SBU
Required disk space:     2.6 MB
```

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

Installation von Gzip

Bereiten Sie Gzip zum Kompilieren vor:

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

Das Programm **gzexe** bekommt den Pfad zu **gzip** fest eingebaut. Da wir diese Datei im nachhinein verschieben, müssen wir mit dem folgenden Kommando sicherstellen, dass der korrekte Pfad in die Binärdatei geschrieben wird:

```
cp gzexe.in{,.backup}
sed 's%"BINDIR"%/bin%' gzexe.in.backup > 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
```

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 Beschreibung

gunzip dekomprimiert gzip-Dateien.

gzexe wird zum Erzeugen von selbstentpackenden ausführbaren Dateien verwendet.

gzip komprimiert Dateien mit dem Lempel-Ziv (LZ77) Algorythmus.

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 komprimiert Dateien im compress-Format erneut in das gzip-Format -- .Z zu .gz.

Man-1.5m2

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

```
Approximate build time:  0.1 SBU
Required disk space:     1.9MB
```

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

Installation von Man

Wir nehmen zuerst drei Anpassungen an den Quellen zu vor.

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

```
patch -Np1 -i ../man-1.5m2-80cols.patch
```

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

```
sed -i 's/-is/&R/' 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%#&%' src/man.conf.in
```

Bereiten Sie Man nun zum Kompilieren vor:

```
./configure -default -confdir=/etc
```

Die Bedeutung der configure-Parameter:

- **`-default`**: Veranlasst das configure-Skript, eine sorgfältige Auswahl an Voreinstellungen auszuwählen. Zum Beispiel: Nur englische Manpages, keine Nachrichtenkataloge, man ohne suid-Bit, Unterstützung komprimierter Manpages, komprimieren von cat-Seiten, erstellen von cat-Seiten wenn der zugehörige Ordner existiert, FHS-Konformität durch ablegen der cat-Seiten unter /var/cache/man sofern der Ordner existiert.
- **`-confdir=/etc`**: Durch diese Option sucht das Programm **man** seine Konfigurationsdatei `man.conf` im Ordner `/etc`.

Kompilieren Sie das Paket:

```
make
```

Und installieren Sie es:

```
make install
```

Anmerkung

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

Wenn Sie weitergehende Informationen zur Kompression von Manpages haben möchten, schauen Sie am besten im BLFS-Buch unter http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/compressdoc.html nach.

Inhalt von Man

Installierte Programme: apropos, makewhatis, man, man2dvi, man2html, und whatis

Kurze Beschreibung

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 nach HTML.

whatis durchsucht die whatis-Datenbank und zeigt eine kurze Beschreibung zu den Systemkommandos an, die das übergebene Stichwort als separates Wort enthalten.

Make-3.80

Das Paket Make enthält Programme zum Kompilieren umfangreicher Pakete.

```
Approximate build time:  0.2 SBU
Required disk space:     8.8 MB
```

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

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
```

Inhalt von Make

Installiertes Programm: make

Kurze Beschreibung

make erkennt automatisch, welche Teile eines großen Programms neu kompiliert werden müssen und führt automatisch die notwendigen Kommandos aus.

Modutils-2.4.27

Das Modutils Paket enthält diverse Programme zur Verwaltung von Kernel-Modulen.

```
Approximate build time:  0.1 SBU
Approximate build time:  2.9 MB
```

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

Installation von Modutils

Bereiten Sie Modutils zum Kompilieren vor:

```
./configure
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

Inhalt von Modutils

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 Beschreibung

depmod erzeugt, basierend auf den Symbolen in existierenden Modulen, eine Abhängigkeitsdatei. Diese Datei wird von modprobe benutzt, um benötigte Module automatisch nachzuladen.

genksyms erzeugt Modulversionsinformationen.

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 zeigt an, welche Module geladen sind.

modinfo untersucht eine Objektdatei die mit einem Kernelmodul assoziiert ist und zeigt die darin verfügbaren Informationen an.

modprobe benutzt eine von depmod erzeugte Abhängigkeitsdatei, um benötigte Module automatisch nachzuladen.

rmmod entlädt ein Modul aus dem laufenden Kernel.

Patch-2.5.4

Das Paket Patch enthält ein Programm zum Modifizieren von Dateien.

```
Approximate build time:  0.1 SBU
Required disk space:     1.9 MB
```

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

Installieren von Patch

Bereiten Sie Patch zum Kompilieren vor (die Präprozessor Option -D_GNU_SOURCE wird nur auf PowerPC-Plattformen benötigt. Auf anderen Architekturen können Sie sie weglassen.):

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

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

Inhalt von Patch

Installiertes Programm: patch

Kurze Beschreibung

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. Wenn man Patche anstelle von komplett neuen Tar-Archiven verwendet um Programmquellen auf dem laufenden zu halten, kann man eine Menge Downloadzeit sparen.

Procinfo-18

Procinfo sammelt Systeminformationen wie zum Beispiel Speicherausnutzung und IRQ-Nummern aus dem /proc-Ordner und gibt die Daten sinnvoll formatiert aus.

```
Approximate build time:  0.1 SBU
Required disk space:     0.2 MB
```

Procinfo ist abhängig von: Binutils, GCC, Glibc, Make, Ncurses.

Installation von Procinfo

Kompilieren Sie Procinfo:

```
make LDLIBS=-lncurses
```

Die Bedeutung des make-Parameters:

- **`LDLIBS=-lncurses`**: Das weist Procinfo an, die Bibliothek `libncurses` anstelle der längst veralteten `libtermcap` zu verwenden.

Installieren Sie das Paket:

```
make install
```

Inhalt von Procinfo

Installierte Programme: lsdev, procinfo und socklist

Kurze Beschreibung

lsdev listet die in Ihrem System verfügbaren Geräte, die zugehörigen Interrupts und IO-Ports auf.

procinfo zeigt eine Übersicht über einige Informationen im virtuellen Proc-Dateisystem an.

socklist listet alle offenen Sockets auf und zeigt ihren Typ, Portnummer und andere Details an.

Procps-3.2.1

Procps enthält Programme zur Überwachung und Steuerung von Systemprozessen. Die Informationen zu den Prozessen holt Procps aus dem /proc-Ordner.

```
Approximate build time:  0.1 SBU
Required disk space:     6.2 MB
```

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

Installation von Procps

Kompilieren Sie nun Procps:

```
make
```

Installieren Sie es:

```
make install
```

Und entfernen Sie einen toten symolischen Link auf eine Bibliothek:

```
rm /lib/libproc.so
```

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 Beschreibung

free gibt die Menge an freiem und benutzten Arbeitsspeicher aus, sowohl physischem als auch swap.

kill wird benutzt um Signale an Prozesse zu senden.

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 zeigt eine Übersicht der laufenden Prozesse an.

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. Es 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, trapsm 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.

Psmisc-21.4

Das Paket Psmisc enthält Programme zum Anzeigen von Prozessinformationen.

```
Approximate build time:  0.1 SBU
Required disk space:     2.2 MB
```

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

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 Psmisc Programme häufig in Bootskripten verwendet werden, müssen sie verfügbar sein, auch wenn das `/usr`-Dateisystem noch nicht eingehängt ist.

Kompilieren Sie das Paket:

```
make
```

Installieren Sie es:

```
make install
```

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

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

In der Voreinstellung wird Psmisc's **pidof**-Programm 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
```

Inhalt von Psmisc

Installierte Programme: fuser, killall, pstree, und pstree.x11 (Link auf pstree)

Kurze Beschreibung

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.

Shadow-4.0.4.1

Das Paket Shadow enthält Programme zur sicheren Verwaltung von Kennwörtern.

```
Approximate build time:  0.4 SBU
Required disk space:     11 MB
```

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

Installation von Shadow

Shadow baut den Pfad zu **passwd** in die Binärdatei selbst fest ein, aber macht das leider nicht ganz korrekt. Wenn die Datei **passwd** beim Installieren von Shadow nicht vorhanden ist, nimmt das Paket an, dass es nach `/bin/passwd` gehört, installiert es dann aber nach `/usr/bin/passwd`. Das führt zu dem Fehler, dass `/bin/passwd` nicht gefunden werden kann. Um diesen Fehler zu umgehen erstellen Sie eine dymmy `passwd`-Datei, damit der Pfad korrekt eingebunden wird:

```
touch /usr/bin/passwd
```

Bereiten Sie Shadow nun 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
```

Kompilieren Sie das Paket:

```
make
```

Und installieren Sie es:

```
make install
```

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

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

Wir möchten die voreingestellte *crypt*-Methode zu *MD5* ändern, welche theoretisch sicherer ist. Ausserdem erlaubt sie Passwörter mit mehr als 8 Zeichen. Ausserdem müssen wir den alten Ort für die Benutzermailboxen von `/var/spool/mail` zu `/var/mail` ändern. Das erledigen wir, indem wir die Konfigurationsdatei gleich beim Kopieren an die richtige Stelle ändern (benutzen Sie am besten "Kopieren und Einfügen" um den Befehl auszuführen):

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

Verschieben Sie zwei Links an ihre korrekte Stelle:

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

Und verschieben Sie Shadow's dynamische Bibliotheken an eine bessere 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 das von Shadow installierte wieder löschen:

```
rm /bin/groups
```

Konfigurieren von Shadow

Dieses Paket enthält Werkzeuge zum Bearbeiten, Hinzufügen und Löschen von Benutzerpasswörtern. Wir werden hier nicht erläutern was genau das *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 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.

Vergeben des Passworts für root

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

```
passwd root
```

Inhalt von Shadow

Installierte Programme: chage, chfn, chpasswd, chsh, dpasswd, 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

Kurze Beschreibung

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.

dpasswd wird zum Ändern des Einwähl-Kennwortes eines Benutzers verwendet.

expiry prüft, ob ein Kennwort abgelaufen ist und setzt eine entsprechende Regelung durch.

faillog wird zum Untersuchen der Logdatei über fehlgeschlagene Logins, eine maximale Fehlerzahl vor der Sperrung eines Kontos zu setzen und 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 group-Dateien `/etc/group` und `/etc/gshadow`.

grpconv erzeugt oder aktualisiert die shadow-group-Datei 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 ist ein Dämon, der Beschränkungen auf die Login-Zeit und -Ports durchsetzt.

mkpasswd verschlüsselt ein Passwort mit einer angegebenen Störung.

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.

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.

libmisc...

libshadow enthält Funktionen, die von den meisten der Programme in diesem Paket verwendet werden.

Sysklogd-1.4.1

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

```
Approximate build time:  0.1 SBU
Required disk space:     0.5 MB
```

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

Installation von Sysklogd

Kompilieren Sie Sysklogd:

```
make
```

Installieren Sie es:

```
make install
```

Konfigurieren von Sysklogd

Erstellen Sie die neue Datei `/etc/syslog.conf` indem Sie folgendes Kommando eingeben:

```
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
```

Inhalt von Sysklogd

Installierte Programme: klogd und syslogd

Kurze Beschreibung

klogd ist ein Systemdämon zum Abfangen und Protokollieren von Kernelnachrichten.

syslogd protokolliert die Nachrichten von Systemprogrammen mit. Jeder Logeintrag enthält mindestens einen Datumsstempel und den Hostnamen, und üblicherweise auch den Programmnamen, aber das hängt davon ab wie vertrauensselig der Logdämon konfiguriert wurde.

Sysvinit-2.85

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

```
Approximate build time:  0.1 SBU
Required disk space:     0.9 MB
```

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

Installation von Sysvinit

Wenn Runlevel gewechselt werden (zum Beispiel beim Herunterfahren des Systems), sendet **init** Signale an alle Programme die es gestartet hat. **Init** gibt „Sending processes the TERM signal“ auf den 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“:

```
cp src/init.c{,.backup}
sed 's/Sending processes/& started by init/g' \
    src/init.c.backup > src/init.c
```

Kompilieren Sie Sysvinit:

```
make -C src
```

Und installieren Sie es:

```
make -C src install
```

Konfigurieren 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 tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
EOF
```

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 Beschreibung

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 ist die Mutter aller Prozesse. Es liest seine Kommandos aus `/etc/inittab`, die ihm normalerweise sagt, welche Skripte in einem Runlevel gestartet werden sollen und wieviele gettys gestartet werden sollen.

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 Loginversuche, 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 nacht 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 dem Superbenutzer, sich einzuloggen. Es wird normalerweise 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.

Tar-1.13.94

Das Paket Tar enthält ein Archivprogramm.

```
Approximate build time:  0.2 SBU
Required disk space:     10 MB
```

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

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
```

Inhalt von Tar

Installierte Programme: rmt und tar

Kurze Beschreibung

rmt wird zum entfernten Manipulieren von magnetischen Bandlaufwerken verwendet und benutzt dafür Interprozesskommunikation.

tar wird zum Erzeugen und Extrahieren von Dateien aus einem Archiv verwendet.

Util-linux-2.12a

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

```
Approximate build time:  0.2 SBU
Required disk space:     16 MB
```

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

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:

```
cp hwclock/hwclock.c{,.backup}
sed 's%etc/adjtime%var/lib/hwclock/adjtime%' \
    hwclock/hwclock.c.backup > hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

Installieren von Util-linux

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 installiert wird.
- **`HAVE_SLN=yes`**: Verhindert, dass das Programm **sln** (eine statisch gelinkte Version von **ln**, bereits durch Glibc installiert) erneut installiert wird.

Installieren Sie das Paket:

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

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, swapoff (Link auf swapon), swapon, tunelp, ul, umount, vidmode (Link auf rdev), whereis, und write

Kurze Beschreibung

agetty öffnet einen tty-Port, fragt nach dem Login-Namen und startet das login-Programm.

arch gibt die Systemarchitektur aus.

blockdev ermöglicht den Aufruf von Blockgeräte-ioctls an der Kommandozeile.

cal zeigt einen einfachen Kalender an.

cfdisk wird zum Manipulieren der Partitionstabelle eines Gerätes benutzt.

chkdupexe findet Duplikate von ausführbaren Dateien.

col filtert Rückwärts-Zeilenvorschübe aus.

colcrt filter 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 kann zum Bearbeiten der Partitionstabelle auf einem Gerät verwendet werden.

fsck.cramfs führt eine Konsistenzprüfung auf dem Cramfs Dateisystem durch.

fsck.minix führt eine Konsistenzprüfung auf dem 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 eine IPC-Ressource.

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, 128-bit hexadezimale Zufallszahlen, für xauth.

mkfs erzeugt ein Dateisystem auf einem Gerät (üblicherweise einer Festplattenpartition).

mkfs.bfs erzeugt ein SCO-bfs-Dateisystem.

mkfs.cramfs erzeugt ein cramfs-Dateisystem.

mkfs.minix erzeugt ein Minix-Dateisystem.

mkswap initialisiert ein Gerät oder eine Datei als Auslagerungsbereich.

more ist 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 benutzt werden, um die Größe einer RAM-Disk in einem bootbaren Abbild zu setzen.

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 setzt Terminal-Attribute.

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 ist 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. Zumindest sofern der Benutzer solche Nachrichten nicht deaktiviert hat.

GCC-2.95.3

```
Approximate build time:  1.5 SBU
Approximate build time:  130 MB
```

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.

Dies ist eine ältere Version von GCC die wir nur installieren, um damit in Chapter 8[p.171] den Linux-Kernel zu kompilieren. Diese Version wird von den Kernel-Entwicklern empfohlen wenn Sie absolute Stabilität brauchen. Neuere Versionen von GCC wurden nicht so intensiv mit dem Linux-Kernel getestet. Eine neuere Version funktioniert höchstwahrscheinlich, dennoch folgen wir dem Rat der Kernel-Entwickler und benutzen hier diese Version um den Kernel zu kompilieren.

Anmerkung

Wir installieren hier nicht den C++-Compiler und seine Bibliotheken. Dennoch könnten Sie Gründe haben, diese zu installieren. Mehr Informationen dazu finden Sie unter http://www.linuxfromscratch.org/blfs/view/stable/general/gcc2.html.

Wir installieren diese alte Version von GCC im nicht-standard-Prefix `/opt` um nicht mit dem auf dem System bereits unter `/usr` installierten GCC durcheinander zu geraten.

Wenden Sie die Patche an und nehmen Sie eine kleine Anpassung vor:

```
patch -Np1 -i ../gcc-2.95.3-2.patch
patch -Np1 -i ../gcc-2.95.3-no-fixinc.patch
patch -Np1 -i ../gcc-2.95.3-returntype-fix.patch
echo timestamp > gcc/cstamp-h.in
```

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

```
mkdir ../gcc-2-build
cd ../gcc-2-build
```

Kompilieren und installieren Sie den Compiler:

```
../gcc-2.95.3/configure --prefix=/opt/gcc-2.95.3 \
    --enable-shared --enable-languages=c \
    --enable-threads=posix
make bootstrap
make install
```

Informationen zu Debugging Symbolen

Die meisten Programme und Bibliotheken werden in der Voreinstellung mit Debugging-Symolen kompiliert (mit der Option **gcc** *-g*). Wenn Sie ein Programm oder eine Bibliothek debuggen die mit debugging Symbolen kompiliert wurde, dann 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 (/lib und /usr/lib) ohne Debugging-Symbole: 16 MB

Die Grössen variieren ein wenig, abhängig davon welchen Compiler und welche C-Bibliothek Sie benutzen. 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 eine Menge Platz gespart werden indem wir die debugging Symbole entfernen. 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.

Erneutes Stripping

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

Die meisten Leute haben keine Probleme mit dem unten stehenden Kommando. 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
```

Führen Sie nun dieses Kommando aus, um Binärdeiten und Bibliotheken mit strip zu bearbeiten:

```
/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 wirklich 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.

Aufräumen

Wenn Sie von nun an die chroot Umgebung verlassen und wieder betreten möchten, sollten Sie folgendes modifiziertes 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 wir 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 wir 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 Chapter 5[p.26], 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 Boot Skripte

Einführung

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

Wir haben den System-V Init-Stil gewählt, weil er weit verbreitet ist und die meisten gut damit umgehen können. Wenn Sie etwas anderes bevorzugen, Marc Heerdink hat eine Aneitung zu BSD-Stil Init-Skripten geschrieben, Sie finden das Dokument unter http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt. Und wenn Sie etwas radikaleres möchten, durchsuchen Sie die LFS-Mailinglisten nach depinit.

Wenn Sie sich für etwas ganz anderes entschieden haben, können Sie das nachfolgende Kapitel überspringen und direkt bei Chapter 8[p.171] weitermachen.

LFS-Bootscripts-2.0.5

Das Paket LFS-Bootskripte enthält einige Boot-Skripte.

```
Approximate build time:  0.1 SBU
Required disk space:     0.3 MB
```

LFS-Bootskripte sind abhängig von: Bash, Coreutils.

Installation von LFS-Bootskripte

Die Installation der Boot-Skripte ist sehr einfach:

```
make install
```

Inhalt von LFS-Bootskripte

Installierte Skripte: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, und template

Kurze Beschreibung

Das **checkfs**-Skript prüft Dateisysteme bevor sie eingehängt werden (mit der Ausnahme von journal- und netzwerkbasierten Dateisystemen).

Das **cleanfs**-Skript 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`.

Das **functions**-Skript enthält Funktionen, die gemeinsam von verschiedenen Skripten genutzt werden, wie z. B. Fehler- oder Statusprüfung.

Das **halt**-Skript fährt das System herunter.

Das **ifdown**- und **ifup**-Skript unterstützen das network-Skript in Bezug auf Netzwerkgeräte.

Das **loadkeys**-Skript läd das Tastaturlayout, das Sie für Ihre Tastatur konfiguriert haben.

Das **localnet**-Skript setzt den Hostnamen und das lokale Loopback-Gerät auf.

Das **mountfs**-Skript hängt alle Dateisysteme ein, die nicht als *noauto* markiert und nicht netzwerkbasiert sind.

Das **mountkernfs**-Skript wird zum Einhängen von Dateisystemen benutzt, die der Kernel bereitstellt (z. B. /proc).

Das **network**-Skript macht Netzwerkschnittstellen wie z. B. Netzwerkkarten verfügbar und richtet - wenn nötig - das Standard-Gatway ein.

Das **rc**-Skript ist das Haupt-Runlevel-Kontrollskript. Es ist dafür verantwortlich, alle anderen Skripte eins nach dem anderen in der richtigen Reihenfolge auszuführen.

Das **reboot**-Skript startet das System neu.

Das **sendsignals**-Skript stellt sicher, dass jeder Prozess beendet wird, bevor das System herunterfährt oder neu startet.

Das **setclock**-Skript setzt die Kernelzeit auf lokale Zeit, falls die Hardware-Uhr nicht auf GMT-Zeit eingestellt ist.

Das **static**-Skript stellt Funktionen zum Zuweisen einer statischen IP-Adresse an ein Netzwerkgerät zur Verfügung.

Das **swap**-Skript aktiviert und deaktiviert Swap-Dateien und -Partitionen.

Das **sysklogd**-Skript startet und stoppt die System- und Kernel-Log-Dämonen.

Das **template**-Skript ist eine Vorlage, die Sie verwenden können, um Ihre eigenen Bootskripte für eigene Dämonen zu schreiben.

Wie funktioniert der Bootvorgang mit diesen Skripten?

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 kdm)
6: Startet den Computer neu
```

Das Kommando zum wechseln des Runlevel ist **init <Runlevel>**, wobei <Runlevel> der Runlevel ist, in den Sie wechseln möchten. Zum Neustarten des Computers würde ein Benutzer zum Beispiel **init 6** eingeben. Das **reboot**-Kommando ist nur ein Alias darauf, genauso wie das Kommando **halt** ein Alias auf **init 0** ist.

Unter `/etc/rc.d` finden 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 *stop*-Parameter aufgerufen um etwas zu beenden. Die Logik 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 ruhig 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 (nun, wir halten es für gut -- Sie mögen es aber vielleicht hassen).

Konfigurieren des setclock-Skript

Das setclock-Skript liest die Zeit aus der Hardware-Uhr des Computers (auch bekannt als BIOS- oder CMOS-Uhr) und konvertiert sie mit Hilfe von `/etc/localtime` (falls die Hardware Uhr auf GMT gestellt ist) in lokale Zeit. Wenn die Hardware-Uhr auf lokale Zeit eingestellt ist, wird die Zeit nicht konvertiert. Es gibt leider keinen Weg, automatisch herauszufinden, ob die Hardware-Uhr auf GMT gestellt ist oder nicht, deshalb müssen wir das selber konfigurieren.

Ändern Sie den Wert von UTC zu 0 (Null), wenn Ihre Hardware-Uhr nicht auf GMT 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 TZ-Umgebungsvariable.

Brauche ich das loadkeys-Skript?

Falls Sie sich in Chapter 8[p.171] entschieden haben, das Tastaturlayout in den Kernel fest einzubinden, dann brauchen Sie genau genommen dieses Skript nicht, weil der Kernel das Tastaturlayout bereits für Sie läd. Sie können es aber trotzdem ausführen, es schadet nicht. Es kann sogar hilfreich sein, zum Beispiel wenn Sie viele verschiedene Kernel haben und das Tastaturlayout nicht immer fest einkompilieren wollen.

Wenn Sie sich entscheiden, dass loadkeys-Skript nicht laufen zu lassen oder es einfach nicht brauchen, dann löschen Sie einfach den symbolischen Link `/etc/rc.d/rcsysinit.d/S70loadkeys`.

Konfigurieren 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 `sysklogd`-Skript entsprechend. Weitere Informationen erhalten Sie mit **`man syslogd`**.

Konfigurieren des localnet-Skript

Eine Teilaufgabe des localnet-Skript ist das Einstellen des Hostnamen. Dies muss in /etc/sysconfig/network konfiguriert 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.

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 konfigurieren. 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.

Wenn Sie keine Netzwerkkarte konfigurieren, erzeugen Sie `/etc/hosts` mit diesem Kommando:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 <value of HOSTNAME>.example.org <value of HOSTNAME> localhost

# End /etc/hosts (no network card version)
EOF
```

Wenn Sie eine Netzwerkkarte konfigurieren möchten, erzeugen Sie `/etc/hosts` mit diesem 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).

Konfigurieren des network-Skript

Dieser Arbschnitt ist nur interessant, wenn Sie eine Netzwerkkarte konfigurieren 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`)

Konfiguration des Standard-Gateway

Wenn Sie in einem Netzwerk sind, müssen Sie wahrscheinlich das Standard-Gateway für diesen Rechner konfgurieren. Fügen Sie den korrekten Wert in die Datei /etc/sysconfig/network ein:

```
cat >> /etc/sysconfig/network << "EOF"
GATEWAY=192.168.1.2
GATEWAY_IF=eth0
EOF
```

Die Werte für GATEWAY und GATEWAY_IF müssend angepasst werden, so das sie mit Ihrem Netzwerk funktionieren. GATEWAY enthält die IP-Adresse für das Standard-Gateway, und GATEWAY_IF enthält das Netzwerkgerät über welches das Standard-Gateway erreicht werden kann.

Erstellen der Netzwerkgeräte-Konfigurationsdateien

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 wir neue Dateien. Das folgende Kommando erzeugt die Beispieldatei ifconfig.eth0:

```
cat > /etc/sysconfig/network-devices/ifconfig.eth0 << "EOF"
ONBOOT=yes
SERVICE=static
IP=192.168.1.1
NETMASK=255.255.255.0
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).

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

nameserver <IP-Adresse des Nameservers>

# End /etc/resolv.conf
```

```
EOF
```

Natürlich müssen Sie <IP-Adresse des Nameservers> durch die echte IP-Adresse Ihres 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.

Kapitel 8. Das LFS-System bootfähig machen

Einführung

Dieses Kapitel macht Ihr LFS bootfähig. In diesem Kapitel erstellen Sie die fstab-Datei, erstellen einen neuen Kernel für Ihr LFS-System und installieren den Grub Bootloader damit Sie Ihr LFS-System zum booten auswählen können.

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  fs-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
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 `reiserfs`. 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 Ihrer `fstab` hinzufügen wollen. Eine zum Beispiel zum Verwenden von USB-Geräten:

```
usbfs       /proc/bus/usb  usbfs    defaults    0     0
```

Diese Option funktioniert natürlich nur, wenn Sie die entsprechende Unterstützung in den Kernel einkompilieren.

Linux-2.4.26

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

```
Approximate build time:  Mit allen Voreinstellungen: 4.20 SBU
Required disk space:     Mit allen Voreinstellungen: 181 MB
```

Linux ist abhängig von: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Installation des Kernel

Kompilieren und Installieren des Kernels sind nur ein paar Schritte: Konfiguration, 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 Kernelsauszuführen. Sie sollten sich nicht darauf verlassen, dass die Quellen nach dem Entpacken sauber sind.

Konfigurieren Sie den Kernel mit der menügeführten Benutzeroberfläche:

```
make menuconfig
```

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

Wenn Sie möchten, können Sie die Kernelkonfiguration überspringen und einfach die Kernel-Konfigurationsdatei `.config` von Ihrem Host-system nach `linux-2.4.26` kopieren (falls sie verfügbar ist). Das empfehlen wir allerdings nicht, Sie sind besser dran, wenn Sie alle Konfigurationsmenüs durchsehen und Ihre eigene Kernelkonfiguration frisch einrichten.

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

Überprüfen Sie die Abhängigkeiten und erzeugen Sie die entsprechenden Abhängigkeitsdateien:

```
make CC=/opt/gcc-2.95.3/bin/gcc dep
```

Kompilieren Sie das Kernel-Abbild:

```
make CC=/opt/gcc-2.95.3/bin/gcc bzImage
```

Kompilieren Sie die Treiber, die als Modul konfiguriert wurden:

```
make CC=/opt/gcc-2.95.3/bin/gcc modules
```

Wenn Sie planen, Kernel-Module zu verwenden, dann brauchen Sie die Datei `/etc/modules.conf`. Informationen betreffend Module und Kernelkonfiguration im allgemeinen finden Sie in der Kernel-Dokumentation; Sie finden Sie im Ordner `linux-2.4.26/Documentation`. Die modules.conf-Man-page und das Kernel-HOWTO unter http://www.tldp.org/HOWTO/Kernel-HOWTO.html könnten für Sie auch von Interesse sein.

Installieren Sie die Module:

```
make CC=/opt/gcc-2.95.3/bin/gcc modules_install
```

Wenn Sie viele Module aber dafür wenig Festplattenspeicher haben, können Sie die Module strippen und komprimieren. Für die meisten ist 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.

Weil ohne Dokumentation nichts komplett ist, erzeugen Sie die Man-pages die mit dem Kernel kommen:

```
make mandocs
```

Und installieren diese:

```
cp -a Documentation/man /usr/share/man/man9
```

Das Kompilieren des Kernel ist nun abgeschlossen, aber einige der erzeugten Dateien befinden sich noch im Quellordner. Um die Installation abzuschließen, müssen Sie noch zwei 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
```

`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
```

`.config` ist die Kernel-Konfigurationsdatei die durch das Kommando **make menuconfig** erzeugt wurde. Sie enthält alle Konfigurationsoptionen für den soeben kompilierten Kernel. Es ist sinnvoll, diese Datei aufzubewahren:

```
cp .config /boot/config-lfskernel
```

Beachten Sie bitte, dass die Dateien im Kernel-Quellordner nicht *root* gehören. Immer wenn Sie ein Paket als *root*-Benutzer entpacken (so wie wir 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-2.4.26` anwenden damit alle Dateien dem *root*-Benutzer gehören.

Inhalt von Linux

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

Kurze Beschreibung

Der *Kernel* ist 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.

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

Die Datei `System.map` enthält eine Liste von Adressen und Symbolen. Sie ordnet Einstiegspunkte und Adressen aller Funktionen und Datenstrukturen dem entsprechenden Kernel zu.

Das LFS-System bootfähig machen

Ihr frisches LFS-System ist nun beinahe fertig. Eines der letzten Dinge ist, sicherzustellen, dass es booten kann. Die untenstehende Anleitung gilt nur auf Computern mit IA32-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 wir den Grub Bootloader in Vorbereitung zu diesem Schritt installiert. In dieser Prozedur müssen ein paar Grub-Dateien an spezielle Orte auf der Festplatte kopiert werden. Bevor wir dazu kommen, empfehlen wir, dass 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. Wir starten 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 nehmen wir an, dass Ihre root-Partition `hda4` ist.

Sagen Sie Grub zuerst, wo er seine `stage{1,2}`-Dateien findet -- 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 zu verwalten (MBR = Master Boot Record). In dem Fall würde es Sinn machen, Grub in den „Bootsektor“ Ihrer LFS-Partition zu installieren, das Kommando würde dann lauten: **`setup (hd0,3)`**.

Sagen Sie Grub nun, dass er sich in den MBR von `hda` installieren soll:

```
setup (hd0)
```

Wenn alles in Ordnung ist, wird Grub nun berichten, dass er seine Dateien in `/boot/grub` findet. Das ist alles soweit:

```
quit
```

Nun müssen wir die Datei „menu.lst“ 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 5.1.1
root (hd0,3)
kernel --no-mem-option /boot/lfskernel root=/dev/hda4
EOF
```

Anmerkung

In der Voreinstellung wird Grub dem Kernel automatisch die Zeile „mem=xxx“ übergeben. Manchmal erkennt Grub allerdings den Hauptspeicher nicht richtig, und das kann unter bestimmten Umständen zu Problemen führen. Es ist gut, diese Option zu deaktivieren und den Kernel den Hauptspeicher selber erkennen zu lassen. Daher verwenden wir oben die Option *--no-mem-option*.

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 **info grub** Ihnen nicht alle Informationen gibt, die Sie brauchen, finden Sie mehr dazu auf den Grub-Webseiten unter http://www.gnu.org/software/grub/.

Kapitel 9. Das Ende

Das Ende

Herzlichen Glückwunsch! Sie sind fertig mit der Installation Ihres eigenen LFS-Systems. Vielleicht war das eine lange Prozedur, aber wir hoffen es war die Zeit Wert. 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 5.1.1 > /etc/lfs-release
```

Lassen Sie sich zählen

Möchten Sie nun, wo 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 in Ihr LFS booten...

Neustarten des Systems

Jetzt, wo sämtliche Software installiert wurde, wird es Zeit Ihren Computer neu zu starten. Ein paar Dinge sollten Sie aber noch beachten. Das in diesem Buch installierte System ist sehr minimal, wahrscheinlich fehlen Ihnen einige Funktionen zur weiteren Konfiguration des Systems. Während wir uns nun noch in der chroot-Umgebung befinden, könnten Sie einige Pakete aus dem BLFS-Buch installieren und damit Ihre Situation nach dem Neustart deutlich verbessern. Zum Beispiel können Sie durch installieren des Text-Browsers Lynx das BLFS-Buch auf der einen Konsole lesen, während Sie auf der anderen weitere Pakete installieren. Mit dem GPM-Paket können Sie Kopieren und Einfügen mit der Maus benutzen. Ausserdem könnte es in Ihrem Netzwerk sinnvoll sein, das Paket dhcpcd oder ppp zu installieren.

Nun lassen Sie uns Ihr System Neustarten, als erstes verlassen Sie die chroot-Umgebung:

```
logout
```

Hängen Sie die virtuellen Dateisysteme aus:

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

Und hängen Sie das LFS-Dateisystem aus:

```
umount $LFS
```

Wenn Sie sich zu Beginn für mehrere Partitionen entschieden haben, müssen Sie die anderen Partitionen aushängen bevor Sie $LFS wie folgt aushängen:

```
umount $LFS/usr
umount $LFS/home
umount $LFS
```

Und 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 5.1.15.1.1* automatisch booten.

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

Was nun?

Vielen Dank, dass Sie das LFS-Buch gelesen haben. Wir hoffen, dass Sie das Buch nützlich fanden und das es Ihre Zeit wert war.

Jetzt wo Sie mit der Installation von LFS fertig sind, fragen Sie sich vielleicht: „Was 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 Chapter 1 - Mailing lists[p.6] 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/.

Index of packages and important installed files

Packages

Programs

Libraries

Scripts

Others

/etc/services:	Iana-Etc-1.00[p.96]	
/etc/syslog.conf:	Konfigurieren von Sysklogd[p.150]	
/etc/vim:	Konfigurieren von Vim[p.101]	
/var/log/btmp:	Erstellen der Dateien passwd, group und der Logdateien[p.72]	
/var/log/lastlog:	Erstellen der Dateien passwd, group und der Logdateien[p.72]	
/var/log/wtmp:	Erstellen der Dateien passwd, group und der Logdateien[p.72]	
/var/run/utmp:	Erstellen der Dateien passwd, group und der Logdateien[p.72]	
Kernel-Header:	Linux-2.4.26[p.173]	description[p.174]
Man-pages:	Man-pages-1.66[p.76]	description[p.76]