Linux From Scratch

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Linux From Scratch

Version 5.0

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

Vorwort

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 Cdrom oder Bootdisk für die Installation der Basiswerkzeuge zu benutzen. Ich würde mein gerade laufendes Linux System verwenden, um mein eigenes Linux zu entwickeln.

Diese Idee schien zu diesem Zeitpunkt sehr schwierig und oft schien sie sogar unmöglich umzusetzen zu sein. Nachdem ich jedoch alle möglichen Probleme wie Abhängigkeiten und Kompilierfehler aus der Welt geschafft hatte, war mein eigenes Linux System gebaut und voll funktionsfähig. Ich nannte es Linux From Scratch System, 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 der Basis an zu erstellen, wenn ich einfach ein existierendes Linux herunterladen und installieren kann?". Das ist natürlich eine berechtigte Frage und gleichzeitig auch der Anstoß für dieses Kapitel des Buches.

Ein wichtiger Grund für die Existenz von LFS ist, dem Leser beizubringen wie Linux von innen heraus funktioniert. Ein Linux System selber zu bauen demonstriert, was Linux seinen Herzschlag verleiht, wie die Dinge zusammenarbeiten und voneinander abhängen. Das beste ist, das man durch den Lernprozess versteht, wie man Linux an seine eigenen Bedürfnisse anpassen kann.

Einer der grössten Vorteile von LFS ist, das man mehr Kontrolle über sein System erhält 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 Verzeichnis 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 man eine übliche Linux Distribution installiert ist man für gewöhnlich gezwungen viele Programme zu installieren die man höchstwahrscheinlich niemals benutzen wird. Diese liegen dann unnütz auf der Festplatte und verbrauchen Speicherplatz (oder noch schlimmer, CPU Ressourcen). Es ist leicht ein LFS System unter 100Mb 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 8Mb belegtem Festplattenspeicher. Durch weitere Beschneidungen könnte das System auf bis zu 5Mb oder weniger schrumpfen. Versuchen sie das mit einer herkömmlichen Linux Distribution.

Man könnte die verschiedenen Linux Distributionen mit einem Hamburger vergleichen den man in einer Fast Food Kette kauft −− man weis nie genau was man ißt. LFS auf der anderen Seite wäre kein Hamburger, sondern vielmehr das Rezept wie man einen Hamburger macht. Das ermöglicht einem, 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 ist 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 die gesamten Quelltexte sichten und alle Sicherheitspatches installieren die man für wichtig hält. Man muss nicht darauf warten das jemand anders Binärpakete zum beheben von Sicherheitslöchern bereitstellt. Solange man Patches nicht selber überprüft und installiert gibt es keine Garantie das das Binärpaket korrekt kompiliert wurde und das 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. Während sie mit LFS arbeiteten 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 dieses Buches sehr ähnlich ist, aber es konzentriert sich ausschliesslich auf das erstellen eines Systems 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, das 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 Software Pakete 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 speziell geschrieben für neue Linux Anwender. Es ist hauptsächlich eine Linksammlung zu sehr guten Informationsquellen zu allen möglichen Themen. Jeder der LFS installieren möchte sollte zumindest die meisten der dort behandelten Themen verstehen. Es ist verfügbar unter http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

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, Changelog, 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 des LFS: kompilieren und installieren aller Pakete Schritt für Schritt, aufsetzen der Bootskripte und installieren des Kernel. 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.

Teil IV − Anhänge

Teil IV enthält zwei Anhänge. Der erste ist eine alphabetische Liste alle Pakete die installiert werden. Für jedes Paket ist die URL zum herunterladen, der Inhalt und die Installationsabhängigkeiten angegeben. Der zweite Anhang listet in alphabetischer Reihenfolge alle Programme und Bibliotheken auf die im Rahmen dieses Buches installiert wurden. So können sie einfach herausfinden, welche Programme oder Bibliotheken

zu welchem Paket gehören.

(Vieles aus dem ersten Anhang ist bereits in Teil II und III enthalten. Das dehnt das Buch natürlich ein wenig aus, aber wir glauben das es so einfacher zu lesen ist. So brauchen sie während der Installation nicht immer im Anhang nachschlagen. Das ständige vor und zurück wäre lästig, vor allem wenn sie eine reine Textversion des Buches lesen sollten.)

I. Teil I − Einführung

Inhaltsverzeichnis

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 Entwicklung bei den zu installierenden Programmen ausgewählt haben.

In Kapitel 3 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 Kapitel 4 alle für LFS notwendigen Pakete und Patche herunter und speichern sie auf dem neuen Dateisystem.

Kapitel 5 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 Kapitel 6 zu erstellen. Einige dieser Pakete werden benötigt um rekursive Abhängigkeiten aufzulösen −− zum Beispiel brauchen sie einen Compiler um einen Compiler zu kompilieren.

Als erstes im Kapitel 5 erstellen sie eine erste Version der Basiswerkzeuge, bestehend aus Binutils und GCC. Die Programme aus diesen Paketen werden statisch verlinkt damit sie unabhängig vom Hostsystem 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 wir eine zweite Version der Basiswerkzeuge. Dieses Mal verlinken wir die Programme dynamisch gegen die gerade frisch installierte Glibc. 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, das dies eine ganze Menge Arbeit ist, nur um von der Host Distribution unabhängig zu werden . Nun, eine vollständige Erklärung befindet sich am Anfang von Kapitel 5, inklusive einiger Hinweise auf die Unterschiede zwischen statisch und dynamisch verlinkter Programme.

In Kapitel 6 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 gesetzt ist. Das ist ähnlich wie neustarten und mounten der LFS Partition als root Partition. Der Grund warum sie nicht wirklich neustarten sondern stattdessen chroot'en ist, weil das erstellen eines bootfähigen Systems zusätzliche Arbeit erfordert die im Moment noch unnötig ist. Der grosse Vorteil gegenüber dem Neustart ist, das chroot'en des Systems erlaubt es, das Host Betriebssystem weiter zu benutzen 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 normal weiter arbeiten wie gewohnt.

Zum Abschluss der Installation werden im Kapitel 7 die Boot Skripte eingerichtet, der Kernel und der Bootloader werden in Kapitel 8 konfiguriert, und Kapitel 9 enthält Verweise 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 sie abarbeiten. Machen sie sich keine Gedanken falls jetzt noch etwas unklar sein sollte, alle Fragen werden im weiteren Verlauf beantwortet werden.

Bitte lesen Sie Kapitel 2 sehr genau, es erklärt einige sehr wichtige Dinge über die sie sich im klaren sein sollten bevor sie mit Kapitel 5 und den folgenden beginnen.

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, Downloadseiten 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.

Version dieses Buches

Dies ist das Linux From Scratch Buch in der Version 5.0 vom 5. November 2003. Wenn dieses Buch älter als zwei Monate ist, gibt es vielleicht bereits eine neuere, bessere Version. Das können sie überprüfen indem sie einen unserer Spiegel aus der Liste von http://www.linuxfromscratch.org/ besuchen.

Änderungsprotokoll

5.0 − 5. November 2003

- Upgrade zu:
  - automake−1.7.6
  - bash−2.05b
  - binutils−2.14
  - e2fsprogs−1.34
  - file−4.04
  - findutils−4.1.20
  - gawk−3.1.3
  - gcc−3.3.1
  - gettext−0.12.1
  - glibc−2.3.2
  - glibc−2.3.2−sscanf−1.patch
  - grep−2.5.1
  - groff−1.19
  - gzip−1.3.5
  - less−381
  - lfs−bootscripts−1.12
  - libtool−1.5
  - linux−2.4.22
  - man−1.5m2
  - man−1.5m2−80cols.patch
  - man−1.5m2−manpath.patch
  - man−1.5m2−pager.patch
  - man−pages−1.60
  - modutils−2.4.25
  - procps−3.1.11
  - procps−3.1.11.patch
  - psmisc−21.3
  - sed−4.0.7
  - sysvinit−2.85
  - tar−1.13.25
  - texinfo−4.6
  - util−linux−2.12
  - vim−6.2
- Hinzugefügt:
  - bash−2.05b−2.patch
  - bison−1.875−attribute.patch
  - coreutils−5.0
  - coreutils−5.0−uname.patch
  - coreutils−5.0−hostname−2.patch
  - dejagnu−1.4.3
  - expect−5.39.0
  - expect−5.39.0.patch

    - gawk−3.1.3.patch
    - gcc−2.95.3
    - gcc−2.95.3−2.patch
    - gcc−2.95.3−no−fixinc.patch
    - gcc−2.95.3−returntype−fix.patch
    - gcc−3.3.1−no_fixincludes−2.patch
    - gcc−3.3.1−specs−2.patch
    - gcc−3.3.1−suppress−libiberty.patch
    - grub−0.93
    - grub−0.93−gcc33−1.patch
    - inetutils−1.4.2
    - lfs−utils−0.3
    - ncurses−5.3−etip−2.patch
    - ncurses−5.3−vsscanf.patch
    - perl−5.8.0−libc−3.patch
    - shadow−4.0.3−newgroup−fix.patch
    - tcl−8.4.4
    - zlib−1.1.4−vsnprintf.patch
- Entfernt:

    - bin86−0.16.3
    - fileutils−4.1
    - fileutils−4.1.patch
    - findutils−4.1−segfault.patch
    - findutils−4.1.patch
    - glibc−2.3.1−libnss.patch
    - glibc−2.3.1−root−perl.patch
    - gzip−1.2.4b.patch
    - lilo−22.2
    - netkit−base−0.17
    - sh−utils−2.0
    - sh−utils−2.0.patch
    - sh−utils−2.0−hostname.patch
    - tar−1.13.patch
    - textutils−2.1
    - vim−6.1.patch
- November 2nd, 2003 [alex]: Appendix A − Commented out all the "last checked against" lines.
- October 28th, 2003 [greg]: Strengthened the seds in "Locking in Glibc" and "Re−adjusting the toolchain" sections.
- October 26th, 2003 [greg]: Chapter 6 − Glibc: Added command to create /etc/ld.so.conf to match Chapter 5 Glibc. Closes bug 700.
- October 24th, 2003 [alex]: Appendix A − Changed the dependencies to the concise format, based on Tushar's post .
- October 23rd, 2003 [gerard] Chapter 9 − The End: Changed the `/etc/lfs` filename to `/etc/lfs−release` to be more consistent with other distributions out there.
- October 23rd, 2003 [alex]: Changed most of the "Chapter" references to proper "xref" cross references .
- October 22nd, 2003 [alex]: Chapter 6 − Gawk and Shadow: Adjusted the text. And added some markup elsewhere.
- October 22nd, 2003 [alex]: Chapter 6 − Entering the chroot environment: Dropped the **`set +h`** command, as it is pointless there: it's redone several sections later.

- October 15th, 2003 [greg]: Chapter 9: Reworked final strip command. Relocated paragraphs about directory removal from Chapter 6.
- October 14th, 2003 [greg]: Chapter 8 − Making the LFS system bootable: Expanded Grub details and added a warning.
- October 14th, 2003 [alex]: Appendix A − Updated the contents of Perl and Procps.
- October 14th, 2003 [alex]: Chapter 4 and 5 − Added a suggestion to use $LFS/sources as the working and storage place.
- October 13th, 2003 [greg]: Chapter 9 − Rebooting the system: Reworked umount commands.
- October 11th, 2003 [alex]: Adapted the required disk space values and SBUs, as posted by Bruce Dubbs.
- October 11th, 2003 [alex]: Chapter 5 − Toolchain technical notes: Added and changed some markup.
- October 9th, 2003 [gerard]: Upgraded to lfs−bootscripts−1.12.
- October 9th, 2003 [greg]: Performed internal markup reworking to fix an extraneous whitespace problem in "tidy generated" web site pages. Essentially replace all occurrences of <para><screen> with <screen> (and the matching closing tags).
- October 9th, 2003 [alex]: Chapter 6 − Basic Networking: Moved one half to the Lfs−Utils section, the other half to Perl.
- October 8th, 2003 [alex]: Chapter 8 − Making bootable: Adapted the style of the screens, and reworded some paragraphs.
- October 8th, 2003 [alex]: Removed a series of unused entities.
- October 7th, 2003 [jeremy]: Added notes to the linking tests in chapter 5 and 6 stating that blank output is a bad thing.
- October 7th, 2003 [alex]: Changed the patch entities to contain the full filename instead of just the version number.
- October 7th, 2003 [jeremy]: Chapter 1 − Added a note regarding #LFS−support on IRC.
- October 7th, 2003 [greg]: Preface: Add note about the Essential Pre−Reading Hint. Closes Bug 585.
- October 6th, 2003 [alex]: Changed the style of the Contents subsections in Chapters 5 and 6 and Appendix A.
- October 6th, 2003 [greg]: Simplified seds in "Locking in Glibc" and "Re−adjusting the toolchain" sections. Rearranged "How things are going to be done" section.
- October 5th, 2003 [greg]: Chapter 5: Added new section "Toolchain technical notes". Integrated and scaled back the old "Why we use static linking" section. Closes Bug 658.
- October 4th, 2003 [alex]: Minor rewordings and additions of markup here and there.
- October 4th, 2003 [greg]: Chapter 5 − Binutils Pass 1: Added extra LDFLAGS to ensure static rebuild of ld.
- October 2nd, 2003 [greg]: Chapter 6: Reinstated INSTALL=/tools/bin/install for linker adjustment command due to issues on hosts where a ginstall symlink exists. This renders the "install" symlinks redundant, so removed those too.
- October 2nd, 2003 [greg]: Chapter 6 − Shadow: Enabled MD5 passwords. Closes Bug 600.
- September 27th, 2003 [greg]: Chapter 5 − Expect: Tweaked install so that redundant scripts are not installed. Chapter 6 − Creating essential symlinks: Removed redundant links. Chapter 6 − man: Removed PATH, closes Bug 574.
- September 27th, 2003 [greg]: Added Tcl, Expect and DejaGnu items to Appendix A. Closes Bug 661.
- September 26th, 2003 [jeremy]: Added new workaround for the devpts problems.
- September 24th, 2003 [greg]: Various changes across the board addressing Bug 675.
- September 24th, 2003 [alex]: Appendix A − Changed the style of the short descriptions, and the content of most of them too.
- September 22nd, 2003 [greg]: Chapter 8 − Creating the /etc/fstab file: Made mounting devpts the default.
- September 22nd, 2003 [jeremy]: Added Net−tools patch to fix mii−tool compilation.

- September 22nd, 2003 [jwrober]: Chapter 5 − Updated the Why Static page to more accurately represent the difference between statically and dynamically linked binaries. Thanks to Ian Molton for pointing this out. Fixes Bug 602.
- September 22nd, 2003 [jeremy]: Removed the make command from DejaGnu, since it performs nothing.
- September 22nd, 2003 [jeremy]: Removed the −k from Tcl's make check, since it's not expected to have failures anymore
- September 22nd, 2003 [jeremy]: Changed the reference to the man hint to a pointer to BLFS.
- September 22nd, 2003 [jeremy]: Added a note to remember to mount devpts if you exit and re−enter chroot.
- September 22nd, 2003 [jeremy]: Removed make check from Patch and Diffutils, since these tests perform no actions.
- September 22nd, 2003 [greg]: Chapter 5 − Setting up the environment: Added unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD to .bash_profile to stop accidental build breakage.
- September 20th, 2003 [greg]: Chapter 5 − GCC Pass 2: Updated to gcc−3.3.1−specs−2.patch. Ncurses: added −−enable−overwrite and description.
- September 19th, 2003 [jeremy]: Corrected bash tags for proper use of the +h flag to bash.
- September 19th, 2003 [jwrober]: Various updates to the acknowledgments page.
- September 18th, 2003 [jeremy]: Chapter 5 − GCC Pass 2: Added some extra comments regarding the 3 tarballs to unpack.
- September 17th, 2003 [greg]: Chapter 6 − GCC−2.95.3: Added rationale notes.
- September 17th, 2003 [jwrober]: Updated the acknowledgments page to match the website.
- September 17th, 2003 [jeremy]: Upgraded File to 4.04.
- September 17th, 2003 [jeremy]: Chapter 6 − Changed 2 of the occurrences of exec bash −−login to include the +h directive.
- September 17th, 2003 [greg]: Chapters 5 and 6 − Locking in Glibc and Re−adjusting the toolchain: Do "make −C ld install" instead of "make −C ld install−data−local" to install a whole new linker instead of just the new ldscripts.
- September 17th, 2003 [alex]: Normalized the spelling of 'Tcl' and 'DejaGnu', following their own documentation.
- September 17th, 2003 [alex]: Properly alphabetized the dependencies.
- September 16th, 2003 [alex]: Finally updated the dependencies for the new Coreutils.
- September 16th, 2003 [greg]: Chapters 5 and 6 − Locking in Glibc and Re−adjusting the toolchain: Added sanity checks.
- September 16th, 2003 [greg]: Chapters 5 and 6 − Binutils, GCC, and Glibc: Added notes on the test suites.
- September 15th, 2003 [alex]: Corrected several typos and some inconsistencies.
- September 14th, 2003 [greg]: Chapter 6 − Revised chroot command: Removed no longer needed set +h.
- September 14th, 2003 [alex]: Fixed some typos, and added some markup. Dropped the removal of program files from the Stripping section in Chapter 5.
- September 14th, 2003 [greg]: Chapter 6 − Create essential symlinks: Add symlink /usr/lib/libgcc_s.so.1 to allow GCC abi_check to run. Future NPTL needs this as well.
- September 13th, 2003 [jwrober]: Added PLFS hint text to the page in Chapter 6 for creating passwd and group: bug 596.
- September 13th, 2003 [jwrober]: Updated the "How things are going to be done" page to include more of the PLFS hint's text.
- September 13th, 2003 [jwrober]: Preface − Merged whoread and whonotread into a single audience page.
- September 13th, 2003 [greg]: Chapter 2 − Added new section about the test suites.

- September 12th, 2003 [jeremy]: Chapter 5 − Ncurses: Added description for the −−without−ada configure switch.
- September 12th, 2003 [jeremy]: Chapter 5 − Gawk: Added the test suite
- September 12th, 2003 [jeremy]: Chapter 5 − Grep: Added descriptions of configure switches courtesy of Anderson Lizardo
- September 12th, 2003 [gerard]: Removed /usr/lib/locale directory creation − it's created during Chapter 6 − Glibc where it's more relevant.
- September 11th, 2003 [jwrober]: Chapter 5 − Fixed GCC Pass 2 specs patch text to be more vague, but in actuality more accurate − provided by Anderson Lizardo.
- September 11th, 2003 [jwrober]: Chapter 5 − Grammar fix in Tcl install directions provided by Anderson Lizardo.
- September 11th, 2003 [jwrober]: Chapter 5 − Small textual change in the locking in Glibc page for /lib/ld.so.1 provided by Anderson Lizardo.
- September 11th, 2003 [jeremy]: Added bootloader setup to Chapter 8, after the addition of Grub to the book.
- September 11th, 2003 [gerard]: Removed Bin86 and LILO and replaced it with Grub.
- September 11th, 2003 [jeremy]: Dropped non−toolchain tests to optional actions. Added a note to use the Wiki for failed tests.
- September 11th, 2003 [jeremy]: Added Bison patch, backported from CVS, to fix pwlib compilation problems
- September 11th, 2003 [jeremy]: Added Greg's patch to GCC to suppress the installation of libiberty, and changed Binutils to allow its libiberty to stay.
- September 11th, 2003 [jeremy]: Added caution tags around the reminder to not delete the Binutils source and build directories in Chapter 5.
- September 11th, 2003 [jeremy]: Added new perl−libc−3 patch from Anderson Lizardo
- September 9th, 2003 [jwrober]: Fixed the Findutils package download link on the packages page closing bug 578.
- September 9th, 2003 [jeremy]: Chapter 6 − GCC 2.95.3: Removed compilation of C++, added Zack's return−type patch.
- September 9th, 2003 [jeremy]: Chapter 6 − Coreutils: Added coreutils−5.0−hostname−2.patch, which suppresses the build of the hostname binary, and also suppresses its check.
- September 9th, 2003 [jeremy]: Added some notes regarding failed tests to Glibc and DejaGnu.
- September 9th, 2003 [jeremy]: Glibc − Added commands to both Chapter 5 and 6 to include minimum locales necessary for checks.
- September 9th, 2003 [jeremy]: Chapter 6 − Removed Zlib's munging of CFLAGS in favor of a note to add −fPIC.
- September 8th, 2003 [matt]: Chapter 5 − Fixed the rm command that deletes unneeded documentation from /tools/share.
- September 6th, 2003 [matt]: Chapter 6 − Removed a reference to "the static" directory in the intro.
- September 6th, 2003 [jeremy]: Chapter 4 − Updated download locations for some packages.
- September 5th, 2003 [jeremy]: Chapter 5 − GCC Pass 2: Corrected the make check error explanation
- September 5th, 2003 [jeremy]: Chapter 6 − Makedev: Changed the default device creation to generic−nopty, because we now use devpts by default.
- September 5th, 2003 [jeremy]: Chapter 6 − GCC: Corrected wording to reflect the removal of the /usr/lib/cpp symlink.
- September 5th, 2003 [jeremy]: Corrected perl libc patch to −2, changing the old /stage1 structure to /tools
- September 5th, 2003 [matt]: Chapter 6 − Updated GCC specs patch and upgraded to man−1.5m2
- September 4th, 2003 [jeremy]: Chapter 6 − Creating Directories: Eliminated the creation of /usr/tmp − Closes bug 176.

- September 4th, 2003 [jeremy]: Chapter 6 − Mounting Proc: Added mounting the devpts filesystem into chroot here. Closes bug 533.
- September 4th, 2003 [jeremy]: Chapter 6 − Mounting Proc: Added a warning at the end regarding checking that proc is still mounted if you stop and restart the lfs process.
- September 4th, 2003 [jeremy]: Chapter 6 − Gzip: Altered text to better explain the reason behind the sed command used in the gzip installation. Closes bug 551.
- September 4th, 2003 [jeremy]: Chapter 4 − Downloading patches: Added a note regarding Tushar's patches project, and a link to the patches home page.
- September 3rd, 2003 [matt]: Fixed issue with Util−linux not utilizing headers and libraries installed in /stage1.
- September 3rd, 2003 [matt]: Removed "rm /bin/pwd" instruction from Chapter 6 kernel−headers installation as the link is still required by Glibc's installation.
- September 2nd, 2003 [alex]: Adjusted all the SBUs from the values posted by Jeremy.
- September 2nd, 2003 [alex]: Finally got around to renaming /stage1 to /tools.
- September 2nd, 2003 [alex]: Merged several of the main book structure files.
- September 2nd, 2003 [alex]: Alphabetized download lists, added note to Tcl instructions.
- September 2nd, 2003 [alex]: Reworded Organization, $LFS and SBUs sections.
- September 1, 2003 [jeremy] − Chapter 6 − Groff − Added note about choice of A4 or letter for the PAGE variable.
- September 1, 2003 [jeremy] − Added in shadow newgrp patch from Greg Schafer
- August 31, 2003 [jeremy] − Chapter 6 − Inetutils − added the −−disable−whois and −−disable−servers flags
- August 31, 2003 [jeremy] − Added in Greg's new instructions for GCC 3.3.1 with respect to the fixincludes process. Also added extra verbiage to the Locking in Glibc and GCC Pass 2 pages on the fixincludes process.
- August 31st, 2003 [alex]: Reworded some paragraphs, added missing markup, and rearranged the changelog.
- August 31st, 2003 [alex]: Wrapped the 'Last checked' lines in parentheses. Several other small retouches.
- August 30, 2003 [jeremy] − Updated fix−includes patch to GCC 3.3.1
- August 29, 2003 [jeremy] − Glibc − updated instructions with the sscanf patch from patches.
- August 29, 2003 [jeremy] − Updated GCC to version 3.3.1, including fixes based on Zack's mini−hint for GCC 3.3, and patches from his docs.
- August 29th, 2003 [alex]: Removed obsolete Netkit−base, Fileutils, Sh−utils, and Textutils files.
- August 29th, 2003 [alex]: Added some missing markup, changed a few /static's to /stage1's.
- August 29th, 2003 [alex]: Chapter 06 − Added all the missing text lines before the make checks, and reworded other lines.
- August 28, 2003 [matt] − Updated packages to linux−2.4.22, man−pages−1.60, expect−5.39.0, findutils−4.1.20 and tcl−8.4.4
- August 28, 2003 [jeremy] − New bash−2.05b−2.patch file to include the 7 patches from ftp.gnu.org
- August 28th, 2003 [alex]: Chapter 06 − Re−adjusting toolchain: Added a forgotten backslash.
- August 28th, 2003 [alex]: Fixed a few typos and added some missing markup.
- August 28th, 2003 [alex]: Chapter 06 − Binutils and GCC: Integrated text from the pure−lfs hint.
- August 27, 2003 [jeremy] − Chapter 06 − Inetutils: Added −−sysconfdir=/etc −−localstatedir=/var and moved the ping binary from /usr/bin to /bin
- August 27th, 2003 [alex]: Chapter 06 − Glibc: Integrated text from the pure−lfs hint.
- August 26, 2003 [jeremy] − Chapter 07 − Creating /etc/hosts: Changed www.mydomain.org to <value of HOSTNAME>.mydomain.org
- August 26th, 2003 [alex]: Chapter 06 & 08 − Moved the installation of the kernel manpages from chapter 6 to 8.

- August 26, 2003 [jeremy] − Chapter 04 − Mounting the LFS partition: Added text regarding mounting with too restrictive permissions.
- August 26, 2003 [jeremy] − Chapter 06 − Creating Directories: Added the creation of the /dev/shm directory.
- August 26, 2003 [jeremy] − Chapter 08 − Creating fstab: Added the mount of tmpfs filesystem to /dev/shm.
- August 26, 2003 [jeremy] − Chapter 08 − Kernel Installation: Added a reminder to compile tmpfs support into the kernel.
- August 25th, 2003 [alex]: Chapter 06 − Rewrote the installation text of Shadow and Util−Linux while correcting some typos.
- August 25th, 2003 [alex]: Chapter 05 & 06 − Made the "Locking in" and "Re−adjusting" look similar.
- August 24th, 2003 [alex]: Chapter 04 − Merged the many little files into one file. Gave packages and patches a separate page.
- August 17th, 2003 [alex]: Chapter 05 − From Bash to Perl: put text in between commands. Added a section on stripping unneeded symbols to decrease the size of the tools.
- August 16th, 2003 [alex]: Chapter 05 − From Make to Texinfo: put text in between commands.
- August 11th, 2003 [alex]: Chapter 05 − From Binutils Pass 1 to Findutils: several small textual adjustments. For the second passes not giving the contents and dependencies.
- August 11th, 2003 [alex]: Chapter 04 − Listed separate core, g++, and test suite tarballs for GCC.
- August 11th, 2003 [alex]: Chapter 04 − Suppressed the mention of a wget script.
- August 9th, 2003 [alex]: Chapter 05 − Binutils Pass 2 and GCC Pass 2: integrated some text from the pure−lfs hint.
- August 8th, 2003 [alex]: Chapter 05 − Tcl, Expect, and DejaGnu: added some text.
- August 6th, 2003 [gerard]: Applied Alex Groenewoud's patch that adds Appendix B, providing a list of all installed programs and libraries plus references to the installation pages.
- July 30th, 2003 [gerard]: Chapter 06 − Vim: Changed the way the global `vimrc` and `gvimrc` locations are defined.
- July 30th, 2003 [gerard]: Chapter 05 − Binutils Pass 2: removed the lib patch, it's no longer needed with the binutils−2.14 upgrade.
- July 30th, 2003 [gerard]: Chapter 05 Binutils Pass 1: Added **`make configure-host`**.
- July 30th, 2003 [gerard]: Upgraded to binutils−2.14, linux−2.4.21, expect−5.38.4, gawk−3.1.3, texinfo−4.6, util−linux−2.12, man−pages−1.58, lfs−utils−0.3, vim−6.2, gettext−0.12.1, automake−1.7.6, file−4.03, e2fsprogs−1.34, procps−3.1.11, psmisc−21.3
- June 3rd, 2003 [gerard]: Chapter 06 − Gawk: removed the removal of `/bin/awk`. This symlink isn't created anymore.
- May 21st, 2003 [gerard]: Chapter 06 − GCC−2.95.3: Added /opt/gcc−2.95.3/lib to the /etc/ld.so.conf file so the libraries can be found during run−time.
- May 21st, 2003 [gerard]: Chapter 05 − Gzip: Simplified commands.
- May 21st, 2003 [gerard]: Chapter 05 − Bzip2: Simplified commands.
- May 21st, 2003 [gerard]: Chapter 06 − Shadow: Added the **`grpconv`** command to complement the enabling of all shadowed passwords.
- May 21st, 2003 [winkie]: Chapter 06 − Creating Files: All those **`ln`** commands can be made into a few long ln commands.
- May 21st, 2003 [winkie]: Chapter 05 − Installing Glibc: Create an ld.so.conf file before building Glibc, to prevent an (harmless) error.
- May 21st, 2003 [winkie]: Chapter 06 − Installing Glibc: Don't bother doing the 'exec /stage1/bin/bash' stuff, it doesn't do anything now that we use PLFS.
- May 21st, 2003 [winkie]: Chapter 05 & 06 − Installing Coreutils: Only test the non−root stuff in Chapter 05, but test everything in Chapter 06.
- May 21st, 2003 [winkie]: Chapter 05 − Installing Expect: Don't bother passing anything more than −−prefix=/stage1. None of it is needed.

- May 16th, 2003 [gerard]: Chapter 06: Net−tools: Changed **`make install`** to **`make update`**.
- May 15th, 2003 [timothy]: Chapter 05: Installing Patch: Added **`CPPFLAGS=-D_GNU_SOURCE`** before **`./configure`** to fix patch build on PPC.
- May 13th, 2003 [gerard]: Chapter 06: When we **`exec /path/to/bash --login`**, also run **`set +h`** to keep the hashing option turned off. Fixes bug #531
- May 13th, 2003 [gerard]: Chapter 06 − Basic Network: Changed the single quotes to double quotes in the echo command. Without it, $(hostname) won't expand which defeats the sole purpose of this command − to make Perl's hostname check work.
- May 13th, 2003 [winkie]: Removed all occurrences &&. Updated bug syntax. Added "make check/test" where necessary in Chapter 6.
- May 13th, 2003 [winkie]: Chapter 06: Applied "Changing ownership" patch to polish the text. Closes bug #511.
- May 13th, 2003 [winkie]: Chapter 06: Applied "Configuring system components" patch to polish the text. Closes bug #510.
- May 13th, 2003 [gerard]: Chapter 06: Removed Tcl, Expect and DejaGnu. Nothing uses this once past GCC in chapter 6. The versions in /stage1/bin do the job just fine.
- May 13th, 2003 [winkie]: Chapter 06 − Installing Shadow: Touching the /usr/bin/passwd file before installation. Not doing so results in Shadow thinking passwd will be in /bin/passwd.
- May 13th, 2003 [winkie]: Chapter 06 − Installing Procps: Remove the /lib/libproc.so symlink. No package outside of Procps itself uses this library, and none should.
- May 13th, 2003 [winkie]: Chapter 06 − Installing Net−tools: Run "make config" before doing make. Fixes bugs #462 and #497.
- May 13th, 2003 [gerard]: Chapter 06 − Ncurses: Added the vsscanf patch.
- May 12th, 2003 [gerard]: Chapter 05 − Gzip: Removed **`make check`**. It doesn't do anything.
- May 12th, 2003 [winkie]: Chapter 05 − Installing Texinfo: Don't install the texmf data. It's not used by anything.
- May 12th, 2003 [winkie]: Chapter 05 & 06 − Installing Ncurses: In Chapter 6, symlink creation has been updated to include libcurses.*, and libncurses++.a has its properties changed to 644. Chapter 5 doesn't need any libcurses.* so those are removed.
- May 12th, 2003 [gerard]: Chapter 06 − Basic Network: Added $(hostname) to /etc/hosts, without it Perl's hostname test doesn't pass.
- May 12th, 2003 [gerard]: Chapter 06 − Installing GCC: Don't try to remove /usr/include/libiberty.h. It isn't installed in the first place.
- May 12th, 2003 [winkie]: Upgraded to findutils−4.1.7, gzip−1.3.5, and tar−1.13.25.
- May 12th, 2003 [winkie]: Chapter 05 − Installing Perl: Added extra commands to build certain modules into Perl. This is to accommodate the Coreutils "make check". Partially fixes bug #528.
- May 12th, 2003 [winkie]: Chapter 05 − Installing Gzip: Nothing in Chapter 6 checks for or uses the uncompress command, therefore we shouldn't create it.
- May 12th, 2003 [winkie]: Chapter 05 − Installing Bzip2: Running "make" implies "make check", therefore there is no reason whatsoever for us to run it manually.
- May 12th, 2003 [winkie]: Chapter 05 − Installing Lfs−Utils: Removed. The only package that checks for mktemp before it is installed is GCC, and that's only for gccbug.
- May 11th, 2003 [gerard]: Chapter 06 − GCC−2.95.3: Added −−enable−threads=posix as well to complete the C++ addition.
- May 11th, 2003 [gerard]: Chapter 06 − GCC−2.95.3: Added −−enable−languages=c,c++ to fix that GCC's version bug with regards to −Wreturn−type. Fixes bug #525
- May 11th, 2003 [gerard]: Chapter 05 − Bash: Removed the −−without−bash−malloc configure option.
- May 11th, 2003 [gerard]: Updated to gcc−3.2.3−specs−4.patch.
- May 11th, 2003 [winkie]: Chapter 06 − Setting up Basic Networking: Added section. Create a basic /etc/hosts files, and create /etc/services and /etc/protocols from IANA. Fixes bugs #359 & #515.

- May 11th, 2003 [winkie]: Upgrading to lfs−utils−0.2.2. This adds two files needed for proper networking configuration.
- May 11th, 2003 [winkie]: Removed Netkit−base 0.17. Added Inetutils 1.4.2. Fixes bug #490.
- May 11th, 2003 [winkie]: Added lfs−utils−0.2.1. Fixes bug #493.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Ncurses: Fix up the symlinks so that they follow suit of other library symlinks. No more weirdness here.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Procps: Removed XSCPT="" cruft and its corresponding paragraph. This stuff isn't needed anymore.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Ncurses: Pass −−without−debug to the configure script. It seems to have gotten lost at some point.
- May 11th, 2003 [timothy]: Chapter 5 & 6 − Installing Bzip2, Installing Zlib: Modified build commands per bug #524.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Glibc: Install the linuxthreads man pages, too. This got lost somewhere.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Grep: Added −−with−included−regex to prevent Grep from using Glibc's somewhat bugged regex.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Coreutils: Fix some functionality of the uname command with a patch.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Net−tools: Just do regular old "make install" instead of "make update". The latter works fine now.
- May 11th, 2003 [winkie]: Chapter 06 − Installing GCC: After installation, remove /usr/include/libiberty.h. It is not used outside of the GCC build tree.
- May 11th, 2003 [winkie]: Upgraded to Bash 2.05b and added its patch.
- May 11th, 2003 [winkie]: Chapter 06 − Installing Zlib: Apply a patch to fix the buffer overflow in gzprintf().
- May 11th, 2003 [winkie]: Chapter 06 − Configuring system components: Moved the creation of the btmp, wtmp, lastlog and utmp to just before Shadow, so that they are detected at their proper locations.
- May 10th, 2003 [winkie]: Chapter 06 − Installing Automake: Run "make" before installing. This is needed now with the newer releases of Automake.
- May 10th, 2003 [winkie]: Chapter 06 − Installing Vim: Removed the patch. It hasn't been required since GCC 3.2.1.
- May 10th, 2003 [winkie]: Chapter 06 − Creating the mtab file: Removed. Mounting /proc has the side effect of creating /etc/mtab for us.
- May 10th, 2003 [winkie]: Chapter 06 − Installing Make: Removed modification of /usr/bin/make file. It is no longer mistakenly installed with strange ownership or permissions.
- May 10th, 2003 [winkie]: Chapter 06 − Installing Glibc: Made /etc/localtime a file instead of a symlink. The symlink method breaks on systems where /usr is a separate partition.
- May 10th, 2003 [winkie]: Chapter 06 − Installing E2fsprogs: Removed install−info commands for e2fsprogs. The "make install" target handles this for us.
- May 10th, 2003 [gerard]: Removed all CFLAGS and LDFLAGS variables where they are not essential (so, not including static binutils, GCC and compiling Zlib with −fPIC).
- May 10th, 2003 [gerard]: Chapter 05 − Binutils (pass1, pass2), locking in Glibc and adjusting toolchain: Changed tooldir to /stage1 (likewise we use tooldir=/usr in Chapter 6).
- May 10th, 2003 [gerard]: Chapter 05 − Kernel headers: Removed the usage of **`cp -H`** because there are distributions out there that do not know about the **`-H`** option.
- May 10th, 2003 [gerard]: New gcc−3.2.3−specs−3.patch.
- May 10th, 2003 [gerard]: Chapter 06 − Adjusting toolchain: Made it more architecture−independent.
- May 10th, 2003 [gerard]: Chapter 05 − Locking in Glibc: Made it more architecture−independent.
- May 7th, 2003 [gerard]: Removed GCC No Debug patches. No longer assume gcc−core and gcc−g++ packages are downloaded, so added appropriate −−enable−languages options.

- May 7th, 2003 [gerard]: Removed Chapter 6 − Glibc−Pass2. It's not needed anymore with the pure−lfs integration.
- May 7th, 2003 [gerard]: Downgraded to flex−2.5.4a again. Newer versions just don't work properly.
- May 5th, 2003 [gerard]: Removed zlib installation from chapter 5 (its inclusion was a mistake).
- May 5th, 2003 [gerard]: Various bug fixes that were introduced during the pure−lfs integration.
- May 2nd, 2003 [gerard]: Upgraded to: automake−1.7.4, e2fsprogs−1.33, file−4.02, flex−2.5.31, gawk−3.1.2, gcc−3.2.3, glibc−2.3.2, grep−2.5.1, groff−1.19, less−381, libtool−1.5, man−1.5l, man−pages−1.56, modutils−2.4.25, procps−3.1.8, sed−4.0.7, sysvinit−2.85, texinfo−4.5, util−linux−2.11z
- May 2nd, 2003 [gerard]: Removed fileutils−4.1, sh−utils−2.0, textutils−2.1 (all replaced with coreutils−5.0).
- May 2nd, 2003 [gerard]: Added binutils−2.13.2−libc.patch, coreutils−5.0, dejagnu−1.4.3, expect−5.38, gawk−3.1.2, gcc−2.95.3, tcl−8.4.2
- May 2nd, 2003 [gerard] − Integrated new installation method from the Pure LFS hint written by Greg Schafer and Ryan Oliver.

Erscheinen der Version 4.1 am 28. April 2003.

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 Emails an die Mailinglisten werden in die dazugehörige Newsgruppe kopiert und umgekehrt.

Der News Server ist erreichbar unter *news.linuxfromscratch.org*.

Software Spiegel

Das LFS Projekt hat viele Software Spiegel 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 Software Spiegel einzusehen.

Kontakt

Bitte senden sie alle Fragen und Kommentare direkt zu einer der LFS Mailinglisten (siehe oben).

Aber wenn sie Gerard Beekmans persönlich erreichen müssen, senden sie eine Email an gerard@linuxfromscratch.org.

Danksagungen

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

Aktuelle Mitglieder des Projekt Teams

- Gerard Beekmans <gerard@linuxfromscratch.org> −− Linux−From−Scratch Initiator, LFS Projektbetreuer.
- Matthew Burgess <matthew@linuxfromscratch.org> −− LFS allgemeiner Pakete−Betreuer, LFS Buch Autor.
- Craig Colton <meerkats@bellsouth.net> −− LFS, ALFS, BLFS und Ersteller des Tipp−Projekt Logo.
- Jeroen Coumans <jeroen@linuxfromscratch.org> −− Website Entwickler, FAQ Betreuer.
- Bruce Dubbs <bdubbs@linuxfromscratch.org> −− LFS Kopf des Qualitätssicherungsteams, BLFS Buch Autor.
- Alex Groenewoud <alex@linuxfromscratch.org> −− LFS Buch Autor.
- Mark Hymers <markh@linuxfromscratch.org> −− CVS Betreuer, Ersteller des BLFS Buch, früherer LFS Buch Autor.
- 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.
- Scot Mc Pherson <scot@linuxfromscratch.org> −− LFS NNTP Gateway Betreuer.
- Ryan Oliver <ryan@linuxfromscratch.org> −− Kopf des Test Teams, Miterschaffer des PLFS.
- James Robertson <jwrober@linuxfromscratch.org> −− Bugzilla Betreuer, Wiki Entwickler, LFS Buch Autor.
- Greg Schafer <greg@linuxfromscratch.org> −− Toolchain Betreuer, LFS Buch Autor, Miterschaffer des PLFS.
- Tushar Teredesai <tushar@linuxfromscratch.org> −− BLFS Buch Autor, Betreuer des Tipps und Patches Projekts.
- Jeremy Utley <jeremy@linuxfromscratch.org> −− LFS Buch Autor, Bugzilla Betreuer.
- 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.

Softwarespiegel Betreuer

- Jason Andrade <jason@dstc.edu.au> −− au.linuxfromscratch.org Spiegel.
- William Astle <lost@l−w.net> −− ca.linuxfromscratch.org Spiegel.
- Baque <baque@cict.fr> −− lfs.cict.fr Spiegel.
- Stephan Brendel <stevie@stevie20.de> −− lfs.netservice−neuss.de Spiegel.
- Ian Chilton <ian@ichilton.co.uk> −− us.linuxfromscratch.org, linuxfromscratch.co.uk Spiegel.
- Fredrik Danerklint <fredan−lfs@fredan.org> −− se.linuxfromscratch.org Spiegel.
- David D.W. Downey <pgpkeys@aeternamtech.com> −− lfs.learnbyexample.com Spiegel.
- Eduardo B. Fonseca <ebf@aedsolucoes.com.br> −− br.linuxfromscratch.org Spiegel.
- Hagen Herrschaft <hrx@hrxnet.de> −− de.linuxfromscratch.org Spiegel.
- Tim Jackson <tim@idge.net> −− linuxfromscratch.idge.net Spiegel.
- Barna Koczka <barna@siker.hu> −− hu.linuxfromscratch.org Spiegel.
- Roel Neefs −− linuxfromscratch.rave.org Spiegel.
- Simon Nicoll <sime@dot−sime.com> −− uk.linuxfromscratch.org Spiegel.
- Ervin S. Odisho <ervin@activalink.net> −− lfs.activalink.net Spiegel.
- Guido Passet <guido@primerelay.net> −− nl.linuxfromscratch.org Spiegel.
- Mikhail Pastukhov <miha@xuy.biz> −− lfs.130th.net Spiegel.
- Jeremy Polen <jpolen@rackspace.com> −− us2.linuxfromscratch.org Spiegel.
- UK Mirror Service −− linuxfromscratch.mirror.co.uk Spiegel.
- Thomas Skyt <thomas@sofagang.dk> −− dk.linuxfromscratch.org Spiegel.
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> −− at.linuxfromscratch.org Spiegel.
- Dag Stenstad <dag@stenstad.net> für die Bereitstellung von no.linuxfromscratch.org und Ian Chilton für das Betreiben.
- Parisian sysadmins <archive@doc.cs.univ−paris8.fr> −− www2.fr.linuxfromscratch.org Spiegel.
- Jesse Tie−Ten−Quee <highos@linuxfromscratch.org> für das Bereitstellen und Betreiben des linuxfromscratch.org Server.
- Alexander Velin <velin@zadnik.org> −− bg.linuxfromscratch.org Spiegel.
- Martin Voss <Martin.Voss@ada.de> −− lfs.linux−matrix.net Spiegel.
- Pui Yong <pyng@spam.averse.net> −− sg.linuxfromscratch.org Spiegel.

Spender

- Dean Benson <dean@vipersoft.co.uk> für etliche Geldspenden.
- DREAMWVR.COM für das ehemalige Sponsoring von Ressourcen zu LFS und dazugehörigen Projekten.
- 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 (früher 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.

Ehemalige Team−Mitglieder und Beitragende

- Timothy Bauscher <timothy@linuxfromscratch.org> −− LFS Buch Autor, Tipps Projekt Betreuer.
- Robert Briggs für die Spende der *linuxfromscratch.org* und *linuxfromscratch.com* Domain Namen.
- Ian Chilton <ian@ichilton.co.uk> für die Betreuung des Tipps−Projekt.
- Marc Heerdink <gimli@linuxfromscratch.org> −− LFS Buch Autor.
- Seth W. Klein <sklein@linuxfromscratch.org> −− Erschaffer der LFS FAQ.
- Garrett LeSage <garrett@linuxart.com> −− Erschaffer des ursprünglichen LFS Banners.
- Simon Perreault <nomis80@videotron.ca> −− Betreuer des Tipps Projekt.
- Geert Poels <Geert.Poels@skynet.be> −− Erschaffer 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 Beantworten zahlloser Fragen im IRC und für seine unendliche Geduld.

Kapitel 2. Wichtige Informationen

Über $LFS

Bitte lesen sie den folgenden Abschnitt sorgfältig. In diesem Buch begegnen sie häufig der Variable LFS. $LFS ist ein Platzhalter, sie müssen ihn durch den Verzeichnisnamen ersetzen in dem sie die LFS Partition eingehängt (gemountet) haben. Wie man eine Partition erstellt und wo man sie einhängt wird im Detail in Kapitel 3 erklärt. Im Moment gehen wir davon aus, das sie die LFS Partition unter `/mnt/lfs` eingehängt haben.

Wenn ihnen gesagt wird sie sollen ein Kommando wie z. B. dieses ausführen: **`./configure --prefix=$LFS/tools`**, dann müssen sie in wirklichkeit **`./configure --prefix=/mnt/lfs/tools`** eingeben.

Es ist sehr wichtig das sie das tun, egal wo sie es lesen; sei es ein Shell Kommando oder eine Datei die sie bearbeiten oder erstellen sollen.

Eine komfortable Lösung zu diesem Umstand wäre, die Umgebungsvariable LFS zu setzen. Dann können sie $LFS einfach so eingeben wie sie es lesen und müssen diesen Platzhalter nicht durch den echten Pfad ersetzen. Das setzen der Umgebungsvariable können sie so erreichen:

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

Immer wenn sie nun ein Kommando wie **`./configure --prefix=$LFS/tools`** eingeben sollen, können sie es tatsächlich so abtippen wie sie es lesen. Ihre Shell wird "$LFS" durch "/mnt/lfs" ersetzen während sie ihre Eingabe verarbeitet (sprich: wenn sie nach Eingabe des Befehls die Enter Taste drücken).

Über SBUs

Die meisten Leute möchten gerne vorher wissen wie lange es ungefähr dauert um die einzelnen Pakete zu kompilieren und installieren. "Linux From Scratch" wird aber auf so unterschiedlichen Systemen gebaut, das es unmöglich ist, echte Zeiten anzugeben die auch nur annähern 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 folgendermaßen: Das erste Paket das sie kompilieren werden ist das statisch gelinkte Binutils Paket in Kapitel 5. Die Zeit die sie benötigen um dieses Paket zu kompilieren ist das was wir eine "Static Binutils Unit" oder auch "SBU" nennen. Alle anderen Kompilierzeiten werden relativ zu dieser Zeit angegeben.

Um zum Beispiel die statische Version von GCC zu bauen braucht es 4.4 SBUs. Das bedeutet wenn es 10 Minuten gedauert hat um die statischen Binutils zu bauen, dann wissen sie das es ungefähr 45 Minuten Zeit braucht 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, währen der Rest der Pakete aber mit der neuen Version gebaut wird. GCC−3.3.1 ist dafür bekannt, ca. 30% langsamer zu sein.

Bitte beachten sie auch, das 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 sie sich nicht darum kümmern.

Über die Test−suites

Die meisten Pakete stellen eine Test−suite zur Verfügung. Es ist prinzipiell immer eine gute Idee, eine solche Test−suite für neu kompilierte Programme auch durchlaufen zu lassen, so stellen sie sicher, das alles korrekt kompiliert wurde. Wenn eine Test−suite alle ihre Tests erfolgreich durchläuft können sie ziemlich sicher sein das das Paket funktioniert wie es der Entwickler vorgesehen hat. Nichtsdestotrotz garantiert das natürlich nicht für fehlerfreiheit.

Ein paar Tests sind wichtiger als andere. Zum Beispiel die Tests der toolchain Pakete −− GCC, Binutils und Glibc (die C Bibliothek) −− sind von höchster Wichtigkeit weil diese Pakete eine absolut zentrale Rolle für die Funktion des gesamten Systems spielen. Aber seien sie gewarnt: die Test−suites von GCC und Glibc benötigen sehr viel Zeit, vor allem auf langsamer Hardware.

Wir werden ihnen im laufe der Arbeit mit diesem Buch sagen, wie wichtig welche Test−suite jeweils ist, sie können dann selbst entscheiden ob sie die Suite durchlaufen lassen möchten oder nicht.

> **Anmerkung:** Ein weit verbreitetes Problem beim durchlaufen der Test−suites 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, das das *devpts* Dateisystem des Host−System nicht korrekt aufgesetzt ist. Wir werden das später in Kapitel 5 ausführlicher behandeln.

Wie sie nach Hilfe fragen können

Wenn sie beim lesen des Buches ein Problem entdecken das nicht von der FAQ (http://www.linuxfromscratch.org/faq) abgedeckt wird, dann sind die meisten Leute im Internet Relay Chat (IRC) und auf den Mailinglisten gern bereit ihnen zu helfen. Eine Übersicht über die LFS Mailinglisten finden sie unter Kapitel 1 − Mailinglisten. Um uns bei der Hilfe zu unterstützen sollten sie uns soviel relevante Informationen wie möglich geben.

Dinge die sie angeben sollten

Neben einer kurzen Erklärung des Problems ist es wichtig das sie uns noch folgende Dinge mitteilen:

- Die Version des Buches das sie benutzen (dies ist die Version 5.0),
- die Host−Distribution und Versionsnummer die sie benutzen um LFS zu installieren,
- das Paket oder die Sektion die ihnen Probleme macht,
- die exakte Fehlermeldung oder 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 das wir ihnen nicht helfen werden. Der Grundsatz von LFS ist es, die Wahl zu haben. Diese Angabe hilft uns lediglich mögliche Ursachen für ihre 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`. Die Datei enthält viele Fehlermeldungen die auf dem Bildschirm sonst nicht angezeigt werden. Geben sie diese Fehlermeldungen mit an wenn sie nach Hilfe fragen.

Kompilierprobleme

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, aber auf der anderen Seite sollte es 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
```

Oft kopieren viele Leute 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 das *etwas* schiefgelaufen ist, aber nicht *was*. Der ganze Abschnitt, wie oben gezeigt, sollte angegeben werden, denn er enthält das ausgeführte Kommando und die dazugehörige Fehlermeldung.

Einen sehr guten Beitrag zu diesem Thema hat Eric S. Raymond geschrieben. sie können ihn lesen unter http://catb.org/~esr/faqs/smart−questions.html. Lesen und befolgen sie bitte seine Tipps in dem Dokument, das erhöht die Chance das sie eine Antwort auf ihre Frage erhalten mit der sie auch etwas anfangen können.

Probleme mit Test−suites

Viele Pakete enthalten eine Test−suite. Abhängig von der Wichtigkeit eines Paketes empfehlen wir ihnen die Test−suite durchlaufen zu lassen. Manchmal erzeugen die Pakete Fehlmeldungen oder unerwartete Ergebnisse. Falls sie solchen Problemen begegnen können sie auf den LFS Wiki Seiten 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.

II. Teil II − Vorbereitungen zur Installation

Inhaltsverzeichnis

Kapitel 3. 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 Patition und keinen unpartitionierten Platz auf ihrer Festplatte haben dann können sie LFS auch auf der selben Partition installieren auf der ihre gerade installierte Distribution bereits 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 1.2 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 benutzt 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, sie müssen also nicht noch eine weitere Partition erstellen wenn sie bereits eine funktionsfähige Swap Partition haben.

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 neuen Dateisystems auf der Partition

Nun, wo wir eine leere Partition haben, können wir 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 Dateisystem immer beliebter. An dieser Stelle werden wir ein ext2 Dateisystem erstellen, Anleitungen für andere Dateisysteme können sie aber 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 (etwas wie zum Beispiel hda5).

Wenn sie eine (neue) Swap Partition erstellt haben, müssen sie diese initialisieren (wird auch als formatieren bezeichnet). Das machen sie ähnlich wie oben beschrieben mit dem Kommando:

```
mkswap /dev/yyy
```

Bitte ersetzen sie yyy mit dem echten Namen der Swap Partition.

Einhängen (mounten) der neuen Partition

Nun, nachdem wir ein Dateisystem erzeugt haben, möchten wir auch auf dieses Zugreifen können. Dazu müssen wir es erst einhängen (mounten), ausserdem müssen wir einen Mountpunkt auswählen. In diesem Buch nehmen wir an, das das Dateisystem unter /mnt/lfs eingehängt wird, aber im Grunde ist es egal, welches Verzeichnis sie sich aussuchen.

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

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

Nun erstellen sie den Mountpunkt und mounten sie das LFS Dateisystem mit diesen Kommandos:

```
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 wir 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 mit den korrekten Bezeichnungen ersetzen.

Sie sollten sicherstellen, das die neue Partition nicht mit zu einschränkenden Rechten eingehängt wird (wie zum Beispiel mit den "nosuid", "nodev" oder "noatime" Optionen). Rufen sie **mount** ohne Parameter auf um zu sehen mit welchen Optionen ihre Dateisystem 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.

Kapitel 4. Das Material: Pakete und Patche

Einführung

Die Liste unten enthält alle Pakete die sie für ein minimales Linux System herunterladen müssen. Die angegebenen Versionsnummern entsprechen Softwareversionen bei denen wir *wissen* das sie funktionieren, und das Buch basiert darauf. Solange sie wenig Erfahrung mit LFS haben empfehlen wir dringend, keine neueren Versionen zu probieren. Die angegebenen Kommandos könnten 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 URL auf die Projektseite unter http://www.freshmeat.net/. Die Freshmeat Seiten erlauben einfachen Zugriff auf die offiziellen Download− und Projektseiten, Mailinglisten, FAQ's, Changelogs und noch mehr.

Wir können nicht garantieren das 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://linuxfromscratch.org/lfs/print/, oder noch besser, sie probieren eine alternative Download−Methode aus, beschrieben auf der Seite http://linuxfromscratch.org/lfs/packages.html.

Sie müssen alle heruntergeladenen Pakete und Patche an einem Ort speichern auf den sie während des gesamten Buch einfachen Zugriff haben. Weiterhin brauchen sie ein Arbeitsverzeichnis in dem sie die Quellen entpacken und kompilieren können. Eine gute Vorgehensweise ist, das Verzeichnis `$LFS/sources` zum speichern der Quellen und Patche *und* als Arbeitsverzeichnis zu benutzen. So ist alles was sie benötigen immer auf der LFS Partition abgelegt und in allen Arbeitsschritten des Buches verfügbar.

Wie empfehlen ihnen daher folgendes Kommand als *root* auszuführen bevor sie mit dem herunterladen der Pakete beginnen:

```
mkdir $LFS/sources
```

Machen sie dieses Verzeichnis für normale Benutzer beschreibbar (und sticky) −− denn sie werden das downloaden 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 (2.57) − 792 KB:
http://freshmeat.net/projects/autoconf/

Automake (1.7.6) − 545 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 (5.0) − 3,860 KB:
http://freshmeat.net/projects/coreutils/

DejaGnu (1.4.3) − 1,775 KB:
http://freshmeat.net/projects/dejagnu/

Diffutils (2.8.1) − 762 KB:
http://freshmeat.net/projects/diffutils/

E2fsprogs (1.34) − 3,003 KB:
http://freshmeat.net/projects/e2fsprogs/

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

Expect (5.39.0) − 508 KB:
http://freshmeat.net/projects/expect/

File (4.04) − 338 KB: (*) See Note Below
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 (3.3.1) − 10,969 KB:
http://freshmeat.net/projects/gcc/

GCC−g++ (3.3.1) − 2,017 KB:
http://freshmeat.net/projects/gcc/

GCC−testsuite (3.3.1) − 1,033 KB:

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

Gettext (0.12.1) − 5,593 KB:
http://freshmeat.net/projects/gettext/

Glibc (2.3.2) − 13,064 KB:
http://freshmeat.net/projects/glibc/

Glibc−linuxthreads (2.3.2) − 211 KB:
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 (0.93) − 870 KB:
ftp://alpha.gnu.org/pub/gnu/grub/

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

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

Kbd (1.08) − 801 KB:
http://freshmeat.net/projects/kbd/

Less (381) − 259 KB:
http://freshmeat.net/projects/less/

LFS−Bootscripts (1.12) − 25 KB:
http://downloads.linuxfromscratch.org/lfs−bootscripts−1.12.tar.bz2

Lfs−Utils (0.3) − 221 KB:
http://www.linuxfromscratch.org/~winkie/downloads/lfs−utils/

Libtool (1.5) − 2,751 KB:
http://freshmeat.net/projects/libtool/

Linux (2.4.22) − 28,837 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

MAKEDEV (1.7) − 8 KB:

http://downloads.linuxfromscratch.org/MAKEDEV−1.7.bz2

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

Man−pages (1.60) − 627 KB:
http://freshmeat.net/projects/man−pages/

Modutils (2.4.25) − 215 KB:
http://freshmeat.net/projects/modutils/

Ncurses (5.3) − 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 (5.8.0) − 10,765 KB:
http://freshmeat.net/projects/perl/

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

Procps (3.1.11) − 242 KB:
http://freshmeat.net/projects/procps/

Psmisc (21.3) − 259 KB:
http://freshmeat.net/projects/psmisc/

Sed (4.0.7) − 678 KB:
http://freshmeat.net/projects/sed/

Shadow (4.0.3) − 760 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 (1.13.25) − 1,281 KB:
ftp://alpha.gnu.org/gnu/tar/

Tcl (8.4.4) − 3,292 KB:
http://freshmeat.net/projects/tcltk/

Texinfo (4.6) − 1,317 KB:

http://freshmeat.net/projects/texinfo/

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

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

Zlib (1.1.4) − 144 KB:
http://freshmeat.net/projects/zlib/

Gesamtgröße der Pakete: 134 MB

> **Anmerkung:** Wenn sie das hier lesen ist File (4.04) möglicherweise nicht in dieser Version verfügbar. Der Hauptdownloadserver ist dafür bekannt, alte Versionen zu löschen wenn neuere verfügbar sind. Bitte nutzen sie eine der alternativen download Adressen aus dem entsprechenden Abschnitt in Anhang A.

Benötigte Patche

Neben all den Paketen benötigen sie auch einige Patche. Dieses umgehen entweder kleine Fehler die von dem Maintainer noch behoben werden, oder sie machen kleine Modifikationen und Anpassungen an unser LFS. Sie brauchen folgende Patche:

Bash Patch − 7 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/bash−2.05b−2.patch

Bison Attribute Patch − 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/bison−1.875−attribute.patch

Coreutils Hostname Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−hostname−2.patch

Coreutils Uname Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−uname.patch

Ed Mkstemp Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/ed−0.2−mkstemp.patch

Expect Spawn Patch − 6 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/expect−5.39.0−spawn.patch

Gawk Libexecdir Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gawk−3.1.3−libexecdir.patch

GCC No−Fixincludes Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−no_fixincludes−2.patch

GCC Specs Patch − 10 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−specs−2.patch

GCC Suppress−Libiberty Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−suppress−libiberty.patch

GCC−2 Patch − 16 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−2.patch

GCC−2 No−Fixincludes Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−no−fixinc.patch

GCC−2 Return−Type Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−returntype−fix.patch

Glibc Sscanf Patch − 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/glibc−2.3.2−sscanf−1.patch

Grub Gcc33 Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/grub−0.93−gcc33−1.patch

Kbd More−Programs Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/kbd−1.08−more−programs.patch

Man 80−Columns Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−80cols.patch

Man Manpath Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−manpath.patch

Man Pager Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−pager.patch

Ncurses Etip Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−etip−2.patch

Ncurses Vsscanf Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−vsscanf.patch

Net−tools Mii−Tool−Gcc33 Patch − 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/net−tools−1.60−miitool−gcc33−1.patch

Perl Libc Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/perl−5.8.0−libc−3.patch

Procps Locale Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/procps−3.1.11−locale−fix.patch

Shadow Newgrp Patch − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/shadow−4.0.3−newgrp−fix.patch

Zlib Vsnprintf Patch − 10 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/zlib−1.1.4−vsnprintf.patch

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

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.

Das erstellen dieses minimalen Systems erfolgt in zwei Schritten: Als erstes erzeugen wir eine brandneue Host−unabhängige toolchain (Compiler, Assembler, Linker und Bibliotheken), und dann benutzen wir diese um alle weiteren essentiellen Werkzeuge zu kompilieren.

Die in diesem Kapitel kompilierten Dateien werden im Verzeichnis `$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.

Der Schlüssel um zum lernen wie Linux funktioniert ist, zu wissen wofür die installierte Software verwendet wird und warum der Benutzer oder das System sie benötigen. Aus diesem Grund steht vor den eigentlichen Installationsanweisungen jeweils eine kurze Zusammenfassung des Paketinhalts. Für eine kurze Beschreibung eines jeden Programmes schauen sie bitte in Anhang A.

Die Anweisungen zum kompilieren setzen voraus das sie die Bash Shell benutzen. Ausserdem wird grundsätzlich vorausgesetzt, das sie die Archive bereits entpackt haben und mit **`cd`** bereits in das jeweilige Quellverzeichnis 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.

Während der Installation der meisten Pakete werden sie alle möglichen Compiler Warnungen vorbeirollen sehen. Das ist normal und kann einfach ignoriert werden. Sie sind wirklich nur Warnungen −− meistens über missbilligte,(aber nicht ungültige) 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 können sie die Quell− und Kompilierverzeichnisse nach dem installieren des jeweiligen Paketes löschen −− zum Beispiel um aufzuräumen oder Platz zu sparen.

Bevor sie fortfahren stellen sie bitte mit folgenden Kommando sicher, das die LFS Umgebungsvariable korrekt gesetzt ist:

```
echo $LFS
```

Die Ausgabe sollte den Pfad zum Einhängepunkt 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 Bau− und Installationsprozess erläutern. Es ist nicht zwingend erforderlich das 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 Kapitel 5 ist es, eine gut funktionierende temporäre Arbeitsumgebung zu erschaffen in die wir uns später abkapseln und von wo aus wir in Kapitel 6 ohne Schwierigkeiten ein sauberes endgültiges LFS−System erzeugen können. Wir werden uns so weit wie möglich vom Host−System abschotten und so eine in sich geschlossene toolchain erzeugen. Bitte beachten sie das der gesamte Prozess ausgelegt ist um jegliches Risiken für neue Leser zu minimieren und gleichzeitig den Lerneffekt zu maximieren. Kurz gesagt könnte man 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−Triplet* bezeichnet. Für die meisten wird das Ziel−Triplet zum Beispiel *i686−pc−linux−gnu* sein. Ein einfacher Weg sein Ziel−Triplet herauszufinden ist, das `config.guess` Skript auszuführen welches 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* welcher Teil 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 zum ausführen 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 64bit Plattformen könnte er sogar völlig verschieden sein. Sie müssten den Namen ihres dynamischen Linkers herausfinden können wenn sie auf ihrem Host−System in das Verzeichnis `/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 Hauptverzeichnis des Glibc Quellverzeichnisses.

Ein paar technische Hinweise zum Kompilierprozess in Kapitel 5:

- Er ist im Grunde ähnlich wie Cross−Kompilieren wo Programme im selben Prefix in Kooperation zusammen funktionieren können und dazu ein wenig GNU "Magie" benutzen.
- Durch vorsichtiges anpassen des Standard Linker Suchpfades erreichen wir, das Programme nur gegen die gewünschten Bibliotheken gelinkt werden.
- Vorsichtiges anpassen von **`gcc`**'s *specs* Dateie um dem Compiler mitzuteilen welcher Dynamische Linker verwendet wird.

Binutils wird als erstes installiert weil sowohl GCC als auch Glibc beim durchlaufen des configure Skriptes einige Tests zum Assembler und Linker durchführen und auf dem Ergebnis basierend bestimmte Features ein− bzw. ausschalten. Das ist wichtiger als man anfangs 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 Test−suites im Regelfall auf solche Probleme hin bevor zuviel Zeit vergeudet wird.

Binutils installiert seinen Assembler an zwei Stellen, `/tools/bin` und `/tools/$TARGET_TRIPLET/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 erhält man 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 das alle Dateien beim Linken erfolgreich geöffnet werden konnten.

Das nächste zu installierende Paket ist GCC, und während dem Durchlauf 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, das 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** wirklich verwendet, kann man mittels **'gcc −print−prog−name=ld'** herausfinden. Dateillierte 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 Proprozessor, Komilierungs− und Assemblierungsphasen inklusive **gcc**'s Suchpfaden und der Reihenfolge aus.

Das nächste zu installierende Paket ist Gilbc. Die wichtigsten Überlegungen zum kompilieren von Glibc muss man zu Compiler, Binärtools und den Kernel Headern machen. Der Compiler ist normalerweise kein Proglem weil Glibc immer den **gcc** nimmt der in den $PATH Verzeichnissen gefunden wird. Die Binärtools und 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` Verzeichnis nach den Details durchsuchen. Sie werden ein paar interessante Dinge finden, wie zum Beispiel **CC="gcc −B/tools/bin/"** zum kontrollieren der verwendeten Binärtools, oder die Parameter *−nostdinc* und *−isystem* zum kontrollieren des Suchpfades des Compilers. Diese Besonderheiten heben einen wichtigen Aspekt des Glibc Paketes hervor: Es ist kompiliertechnisch 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 das Suchen und Verlinken nur innerhalb unseres `/tools` Prefix stattfindet. Wir installieren einen angepassten **ld**, welcher einen fest angegebenen Suchpfad auf `/tools/lib` hat. Dann bearbeiten wir **gcc**'s Spec Dateie so, das sie auf den neuen dynamischen Linker in `/tools/lib` zeigt. Der letzte Schritt ist *entscheidend* für den gesamten Prozess. Wie oben bereits angemerkt, wird ein fest eingestellter Pfad zum dynamischen Linker in jede ausführbare ELF Datei eingebettet. Sie können das überprüfen indem sie dieses Kommando ausführen: **'readelf −l <Name der ausführbaren Datei> | grep interpreter'**. Durch das anpassen von **gcc**'s Specs Datei stellen wir sicher, das jedes von hier an kompilierte Programm bis zum Ende von Kapitel 5 unseren neuen dynamischen Linker in `/tools/lib` benutzt.

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

Während dem zweiten Durchlauf von 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 Kapitel 5 kompilieren alle mit der neuen Glibc in `/tools` und alles ist in Ordnung.

Aufgrund ihrer bereits erwähnten eigenständigen Natur ist die Glibc das erste wichtige Paket das wir nach dem eintreten in die chroot Umgebung in Kapitel 6 installieren. Wenn die Glibc erstmal nach `/usr` installiert ist werden wir schnell ein paar Vorgaben in der toolchain ändern und dann schreiten wir in Kapitel 6 mit dem erstellen des endgültigen LFS Systems fort.

Bemerkungen zum statischen linken

Fast alle Programme führen neben ihrer eigentlichen Aufgabe noch einige weniger übliche, manchmal sehr triviale Dinge aus. Das beinhaltet das reservieren von Speicher, durchsuchen von Verzeichnissen, lesen und schreiben von Dateien, bearbeiten 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 Code der genutzten Funktionen in die ausführbare Datei eingebettet; das resultiert in einem relativ umfangreichen 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 ist 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 der ausführbaren Bibliothek anstatt viele Kopien in allen möglichen ausführbaren Dateien eingebettet zu haben -- nebenbei spart das auch Speicherplatz auf der Festplatte. Zweitens: Wenn viele Programme die gleichen Bibliotheksfunktionen gleichzeitig nutzen, 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 die eine Bibliothek neu kompilieren anstatt alle Programme neu zu kompilieren die die Bibliothek benutzen.

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 einen weiteren Punkt 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.

Erstellen des Verzeichnisses $LFS/tools

Alle in diesem Kapitel kompilierten Programme werden unter `$LFS/tools` installiert. Dadurch trennen wir sie von den Programmen die im nächsten Kapitel installiert werden. Die hier kompilierten Programme sind nur temporäre Hilfsmittel und werden kein Teil des endgültigen LFS Systems. Wenn wir diese Programme in

einem separaten Verzeichnis installieren, können sie später leichter gelöscht werden.

Falls sie später ihre ausführebaren Dateien des Systems durchsuchen möchten um zum Beispiel herauszufinden, welche Dateien sie benutzen oder wogegen sie verlinkt sind, dann können sie die Suche vereinfachen indem sie einen eindeutigen Namen vergeben. Statt dem einfachen "tools" können sie etwas wie "tools−fuer−lfs" benutzen.

Erstellen sie das Verzeichnis mit diesem Kommando:

```
mkdir $LFS/tools
```

Im nächsten Schritt erstellen sie auf ihrem Host−System einen symbolischen Link nach `/tools`. Er zeigt auf das Verzeichnis das wir gerade auf der LFS Partition erstellt haben:

```
ln −s $LFS/tools /
```

Dieser symbolische Link ermöglicht es uns, die toolchain so zu kompilieren das sie immer `/tools` referenziert; das bedeutet für uns das Compiler, Assember und Linker sowohl in diesem Kapitel (in dem wir immer noch einige Programme vom Host−System benutzen) *und* im nächsten Kapitel (wenn wir in die LFS Partition chrooted haben) funktionieren werden weil wir immer den gleichen gültigen Pfad haben.

> **Anmerkung:** Schauen sie sich das obige Kommando genau an. Es kann auf den ersten Blick verwirrend sein. Das `ln` Kommando hat verschiedene Syntax Variationen, also überprüfen sie erst die ln manpage bevor sie einen vermeintlichen Fehler berichten.

Erstellen des Benutzers lfs

Als *root* Benutzer eingeloggt können kleinste Fehler ihr System beschädigen oder gar zerstören. Deshalb empfehlen wir, das sie die Pakete in diesem Kapitel mithilfe 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 Installationsprozesses benutzen. Bitte führen sie als *root* diese Kommandos aus um den neuen Benutzer zu erzeugen:

```
useradd −s /bin/bash −m lfs
passwd lfs
```

Nun geben sie dem Benutzer *lfs* volle Zugriffsrechte auf `$LFS/tools` indem sie ihn zum Besitzer des Verzeichnisses machen:

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

Wenn sie wie vorgeschlagen ein extra Arbeitsverzeichnis 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, das **su** eine neue Shell startet.

Vorbereiten der Installationsumgebung

Um ihre Arbeitsumgebung für die weiteren Schritte vorzubereiten geben sie das folgende Kommando ein während sie als Benutzer *lfs* angemeldet sind:

```
cat > ~/.bash_profile << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
PATH=/tools/bin:$PATH
export LFS LC_ALL PATH
unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD
EOF

source ~/.bash_profile
```

Das Kommando **set +h** schaltet die Hash Funktion der Bash ab. Normalerweise ist das sog. hashing der Bash ein nützliches Feature: **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, das neu erzeugte Dateien nur durch ihren Besitzer beschreibbar sind, aber les− und ausführbar für jeden.

Die LFS Variable sollte natürlich auf den von ihnen gewählten Einhängepunkt der LFS Partition gesetzt sein.

Die Variable LC_ALL beeinflusst die lokalisierung einiger Programme, so das ihre 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. Wenn LC_ALL auf "POSIX" oder "C" (die beiden Werte haben die gleiche Wirkung) gesetzt ist sollte es beim hin− und herwechseln in der chroot Umgebung keine Probleme geben.

Wir setzen `/tools/bin` an den Anfang der PATH Variable, so das wir beim durcharbeiten dieses Kapitels die erzeugten Werkzeuge und Programme auch automatisch benutzen.

Die Umgebungsvariablen CC, CXX, CPP, LD_LIBRARY_PATH und LD_PRELOAD verursachen Probleme mit unserer toolchain in Kapitel 5. Daher setzen wir sie zurück um jegliche Schwierigkeiten zu vermeiden.

Nun, nach dem sog. sourcen (also einlesen) des gerade erstellen Profils, ist alles vorbereitet. Wir können mit dem bauen der Werkzeuge beginnen, diese werden uns dann in den weiteren Kapiteln von Nutzen sein.

Installieren von Binutils−2.14 − Durchlauf 1

```
Geschätzte Kompilierzeit:              1.0 SBU
Ungefähr benötigter Festplattenplatz:  194 MB
```

Inhalt von Binutils

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

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]

Binutils Installationsabhängigkeiten

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

Installieren von Binutils

Es ist wichtig, das Binutils als erstes Paket kompiliert wird, weil Glibc und GCC verschiedene Tests bezüglich Linker und Assembler durchführen und daraufhin erst diverse Features einschalten.

> **Anmerkung:** Auch wenn Binutils ein wichtiges Paket in der toolchain ist werden wir nicht die Test−suite durchlaufen lassen. Erstens ist die Test Umgebung noch nicht entsprechend vorbereitet und zweitens werden die nun kompilierten Programme sowieso von denen des zweiten Durchlaufes überschrieben.

Dieses Paket funktioniert nicht gut wenn nicht die Standard Optimierungseinstellungen (inklusive der −march und −mcpu Optionen) benutzt werden. Deshalb sollten event. gesetzte Umgebungsvariablen die die Standard Optimierung überschreiben − zum Beispiel CFLAGS und CXXFLAGS − für den Kompiliervorgang von Binutils zurückgesetzt oder entsprechend abgeändert werden.

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

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

> **Anmerkung:** Wenn sie die angegebenen SBU Werte im Buch benutzen möchten müssen sie nun die Zeit messen, die sie zum kompilieren dieses Pakets benötigen. Dies können sie relativ einfach auf folgende Art 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 dem Verzeichnis `/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 Unterverzeichnisse. Eine statisch gebaute Version würde ansonsten fehlschlagen.Wir benutzen diese Option um dieses Problem zu umgehen.
- **`LDFLAGS="-all-static"`**: Dies teilt dem Linker mit, das alle Binutils Programme statisch gelinkt werden sollen. Genaugenommen wird **`"-all-static"`** zuerst an *libtool* übergeben, welches dann wiederum **`"-static"`** an den Linker übergibt.

Und installieren sie das Paket:

```
make install
```

Bereiten sie nun den Linker auf das spätere hinzufügen von Glibc 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 make Programm an, alle kompilierten Dateien im Unterverzeichnis `ld` zu löschen.
- **`-C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib`**: Diese Option kompiliert alles im Unterverzeichnis `ld` erneut. Das angeben der LIB_PATH makefile Variable auf der Kommandozeile überschreibt den Standardwert und zeigt auf das temporäre tools Verzeichnis. Der Wert dieser Variable spezifiziert den Standard Bibliothek Suchpfad für den Linker. Sie werden später in diesem Kapitel sehen wie diese Vorbereitung angewendet wird.

Warnung
Entfernen sie die Binutils Kompilier− und Quellverzeichnisse noch nicht. Sie benötigen sie später in dem jetzigen Zustand.

Installieren von GCC−3.3.1 − Durchlauf 1

```
Geschätzte Kompilierzeit:              4.4 SBU
Ungefähr benötigter Festplattenplatz:  300 MB
```

Inhalt von GCC

Das Paket GCC enthält die Gnu Compiler Sammlung, inklusive dem C und C++ Compiler.

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

GCC Installationsabhängigkeiten

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

Installieren von GCC

Entpacken sie nur das GCC−core Tar−Archiv; wir werden erstmal keinen C++ Compiler brauchen.

> **Anmerkung:** Auch wenn GCC ein wichtiges toolchain Paket ist werden wir die Test−suite in diesem frühen Stadium nicht durchlaufen lassen. Erstens ist die Test Umgebung noch nicht entsprechend vorbereitet und zweitens werden die nun kompilierten Programme sowieso von denen des zweiten Durchlaufes überschrieben.

Es ist bekannt, das dieses Paket nicht richtig funktioniert wenn die standard Optimierungseinstellungen (inklusive der −march und −mcpu Optionen) verändert wurden. Deshalb sollten sie event. gesetzte Umgebungsvariablen die die Standard Optimierung überschreiben − zum Beispiel CFLAGS und CXXFLAGS − für den Kompiliervorgang von GCC zurücksetzen oder entsprechend abändern.

Die GCC Dokumentation empfiehlt, GCC nicht im Quellenverzeichnis sondern in einem dazu dedizierten Verzeichnis zu kompilieren:

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

Bereiten sie GCC zum kompilieren vor:

```
../gcc-3.3.1/configure --prefix=/tools \
    --with-local-prefix=/tools \
    --disable-nls --enable-shared \
    --enable-languages=c
```

Die Bedeutung der configure Optionen:

- **`--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 klug. Aber durch ihn kompilieren wir sowohl `libgcc_s.so.1` und `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 erziehlt. Beachten sie, das **`gcc`** selbst trotzdem statisch gelinkt wird, das wird durch den Wert **`-static`** zu BOOT_LDFLAGS im nächsten Schritt erreicht.

- **`--enable-languages=c`**: Diese Option stellt sicher, das 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 Programme die im ersten Durchlauf erzeugt werden um sich selbst im zweiten Durchlauf erneut zu kompilieren. Dann kompiliert sich GCC noch ein drittes mal. Am Schluss werden die Ergebnisse des zweiten und dritten Kompiliervorgangs verglichen um sicherzustellen das GCC sich selbst problemlos kompilieren konnte. Das bedeutet normalerweise das alles korrekt kompiliert wurde.

Installieren sie das Paket:

```
make install
```

Zum Abschluss erstellen wir noch den Symlink `/tools/bin/cc`. Viele Programme benutzen **`cc`** anstelle von **`gcc`**, das ist gedacht um Programme generisch zu halten und damit auf verschiedenen Unix System nutzbar zu machen. Nicht jeder hat den GNU C Compiler installiert. Einfach nur **`cc`** aufzurufen lässt dem Administrator die Wahl, welchen C Compiler er installieren möchte, solange es einen Symlink auf den echten Compiler gibt:

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

Installieren der Linux-2.4.22 Header

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  186 MB
```

Inhalt von Linux

Der Linux Kernel ist der Kern eines jeden Linux Systems. Er ist sozusagen der Herzschlag von Linux. Wenn der Computer eingeschaltet wird und ein Linux System startet, dann ist der Kernel das erste Stück Software das gestartet wird. Der Kernel initialisiert die Geräte und Hardware Komponenten: serielle Schnittstellen, parallele Schnittstellen, Soundkarten, Netzwerkkarten, IDE und SCSI Controller und vieles mehr. Zusammenfassend kann man sagen, der Kernel stellt dem System die Hardware zur Verfügung, so das die Software damit laufen kann.

Installierte Dateien: Der Kernel und die Kernel Header

Linux Installationsabhängigkeiten

Linux ist abhängig von: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Installation der Kernel Header

Da einige Pakete die Kernel Header Dateien referenzieren, entpacken wir nun das Kernel Archiv, 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
```

Das stellt sicher das der Kernel Baum absolut sauber ist. Das Kernel Team empfiehlt, dieses Kommando vor *jedem* Kernel kompilieren auszuführen. Sie sollten sich nicht darauf verlassen das die Quellen nach dem entpacken sauber sind.

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

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

Erstellen sie den Platform−spezifischen Symlink `include/asm`:

```
make symlinks
```

Installieren sie die Plattform−spezifischen Header Dateien:

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

Installieren sie die Multi−Plattform Header Dateien:

```
cp −R include/linux /tools/include
```

Es gibt einige wenige Header Dateien die `autoconf.h` benutzen. Da wir den Kernel jetzt noch nicht konfigurieren, müssen wir die Datei selbst erstellen um mögliche Kompilierfehler zu vermeiden. Erstellen sie die leere `autoconf.h` Datei:

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

Installieren von Glibc−2.3.2

```
Geschätzte Kompilierzeit:              11.8 SBU
Ungefähr benötigter Festplattenplatz:  800 MB
```

Inhalt von Glibc

Glibc ist die C Bibliothek, sie stellt Systemaufrufe und grundlegende Funktionen wie open, malloc, printf usw. zur Verfügung. Die C Bibliothek wird von allen dynamisch gelinkten Programmen verwendet.

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]

Glibc Installationsabhängigkeiten

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

Glibc Installation

Bevor sie mit der Installation von Glibc beginnen, müssen sie mit **cd** in das `glibc−2.3.2` Verzeichnis wechseln und dort Glibc−linuxthreads entpacken. Entpacken sie Glibc−linuxthreads nicht dort wo sie sonst die Archive entpacken würden, sondern in das Glibc Verzeichnis hinein.

> **Anmerkung:** In diesem Kapitel werden wir die Test−suite für Glibc durchlaufen lassen. Wir sollten allerdings erwähnen, das die Test−suite in diesem Kapitel nicht als kritisch eingestuft wird und nicht so wichtig ist wie in Kapitel 6.

Dieses Paket funktioniert nicht gut wenn nicht die Standard Optimierungseinstellungen (inklusive der −march und −mcpu Optionen) benutzt werden. Deshalb sollten event. gesetzte Umgebungsvariablen die die Standard Optimierung überschreiben − zum Beispiel CFLAGS und CXXFLAGS − für den Kompiliervorgang von Glibc zurückgesetzt oder entsprechend abgeändert werden.

Grundsätzlich kann man sagen, wenn sie von dem in diesem Buch beschriebenen Weg zum kompilieren von Glibc abweichen, riskieren sie die Stabilität ihres gesamten LFS Systems.

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
```

Zusätzlich hat die Glibc ein Problem, wenn sie mit GCC 3.3.1 kompiliert wird. Wenden sie den folgenden Patch an um das Problem zu beheben:

```
patch -Np1 -i ../glibc-2.3.2-sscanf-1.patch
```

Die Glibc Dokumentation empfiehlt, nicht im Quellenverzeichnis sondern in einem dafür dedizierten Verzeichnis zu kompilieren:

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

Als nächstes bereiten sie Glibc zum kompilieren vor:

```
../glibc-2.3.2/configure --prefix=/tools \
    --disable-profile --enable-add-ons \
    --with-headers=/tools/include \
    --with-binutils=/tools/bin \
    --without-gd
```

Die Bedeutung der configure Optionen:

- **`--disable-profile`**: Dies sorgt dafür, das Bibliotheken ohne profiling Informationen erzeugt werden. Lassen sie diese Option weg wenn sie mit den erzeugten Bibliotheken profiling betreiben möchten.
- **`--enable-add-ons`**: Dies aktiviert alle 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, das in Bezug auf die Kernel Header und Binutils Programme nichts schiefgehen kann.
- **`--without-gd`**: Diese Option stellt sicher das wir nicht das **`memusagestat`** Programm erzeugen, welches seltsamerweise immer gegen die Host−Bibliotheken (libgd, libpng, libz und so weiter) verlinkt wird.

Während dieser Phase bemerken 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 Test−suite führen.

Kompilieren sie das Paket:

```
make
```

Starten sie die Test−suite:

```
make check
```

Die Glibc Test−suite ist sehr stark von einigen Funktionen ihres Host−Systems abhängig, vor allem dem Kernel. Zusätzlich können in diesem Kapitel einige Tests von der Umgebung ihres Host−Systems beeinflusst werden. Diese werden natürlich kein Problem mehr sein, wenn wir später die Glibc Test−suite in der chroot

Umgebung in Kapitel 6 ausführen. Grundsätzlich erwarten wir, das die Glibc Test−suite 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 haben. Optimierungseinstellungen haben hier ebenfalls einen gewissen Einfluss.
- Der *gettext* Test schlägt manchmal aufgrund von Host−System bedingten Problemen fehl. Die genauen Gründe sind noch nicht ganz geklärt.
- Der *atime* Test schlägt fehl, wenn die LFS Partition mit der *noatime* Option gemountet wurde. Auch andere Dateisystemeigenheiten können hier Einfluss haben.
- Der *shm* Test könnte fehlschlagen wenn das Host−System das devfs Dateisystem laufen hat, aber aufgrund fehlender Kernel Unterstützung kein tmpfs Dateisystem unter `/dev/shm` gemountet ist.
- Auf alter oder langsamer Hardware können ein paar Tests aufgrund von Timeouts fehlschlagen.

Machen sie sich keine allzugrossen Gedanken wenn ein paar Glibc−Tests in diesem Kapitel fehlschlagen. Die Glibc aus Kapitel 6 ist diejenige die wir endgültig verwenden werden, erst dort ist es wirklich wichtig das die Tests erfolgreich durchlaufen. Aber denken sie daran, selbst in Kapitel 6 können immer noch Fehler auftreten −− beim *math* Test zum Beispiel. Wenn ein Fehler auftritt, notieren sie ihn, dann fahren sie mit **`make check`** fort. Die Test−suite sollte dann dort fortfahren wo sie aufgehört hat. Sie können dieses stoppen und starten umgehen indem sie **`make -k check`** aufrufen. Aber wenn sie das tun, stellen sie sicher das sie die Ausgaben mitloggen damit sie später die Logdatei nach den aufgetretenen Fehlern durchsuchen können.

Nun installieren sie das Paket:

```
make install
```

Verschiedene Länder und Kulturen haben auch unterschiedliche Konventionen zum kommunizieren. Diese Konventionen reichen von einfachen, wie zum Beispiel dem Format für Datum und Uhrzeit, bis hin zu sehr komplexen Konventionen, wie zum Beispiel der gesprochenen Sprache. Die "internationalisierung" von GNU Programmen funktioniert mithilfe der sogenannten *locales*. Wir installieren nun die Glibc locales:

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

Als Alternative zu dem vorigen Kommando können sie auch nur die locales installieren die sie brauchen oder wollen. Das erreichen sie mit dem **`localedef`** Kommando. Informationen dazu finden sie in der Datei `INSTALL` in den Quellen zu `glibc-2.3.2`. 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 install−locales target oben installieren einen minimalen Satz an locales die notwendig sind um die nachfolgenden Tests erfolgreich abschliessen 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 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
```

Die Glibc "integrieren"

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

Installieren sie zuerst den angepassten Linker in dem sie folgendes Kommando innerhalb des `binutils-build` Verzeichnisses ausführen:

```
make -C ld install
```

Den Linker haben wir erst kürzlich angepasst, nämlich am Ende des ersten Durchlaufs der Binutils. Ab diesem Punkt wird alles *nur* gegen die Bibliotheken in `/tools/lib` verlinkt.

> **Anmerkung:** Falls sie die vorige Warnung, das sie die Binutils Verzeichnisse nicht löschen sollen, übersehen haben sollten oder sie vielleicht versehentlich gelöscht haben, seien sie nicht besorgt. Es ist noch nicht alles verloren. Ignorieren sie das obige Kommando einfach. Das Ergebnis ist ein gewisses Risiko das 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 wo der angepasste Linker installiert ist müssen sie die Binutils Verzeichnisse löschen.

Als nächstes müssen sie die GCC Spec Datei ergänzen so das sie den neuen dynamischen Linker referenziert. Ein einfaches sed Kommando erledigt dies:

```
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 mittels kopieren und einfügen auszuführen anstelle es abzutippen. 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 an 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 namens *Technische Anmerkungen zur toolchain* nach.

Letztlich könnten möglicherweise einige include Dateien vom Host-System mit in das private include Verzeichnis des GCC geraten sein. Soetwas kann durch GCC's "fixincludes" Prozess geschehen der während dem kompilieren von GCC ausgeführt wird. Dazu werden wir später noch näheres erklären. Nun führen sie erstmal das folgende Kommando aus um dieses Problem zu umgehen:

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

Achtung
Es ist unbedingt notwendig an diesem Punkt die korrekte Funktion der toolchain (kompilieren und linken) zu

überprüfen. Darum führen wir nun einen kleinen "Gesundheitscheck" durch:

```
echo 'main(){}' > dummy.c
gcc 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]
```

Wenn sie nicht die obige Ausgabe oder überhaupt keine Ausgabe erhielten, 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. Am wahrscheinlichsten ist, das beim manipulieren der Specs Datei etwas nicht richtig funktioniert hat. Achten sie besonders darauf, das `/tools/lib` der Prefix zu ihrem dynamischen Linker ist. Wenn sie an einer Plattform arbeiten wo der Name des Linkers nicht `ld-linux.so.2` ist, könnte die Ausgabe natürlich leicht abweichen.

Wenn sie mit dem Ergebnis zufrieden sind löschen sie die Test Dateien:

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

Damit ist die Installation der eigenständigen, in sich geschlossenen toolchain abgeschlossen, sie kann nun zum erstellen der restlichen temporären Hilfsmittel verwendet werden.

Installieren von Tcl-8.4.4

```
Geschätzte Kompilierzeit:              0.9 SBU
Ungefähr benötigter Festplattenplatz:  23 MB
```

Inhalt von Tcl

Das Tcl Paket enthält die sog. Tool Command Language.

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

Installierte Bibliothek: libtcl8.4.so

Tcl Installationsabhängigkeiten

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 Test-suites 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, das unsere allerwichtigsten Programme und Werkzeuge richtig

funktionieren.

Bereiten sie Tcl zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu überprüfen das alles korrekt kompiliert wurde. Es ist jedoch bekannt, das die Tcl Test−suite unter bestimmten Bedingungen fehlschlägt. Daher sind Fehler in der Test−suite nicht überraschend, wir betrachten diese Fehler nicht als kritisch. Wenn sie die Test−suite dennoch ausführen möchten, erledigt das folgende Kommando dies für sie:

```
TZ=UTC make test
```

Die bedeutung des make Parameters:

- **TZ=UTC**: Setzt die Zeitzone für die dauer des Test−suite Durchlauf 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 Kapitel 7.

Manchmal erzeugt eine Test−suite falschen Alarm. Schlagen sie im LFS Wiki unter http://wiki.linuxfromscratch.org/ nach um zu überprüfen das mögliche Fehler normal sind. Das gilt für sämtliche Test−suiten im gesamten Buch.

Installieren sie das Paket:

```
make install
```

> **Wichtig:** Sie sollten das `tcl8.4.4` Quellverzeichnis *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
```

Installieren von Expect−5.39.0

```
Geschätzte Kompilierzeit:             0.1 SBU
Ungefähr benötigter Festplattenplatz:  3.9 MB
```

Inhalt von Expect

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

Installierte Programme: expect

Installierte Bibliotheken: libexpect5.39.a

Expect Installationsabhängigkeiten

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

Installieren von Expect

Wenden sie erst einen Patch an:

```
patch -Np1 -i ../expect-5.39.0-spawn.patch
```

Dies behebt einen Fehler in Expect der ansonsten Fehlalarme beim durchlaufen der GCC Test-suite verursachen könnte.

Bereiten sie nun Expect zum kompilieren vor:

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

Die Bedeutung der configure Optionen:

- **`--with-tcl=/tools/lib`**: So stellen wir sicher das das configure Skript die Tcl-Installation in unserem temporären Verzeichnis findet. Es sollte keine möglicherweise auf dem Host-System installierte Version gefunden werden.
- **`--with-x=no`**: Dies teilt dem configure Skript mit das 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
```

Dieses Paket hat eine Test-suite die sicherstellt das das Paket korrekt gebaut wurde. Es ist jedoch bekannt das diese Test-suite hier im Kapitel 5 Probleme macht die noch nicht ganz nachvollzogen wurden. Es ist daher nicht überraschend, wenn die Test-suite Fehler meldet, diese werden jedoch nicht als kritisch betrachtet. Sollten sie sich entscheiden, die Test-suite dennoch laufen zu lassen, dann benutzen sie dieses Kommando:

```
make test
```

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 Quellverzeichnisse von Tcl und Expect entfernen.

Installieren von DejaGnu−1.4.3

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  8.6 MB
```

Inhalt von DejaGnu

Das Paket DejaGnu enthält ein Grundgerüst zum testen anderer Programme.

Installiertes Programm: runtest

DejaGnu Installationsabhängigkeiten

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
```

Installieren von GCC−3.3.1 − Durchlauf 2

```
Geschätzte Kompilierzeit:              11.0 SBU
Ungefähr benötigter Festplattenplatz:  274 MB
```

Neuinstallation von GCC

Die Hilfmittel 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. Eine Sache die es noch zu beachten gibt: Die Test−suites sind stark von funktionierenden Pseudo−Terminals (PTYs) abhängig. Diese werden von dem Host−System bereitgestellt. Heutzutage werden PTYs meist über das *devpts* Dateisystem 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 Test−suites 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 LSF Wiki unter http://wiki.linuxfromscratch.org/.

Entpacken sie alle drei GCC Tar−Archive (−core, −g++ und −testsuite) in ein und demselben Arbeitsverzeichnis. Die Archive entpacken sich in ein einziges `gcc−3.3.1/` Unterverzeichnis.

Als erstes korrigieren sie ein Problem und machen eine wichtige Anpassung:

```
patch −Np1 −i ../gcc−3.3.1−no_fixincludes−2.patch
patch −Np1 −i ../gcc−3.3.1−specs−2.patch
```

Der este Patch schaltet das GCC "fixincludes" Skript ab. Wir haben das vorher schonmal kurz erwähnt, hier wollen wir eine tiefere Erklärung dazu geben. Unter normalen Umständen durchsucht das GCC fixincludes Skript ihr System nach Header Dateien die repariert werden müssen. Es könnte allerdings der Meinung sein das einige Header Dateien auf ihrem Host−System repariert werden müssen, repariert diese und kopiert sie in das private GCC Include Verzeichnis. Später dann in Kapitel 6, nachdem wir die neuere Glibc installiert haben, würde dieses private Include Verzeichnis vor den System Include Verzeichnissesn durchsucht werden. GCC würde dann die reparierten Include Dateien von ihrem Host−System finden, und diese passen dann nicht zu der Glibc Version die wir für das LFS System verwendet haben.

Der letzte Patch ändert den GCC standard Pfad des dynamischen Linkers (üblicherweise `ld−linux.so.2`). Ausserdem entfernt er `/usr/include` aus dem GCC Include Suchpfad. Durch das jetzige Patchen anstelle des nachträglichen anpassens der specs Datei stellen wir sicher das unser neuer dynamischer Linker während dem Bau von GCC verwendet wird. Das bedeutet, alle endgültigen (und auch temporären) Binärdateien während dem Kompilierdurchlauf werden gegen die neue Glibc gelinkt.

> **Wichtig:** Diese Patche sind *Voraussetzung* für einen erfolgreichen Gesamtdurchlauf. Vergessen sie nicht sie zu installieren.

Erstellen sie erneut ein dediziertes Verzeichnis 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−3.3.1/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−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 Standard−konforme 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-clocale=gnu**: Diese Option stellt sicher, das 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 von *gnu* wählen. Falls aber *de_DE* nicht installiert ist, besteht das Risiko aufgrund des fälschlicherweise ausgewählten Modells *generic* ABI−inkompatible C++ Bibliotheken zu erzeugen.
- **--enable-languages=c,c++**: Diese Option wird benötigt damit sowohl C, als auch 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.

> **Anmerkung:** Es sollte erwähnt werden, das die GCC Test−suite hier nicht so wichtig ist wie in Kapitel 6.

Testen sie die Ergebnisse:

```
make -k check
```

Der Schalter **-k** lässt die Test−suite bis zum Ende durchlaufen, auch wenn Fehler auftreten sollten. Die GCC Test−suite ist sehr umfangreich und es ist beinahe sicher das Fehler auftreten. Um eine Zusammenfassung der Test Ergebnisse zu erhalten benutzen sie dieses Kommando:

```
../gcc-3.3.1/contrib/test_summary | more
```

Sie können ihre Ergebnisse mit denen auf der gcc−testresults Mailingliste veröffentlichten vergleichen die eine ähnliche System−Konfiguration wie sie haben. Ein Beispiel wie GCC−3.3.1 auf i686−pc−linux−gnu aussehen sollte finden sie unter http://gcc.gnu.org/ml/gcc−testresults/2003−08/msg01612.html.

Beachten sie, das das Ergebnis folgendes enthält:

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

Der unerwartet erfolgreiche Durchlauf für g++ ist 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 26 unerwartet erfolgreichen Durchläufe für libstdc++ sind begründet durch die Option **--enable-clocale=gnu**, welches die korrekte Wahl auf Glibc basierten Systemen Version 2.2.5 oder höher ist. Die zugrundeliegende 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++ Test−suite erwartet anscheinend das "generic" Modell, daher wird nicht erwartet das dieses Tests erfolgreich durchlaufen.

Unerwartete Fehler lassen sich oftmals gar nicht vermeiden. Die GCC Entwickler kennen diese üblicherweise bereits, aber hatten noch keine Zeit diese Fehler zu beheben. Kurz gesagt, solange ihre Test Ergebnisse nicht grob von denen unter der obigen URL abweichen, ist es in Ordnung einfach fortzufahren.

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 Abschnitt namens *Die Glibc "integrieren"* nach und wiederholen sie die Prüfung. Wenn die Ergebnisse nicht in Ordnung sind haben sie höchstwahrscheinlich vergessen, den obig erwähnten GCC Specs Patch anzuwenden.

Installieren von Binutils−2.14 − Durchlauf 2

```
Geschätzte Kompilierzeit:              1.5 SBU
Ungefähr benötigter Festplattenplatz:  108 MB
```

Neuinstallation von Binutils

Erstellen sie erneut ein separates Verzeichnis 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 Bibliothek−Suchpfad vorzugeben. Wir möchten im Standard Bibliothek−Suchpfad keine Verzeichnisse unseres Host Systems haben, daher geben wir den gewünschten Pfad vor.

Bevor sie beginnen, Binutils zu kompilieren, denken sie daran alle Umgebungsvariablen zu entfernen, die die Standard Optimierungen übergehen würden.

Kompilieren sie das Paket:

```
make
```

Anmerkung: Wir sollten kurz erwähnen, das die Binutils Test−suite an diesem Punkt nicht so wichtig ist wie in Kapitel 6.

Testen sie das Ergebnis (es sollten keine unerwarteten Fehler auftreten, erwartete Fehler sind in Ordnung):

```
make check
```

Leider gibt es hier keine einfache Möglichkeit die Testergebnisse zusammenfassend anzuzeigen wie im vorigen GCC Paket. Nichtsdestotrotz, wenn ein Fehler auftritt sollte er leicht zu erkennen sein. Die Ausgabe zeigt dann soetwas wie:

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

Und installieren sie das Paket:

```
make install
```

Nun bereiten sie Binutils auf das erneute umkonfigurieren der toolchain im nächsten Kapitel vor:

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

Warnung
Entfernen sie die Binutils Quellen und Kompilierverzeichnisse jetzt noch nicht. Wir brauchen diese Verzeichnisse im jetzigen Zustand noch im nächsten Kapitel.

Installieren von Gawk−3.1.3

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  17 MB
```

Inhalt von Gawk

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

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

Gawk Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite die sicherstellt das alles korrekt kompiliert wurde. Wenn sie die Test−suite ausführen möchten erledigt dies das folgende Kommando:

```
make check
```

Und installieren sie es:

```
make install
```

Installieren von Coreutils−5.0

```
Geschätzte Kompilierzeit:              0.9 SBU
Ungefähr benötigter Festplattenplatz:  69 MB
```

Inhalt von Coreutils

Das Paket Coreutils enthält eine große Anzahl von Shell Werkzeugen.

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, kill, 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

Coreutils Installationsabhängigkeiten

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

Installieren von Coreutils

Bereiten sie Coreutils zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um sicherzustellen das alles korrekt gebaut wurde. Sie sollten die Test−suite ausführen; benutzen sie dazu das folgende Kommando:

```
make RUN_EXPENSIVE_TESTS=yes check
```

Die Bedeutung der make Parameter:

- **`RUN_EXPENSIVE_TESTS=yes`**: Dies teilt der Test−suite 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
```

Installieren von Bzip2−1.0.2

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  2.5 MB
```

Inhalt von Bzip2

Bzip2 ein Block−sortierendes Kompressionsprogramm und erreicht üblicherweise bessere Kompressionsraten als das herkömmliche **`gzip`**.

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

Bzip2 Installationsabhängigkeiten

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
```

Installieren von Gzip−1.3.5

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  2.6 MB
```

Inhalt von Gzip

Gzip enthält Programme für die Dateikompression und −dekompression mit Hilfe des Lempel−Ziv Algorythmus (LZ77).

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

Gzip Installationsabhängigkeiten

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 es:

```
make install
```

Installieren von Diffutils−2.8.1

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  7.5 MB
```

Inhalt von Diffutils

Die Programme dieses Pakets können die Unterschiede zwischen zwei Dateien oder Verzeichnissen anzeigen. Die häufigste Anwendung ist das erstellen von Software−Patches.

Installierte Programme: cmp, diff, diff3 und sdiff

Diffutils Installationsabhängigkeiten

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 es:

```
make install
```

Installieren von Findutils−4.1.20

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  69 MB
```

Inhalt von Findutils

Das Findutils Paket enthält Programme zum auffinden von Dateien, entweder on−the−fly (indem ein Verzeichnisbaum live durchsucht wird) oder durch Suche in einer Datenbank.

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

Findutils Installationsabhängigkeiten

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
```

Dieses Paket hat eine Test−suite die sicherstellt das alles korrekt kompiliert wurde. Falls sie diese durchlaufen wollen, erledigt dies das folgende Kommando:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Make−3.80

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  8.8 MB
```

Inhalt von Make

Make erkennt automatisch, welche Teile eines grossen Programmes erneut kompiliert werden müssen und welche Kommandos dazu nötig sind.

Installiertes Programm: make

Make Installationsabhängigkeiten

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

Installation von Make

Bereiten sie Make zum kompilieren vor:

```
./configure −−prefix=/tools
```

Kompilieren sie das Programm:

```
make
```

Dieses Paket enthält eine Test−suite zum überprüfen ob alles korrekt kompiliert wurde. Sollten sie sich entscheiden die Test−suite durchlaufen zu lassen, erledigt dies das folgende Kommando für sie:

```
make check
```

Nun installieren sie Make und die dazugehörige Dokumentation:

```
make install
```

Installieren von Grep−2.5.1

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  5.8 MB
```

Inhalt von Grep

Grep zeigt alle Zeilen einer Datei an die auf ein bestimmtes Muster passen.

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

Grep Installationsabhängigkeiten

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

Installation von Grep

Bereiten sie Grep zum kompilieren vor:

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

Die Bedeutung der configure Optionen:

- **`--disable-perl-regexp`**: Dies stellt sicher das **`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 das 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
```

Dieses Paket hat eine Test−suite die überprüft ob alles korrekt kompiliert wurde. Wenn sie diese laufen lassen wollen führen sie das folgende Kommando aus:

```
make check
```

Dann installieren sie sie und die dazugehörige Dokumentation:

```
make install
```

Installieren von Sed−4.0.7

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  5.2 MB
```

Inhalt von Sed

Sed ist ein Stream Editor. Mit einem Stream Editor kann man einfache Textmanipulationen an einem Eingabestream vornehmen (z. B. einer Datei oder einer Pipe).

Installiertes Programm: sed

Sed Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite um zu überprüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt das folgende Kommando dies für sie:

```
make check
```

Dann installieren sie Programm und Dokumentation:

```
make install
```

Installieren von Gettext−0.12.1

```
Geschätzte Kompilierzeit:              7.2 SBU
Ungefähr benötigter Festplattenplatz:  55 MB
```

Inhalt von Gettext

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

Installierte Programme: autopoint, config.charset, config.rpath, 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]

Gettext Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite zum überprüfen der kompilierten Programme. Es ist jedoch bekannt das die Gettext Test−suite hier in Kapitel 5 unter verschiedenen Bedingungen fehlschlägt −− zum Beispiel wenn sie einen Java Compiler auf dem Host−System findet. Die Gettext Test−suite braucht sehr viel Zeit und ist nicht als kritisch einzustufen. Deshalb empfehlen wir, diesen Schritt zu überspringen. Sollten sie dennoch die Test−suite laufen lassen wollen, erledigt dieses Kommando das für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Ncurses−5.3

```
Geschätzte Kompilierzeit:              0.7 SBU
Ungefähr benötigter Festplattenplatz:  26 MB
```

Inhalt von Ncurses

Ncurses enthält Bibliotheken zur Verwendung von Zeichen und Terminals, inklusive Schaltflächen und Menüs.

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]

Ncurses Installationsabhängigkeiten

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

Installation von Ncurses

Beheben sie zuerst zwei kleine Fehler:

```
patch -Np1 -i ../ncurses-5.3-etip-2.patch
patch -Np1 -i ../ncurses-5.3-vsscanf.patch
```

Der erste Patch korrigiert die Header Datei `etip.h`, und der zweite Patch unterbindet einige Compiler Warnungen bezüglich der Benutzung veralteter Header.

Bereiten sie Ncurses zum kompilieren vor:

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

Die bedeutung der configure Optionen:

- **`--without-ada`**: Das bewirkt, das 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 das andere Pakete die Ncurses Header Dateien problemlos finden können.

Kompilieren sie die Programme und Bibliotheken:

```
make
```

Und dann installieren sie sie und ihre Dokumentation:

```
make install
```

Installieren von Patch-2.5.4

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  1.9 MB
```

Inhalt von Patch

Patch manipuliert Dateien auf Basis einer Patch−Datei. Eine Patch−Datei ist üblicherweise eine Liste mit Änderungsanweisungen die mit Hilfe des Programms diff erzeugt wurde.

Installiertes Programm: patch

Patch Installationsabhängigkeiten

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

Installieren von Patch

Bereiten sie Patch zum kompilieren vor:

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

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

Kompilieren sie das Programm:

```
make
```

Und dann installieren sie es mit der dazugehörigen Dokumentation:

```
make install
```

Installieren von Tar−1.13.25

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  10 MB
```

Inhalt von Tar

Tar ist ein Programm zum speichern und extrahieren von Dateien aus sog. Tar−Archiven.

Installierte Programme: rmt und tar

Tar Installationsabhängigkeiten

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
```

Komipilieren sie die Programme:

```
make
```

Dieses Paket beinhaltet eine Test−suite um zu überprüfen das alles korrekt kompiliert wurde. Wenn sie die Suite durchlaufen lassen möchten, erledigt dies das folgende Kommando:

```
make check
```

Dann installieren sie die Programme und ihre Dokumentation:

```
make install
```

Installieren von Texinfo−4.6

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  16 MB
```

Inhalt von Texinfo

Texinfo enthält Programme zum lesen, schreiben und konvertieren von Info Dokumenten (System Dokumentation).

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

Texinfo Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite zum überprüfen ob alles korrekt kompiliert wurde. Wenn sie die Test−suite ausführen möchten, führen sie folgendes Kommando aus:

```
make check
```

Dann installieren sie sie und die Dokumentation:

```
make install
```

Installieren von Bash 2.05b

```
Geschätzte Kompilierzeit:              1.2 SBU
Ungefähr benötigter Festplattenplatz:  27 MB
```

Inhalt von Bash

Bash ist die "Bourne Again SHell", ein auf Unix Systemen weit verbreiteter Befehlsinterpreter. Die Bash liest Befehle von der Standard Eingabe (der Tastatur). Ein Anwender gibt etwas ein und die Bash wertet die Eingabe aus. Je nach Eingabe reagiert die Bash entsprechend und führt zum Beispiel ein Programm aus.

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

Bash Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite die mit einigen Prüfungen sicherstellt, das alles korrekt gebaut wurde. Sollten sie diese durchlaufen lassen wollen führen sie dieses Kommando aus:

```
make tests
```

Dann installieren sie das Programm und die Dokumentation:

```
make install
```

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

```
ln −s bash /tools/bin/sh
```

Installieren von Util−linux−2.12

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  8 MB
```

Inhalt von Util−linux

Util−linux enthält verschiedene Werkzeuge. Einige der etwas bekannteren Programme werden z. B. zum mounten, entmounten, formatieren, partitionieren und verwalten von Festplatten, öffnen von tty Ports und zum auslesen von Kernel Meldungen genutzt.

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, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, 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, test.bash, test.tcsh, tunelp, ul, umount, vidmode (Link auf rdev), whereis und write

Util−linux Installationsabhängigkeiten

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 /tools Verzeichnis. 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 Subroutinen:

```
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 unser temporäres tools Verzeichnis:

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

Installieren von Perl-5.8.0

```
Geschätzte Kompilierzeit:              0.8 SBU
Ungefähr benötigter Festplattenplatz:  74 MB
```

Inhalt von Perl

Das Perl Paket enthält die Skriptsprache Perl (Practical Extraction and Report Language). Perl kombiniert einige der besten Merkmale von C, sed, awk und sh in einer einzigen, mächtigen Skriptsprache.

Installierte Programme: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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 aufzulisten)

Perl Installationsabhängigkeiten

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-5.8.0-libc-3.patch
```

Und dann stellen sie sicher das ein paar statische Erweiterungen erzeugt werden:

```
chmod u+w hints/linux.sh
echo 'static_ext="IO re Fcntl"' >> hints/linux.sh
```

Bereiten sie Perl zum kompilieren vor:

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

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/perl5/5.8.0
cp -R lib/* /tools/lib/perl5/5.8.0
```

Stripping

Die Schritte in diesem Abschnitt sind optional. Wenn ihre LFS Partition sehr klein ist werden sie froh sein, das man einige unnötige Dinge loswerden kann. Die ausführbaren Dateien und Bibliotheken die sie bis hierher erstellt haben enthalten ungefäht 130MB nicht benötigte Debugging Symbole. So entfernen sie diese Symbole:

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

Das erste der obigen Kommandos übergeht rund 20 Dateien mit der Meldung das 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 Glibc neu kompilieren.

Um weiteren Platz zu sparen, können sie die Dokumentation entfernen:

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

Sie werden nun mindestens 850MB freien Platz auf ihrem LFS Dateisystem benötigen um die Glibc zu installieren. Wenn sie Glibc bauen und installieren können, werden sie mit allen restlichen Paketen keine Probleme haben.

III. Part III − Installation des LFS Systems

Inhaltsverzeichnis

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 eigentlichen LFS Systems. Im einzelnen chroot'en wir in unser temporäres Mini−Linux System, installieren 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, es wäre einfacher, wenn wir hier nur eine generelle Installationsanleitung geben würden um dann bei davon abweichenden Paketen eine vollständige Erklärung zu 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.

Falls sie vorhaben, in diesem Kapitel Compiler Optimierungen zu verwenden, lesen sie bitte die Anleitung unter http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt. Compiler Optimierungen können ein Programm ein wenig 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 ob das Problem dann weg ist. Selbst wenn das Paket mit Compiler Optmierungen kompilierbar ist, besteht die Gefahr das es falsch kompiliert wurde (zum beispiel wegen komplexer Iterationen zwischen Code 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 zu `/tools` erhalten. Kompilieren sie aus diesem Grund auch *nicht* gleichzeitig. Gleichzeitiges kompilieren könnte sie Zeit sparen, besonders auf Mehrprozessormaschinen, aber es kann in Programmen resultieren die Referenzen zu `/tools` enthalten, und das Programm würde nicht mehr funktionieren sobald dieses Verzeichnis entfernt wurde.

Informationen zu Debugging Symbolen

Die meisten Programme und Bibliotheken werden standardmässig mit debugging Symolen kompiliert (mit der gcc Option −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 angeben.

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 variiern ein wenig, abhängig davon welchen Compiler und welche C Bibliothek sie benutzen. Aber wenn man Programme mit und ohne debugging Symbole vergleicht, ist 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.

Um debugging Symbole aus einer Binärdatei zu entfernen (diese muss eine a.out oder ELF Binärdatei sein), führen sie **`strip --strip-debug Dateiname`** aus. Um mehrere Dateien gleichzeit zu behandeln können sie Platzhalter verwenden (benutzen sie z. B. **`strip --strip-debug $LFS/tools/bin/*`**).

Der Einfachheit halber finden sie in Kapitel 9 ein einziges 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.

Betreten der chroot Umgebung

Es ist nun an der Zeit, die chroot Umgebung zu betreten um mit dem installieren der benötigten Pakete zu beginnen. Bevor sie chroot'en können, müssen sie erst *root* werden, denn nur *root* kann das **`chroot`** Kommando benutzen.

So wie schon einmal, überprüfen sie das die LFS Umgebungsvariable korrekt ist indem sie **`echo $LFS`** ausführen und sicherstellen, das sie den Pfad zum Mount Punkt ihrer LFS Partition zeigt. Das ist `/mnt/lfs` wenn sie unserem Beispiel gefolgt sind.

Werden sie *root* und führen sie das folgende Kommando aus um die chroot Umgebung zu betreten:

```
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
```

Die **`-i`** Option zu **`env`** löscht alle Variablen in der chroot Umgebung. Danach werden nur die HOME, TERM, PS1 und PATH Variable wieder gesetzt. TERM=$TERM setzt die TERM Variable in der chroot Umgebung auf den gleichen Wert wie ausserhalb von chroot, diese Variable wird für Programme wie **`vim`** und **`less`** zur korrekten Funktion benötigt. Wenn sie andere Variablen wie CFLAGS oder CXXFLAGS brauchen ist dies ein guter Platz um sie erneut zu setzen.

Von nun an brauchen wir die LFS Variable nicht mehr weil alles was sie tun ausschliesslich auf das LFS System beschränkt ist -- weil das was die Shell für das `/` Verzeichnis hält, in Wirklichkeit der Wert der `LFS` Variable ist, welcher wiederum dem chroot Kommando übergeben wurde.

Beachten sie, das `/tools/bin` als letztes in der PATH Variable steht. Das bewirkt, das 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 etwas weiter hinten in der Befehlszeile abgeschaltet.

Sie müssen alle Kommandos im Rest des Kapitels in der chroot Umgebung ausführen. Wenn sie die chroot Umgebung aus irgendeinem Grund verlassen (Neustart zum Beispiel), dann denken sie daran die chroot Umgebung wieder zu betreten und das proc und devpts Dateisystem einzuhängen (das wird später behandelt)

bevor sie mit der Installation weitermachen.

Die Bash Eingabeaufforderung wird "I have no name!" ausgeben. Das ist normal weil die Datei `/etc/passwd` noch nicht erstellt wurde.

Ändern des Besitzers

Im moment besitzt der Benutzer *lfs* das Verzeichnis `/tools`, ein Benutzer der aber nur auf dem Host−System existiert. Auch wenn sie das Verzeichnis `/tools` nach der fertigen Installation von LFS löschen möchten, vielleicht entscheiden sie sich es doch aufzubewahren, zum Beispiel um noch mehr LFS System zu bauen. Doch wenn sie das `/tools` Verzeichnis in seinem jetzigen Zustand behalten haben sie Dateien mit einer Benutzer ID, zu der es kein Konto gibt. Das ist gefährlich, denn ein später erstelletes Konto könnte genau diese ID bekommen und wäre damit plötzlich der Besitzer des `/tools` Verzeichnisses und aller Dateien darin. Dieser Benutzer könnte alle Dateien unbemerkt manipulieren.

Um dieses Problem zu vermeiden könnten sie ihrem LFS System den *lfs* Benutzer 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 in dem Buch gehen wir davon aus das sie dies tun) den Inhalt des `/tools` Verzeichnisses 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 Passwort Datei noch nicht erzeugt wurde.

Erstellen der Verzeichnisse

Lassen sie uns nun Struktur in unser LFS System bringen. Lassen sie uns einen Verzeichnisbaum erstellen. Das folgende Kommando erstellt einen mehr oder weniger standardkonformen Verzeichnisbaum:

```
mkdir −p /{bin,boot,dev/{pts,shm},etc/opt,home,lib,mnt,proc}
mkdir −p /{root,sbin,tmp,usr/local,var,opt}
for dirname in /usr /usr/local
    do
    mkdir $dirname/{bin,etc,include,lib,sbin,share,src}
    ln −s share/{man,doc,info} $dirname
    mkdir $dirname/share/{dict,doc,info,locale,man}
    mkdir $dirname/share/{nls,misc,terminfo,zoneinfo}
    mkdir $dirname/share/man/man{1,2,3,4,5,6,7,8}
done
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}}
```

Verzeichnisse werden standardmässig mit den Rechten 755 erzeugt, aber das ist nicht für alle Verzeichnisse gewünscht. Wir nehmen zwei Änderungen vor: eine auf dem Heimatverzeichnis von *root*, und eine weitere auf dem Verzeichnis für temporäre Dateien.

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

Die erste Rechteänderung legt fest, das nicht jeder das `/root` Verzeichnis betreten darf −− das gleiche was ein normaler Benutzer mit seinem Heimatverzeichnis auch tun würde. Die zweite Änderung sorgt dafür, das jeder Benutzer in die Verzeichnisse `/tmp` und `/var/tmp` schreiben kann, aber nicht die Dateien anderer Benutzer löschen kann. Das letztere wird durch das "sticky bit" bewirkt −− dem höchsten Bit in der Bit Maske 1777.

Anmerkung zur FHS Konformität

Unser Verzeichnisbaum basiert auf dem FHS Standard (verfügbar unter http://www.pathname.com/fhs/). Zusätzlich zu den oben erstellten Verzeichnissen 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 Verzeichnisse erstellt die wir für nötig halten.

Einhängen des proc− und devpts Dateisystems

Damit bestimmte Programme richtig funktionieren, müssen die *proc* und *devpts* Dateisysteme in der chroot Umgebung verfügbar sein. Ein Dateisystem kann so oft und an so vielen Stellen eingehängt sein wie sie möchten, daher ist es auch kein Problem, das diese Dateisysteme auch auf ihrem Host−System bereits gemountet sind −− besonders weil diese zwei virtuelle Dateisysteme sind.

Das *proc* Dateisystem ist das Prozess Informations Pseudo−Dateisystem das der Kernel benutzt um Statusinformationen über den Zustand des Systems zur Verfügung zu stellen.

Das proc Dateisystem wird mit dem folgenden Kommando in das `/proc` Verzeichnis gemountet:

```
mount proc /proc −t proc
```

Sie könnten unter Umständen Warnungen erhalten, besonders eine wie diese:

```
warning: can't open /etc/fstab: No such file or directory
not enough memory
```

Ignorieren sie die Warnung, sie ist normal, weil das System noch nicht vollständig installiert ist und noch einige Dateien fehlen. Das einhängen selbst wird erfolgreich sein und das ist das einzig wichtige zu diesem Zeitpunkt.

Das *devpts* Dateisystem wurde schon früher erwähnt, es ist der übliche Weg um Pseudo Terminals (PTYs) zu implementieren.

Das devpts Dateisystem wird mit dem folgenden Kommando in das `/dev/pts` Verzeichnis eingehängt:

```
mount devpts /dev/pts −t devpts
```

Falls dieser Befehl mit dieser Meldung fehlschlägt:

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

dann ist der wahrscheinlichste Grund dafür, das der Kernel des Host−Systems ohne devpts Dateisystemunterstützung kompiliert wurde. Sie können mit diesem Kommando überprüfen welche Dateisysteme ihr Kernel untersützt: **`cat /proc/filesystems`**. Wenn dort ein Dateisystem mit dem Namen *devpts* aufgelistet ist, können wir das Problem umgehen indem wir das devfs Dateisystem des Host−Systems über unsere `/dev` Verzeichnisstruktur mounten − diese erzeugen wir später im Abschnitt "Erzeugen der Gerätedateien (Makedev)". Wenn devfs nicht aufgelistet wurde machen sie sich keine Sorgen, es gibt noch einen dritten Weg um PTYs in der chroot Umgebung ans laufen zu bekommen. Wir behandeln das Thema später in dem schon erwähnten Makedev Abschnitt.

Denken sie daran, wenn sie aus irgendeinem Grund die Arbeit an LFS beenden und später wieder einsteigen, dann müssen sie diese Dateisysteme in der chroot Umgebung erneut mounten. Ansonsten werden sie höchstwahrscheinlich Probleme bekommen.

Erstellen nötiger symbolischer Links

Ein paar 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 und group

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 echte 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 MAKEDEV Skript im nächsten Abschnitt benutzt. Neben der Gruppe "root" schlägt das LSB (http://www.linuxbase.org) 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.

Zum schluss loggen wir uns erneut in die chroot Umgebung ein. Die Auflösung von Benutzer− und Gruppennamen funktioniert sofort nach dem Erstellen von `/etc/passwd` und `/etc/group`, weil wir in Kapitel 5 eine vollständige Glibc installiert haben. Jetzt sind wir endlich den I have no name! Prompt los.

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

Beachten sie die Benutzung der **+h** Anweisung. 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 diese Funktion in diesem Kapitel aus.

Erstellen der Gerätedateien (Makedev−1.7)

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  50 KB
```

Inhalt von MAKEDEV

Das Skript MAKEDEV erstellt statische Gerätedateien. Diese liegen normalerweise im `/dev` Verzeichnis. Detaillierte Informationen über die Gerätedateien finden sie in der Datei `Documentation/devices.txt` in den Linux Kernel Quellen.

Installiertes Skript: MAKEDEV

MAKEDEV Installationsabhängigkeiten

Make ist abhängig von: Bash, Coreutils.

Erstellen von Gerätedateien

Beim entpacken von `MAKEDEV−1.7.bz2` wird keine neues Verzeichnis erstellt in welches sie **cd**'en könnten, da es nur ein Shell Skript enthält.

Installieren sie das **MAKEDEV** Skript:

```
bzcat MAKEDEV-1.7.bz2 > /dev/MAKEDEV
chmod 754 /dev/MAKEDEV
```

Führen sie das Skript aus um die Gerätedateien zu erzeugen:

```
cd /dev
./MAKEDEV -v generic-nopty
```

Die Bedeutung der Argumente:

- **-v**: Dadurch wird das Skript "gesprächiger", der sogenannte verbose Modus.
- **generic-nopty**: Das weist **MAKEDEV** an, die übliche Auswahl der gängigen Spezialdateien zu erstellen, ausser der ptyXX und ttyXX Dateien. Wir brauchen diese Dateien nicht, weil wir Unix98 PTYs über das *devpts* Dateisystem benutzen werden.

Falls sich herausstellen sollte, das ihnen z. B. die Spezialdatei `zzz` fehlt, führen sie einfach **./MAKEDEV -v zzz** aus. Alternativ können sie auch das **mknod** Programm dazu benutzen. Bitte lesen sie seine Manpage wenn sie dazu mehr Informationen benötigen.

Falls sie desweiteren im Kapitel "Einhängen des proc- und devpts Dateisystem" Schwierigkeiten beim mounten hatten, ist jetzt eine gute Gelegenheit die Alternativen zu versuchen. Wenn ihr Kernel das devfs Dateisystem unterstützt führen sie folgendes Kommando aus:

```
mount -t devfs devfs /dev
```

Das mountet das devfs Dateisystem über die statische Struktur des `/etc` Verzeichnisses. Das ist kein Problem, denn die erstellen Gerätedateien werden dadurch nicht gelöscht, das devfs Dateisystem versteckt sie lediglich.

Wenn auch das nicht funktioniert, bleibt ihnen nur die Möglichkeit, mittels dem MAKEDEV Skript auch die ptyXX und ttyXX Dateien zu ereugen, die natürlich sonst nicht benötigt werden würden. Stellen sie erst sicher das sie sich noch im `/dev` Verzeichnis befinden und führen sie dann **./MAKEDEV -v pty** aus. Das unschöne daran ist, das wir zusätzliche 512 spezielle Gerätetreiber Dateien erzeugen, die dann später nach dem fertigstellen des LFS Systems nicht mehr benötigt werden.

Installieren der Linux-2.4.22 Header

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  186 MB
```

Inhalt von Linux

Der Linux Kernel ist der Kern eines jeden Linux Systems. Er ist sozusagen der Herzschlag von Linux. Wenn der Computer eingeschaltet wird und ein Linux System startet, dann ist der Kernel das erste Stück Software das gestartet wird. Der Kernel initialisiert die Geräte und Hardware Komponenten: serielle Schnittstellen, parallele Schnittstellen, Soundkarten, Netzwerkkarten, IDE und SCSI Controller und vieles mehr. Zusammenfassend kann man sagen, der Kernel stellt dem System die Hardware zur Verfügung, so das die Software damit laufen kann.

Installierte Dateien: Der Kernel und die Kernel Header

Linux Installationsabhängigkeiten

Linux ist abhängig von: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Installation der Kernel Header

Wir werden jetzt noch keinen neuen Kernel kompilieren −− das erledigen wir, wenn wir die Installation aller Pakete abgeschlossen haben. Da aber einige Pakete die Kernel Header benötigen entpacken wir nun das Kernel Archiv, bereiten es vor und kopieren die Header Dateien damit sie von diesen Paketen gefunden werden.

Beachten sie bitte das die Dateien im Kernel Quellverzeichnis 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 das Quellverzeichnis nach der Installation löschen. Aber die Linux Quellen liegen normalerweise sehr lange auf ihrem Computer, daher ist die Chance gross, das ein zukünftiger Benutzer auf ihrem System die Benutzer ID erhält die ihre Kernel Quellen nun 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 das `linux-2.4.22` Verzeichnis anwenden damit alle Dateien dem *root* Benutzer gehören.

Bereiten sie die Installation der Header vor:

```
make mrproper
```

Dadurch wird sichergestellt, das die Kernelquellen absolut sauber sind. Das Kernel Team empfiehlt, dieses Kommando vor *jedem* Neubau des Kernels auszuführen. Sie sollten sich nicht darauf verlassen, das die Quellen nach dem entpacken sauber sind.

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

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

Erstellen sie den Plattform−spezifischen symbolischen Link `include/asm`:

```
make symlinks
```

Installieren sie die Plattform−spezifischen Header Dateien:

```
cp -HR include/asm /usr/include
cp -R include/asm-generic /usr/include
```

Installieren sie die Cross−Plattform Header Dateien:

```
cp -R include/linux /usr/include
```

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

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

Warum wir die Kernel Header kopieren und nicht symbolisch linken

Früher war es gängige Praxis, das Verzeichnis `/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 Mailinglist 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, wo Linux sagt, das die Header Dateien die sein sollen, *mit denen Glibc kompiliert wurde*. Das sind die Header Dateien die zum späteren kompilieren von Paketen verwendet werden sollten, weil nur diese exakt auf die Objekt Code Bibliotheken passen. Durch das kopieren der Header stellen wir sicher, das sie verfügbar bleiben falls sie später den Kernel updaten.

Beachten sie, das es vollkommen in Ordnung ist, die Kernel Sourcen in `/usr/src/linux` liegen zu haben, so lange sie nicht die symbolischen Links `/usr/include/{linux,asm}` haben.

Installieren der Man−pages−1.60

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  15 MB
```

Inhalt von Man−pages

Die Hilfeseiten (man−pages) enthalten über 1200 Seiten Hilfetexte. Die Dokumentation beschreibt sehr detailliert C und C++ Funktionen, einige wichtige Gerätedateien und enthält Dokumente die in anderen Paketen sonst fehlen würden.

Installierte Dateien: verschiedene Hilfeseiten

Man−pages Installationsabhängigkeiten

Man ist abhängig von: Bash, Coreutils, Make.

Installation der Man−pages

Installieren sie die Man−pages durch ausführen von:

```
make install
```

Installieren von Glibc−2.3.2

```
Geschätzte Kompilierzeit:              12.3 SBU
Ungefähr benötigter Festplattenplatz:  784 MB
```

Inhalt von Glibc

Glibc ist die C Bibliothek, sie stellt Systemaufrufe und grundlegende Funktionen wie open, malloc, printf usw. zur Verfügung. Die C Bibliothek wird von allen dynamisch gelinkten Programmen verwendet.

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]

Glibc Installationsabhängigkeiten

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

Installation von Glibc

Das Glibc Installatinssystem ist sehr eigenständig und installiert perfekt, selbst wenn unsere Compiler Specs Datei und der Linker immer noch auf `/tools` zeigen. Wir können die Specs Datei und den Linker nicht vor der Installation von Glibc anpassen, weil die Glibc Autoconf Tests dann falsche Resultate ergeben würden.

> **Anmerkung:** Die Test−suite von Glibc in diesem Abschnitt wird als *kritisch* betrachtet. Wir raten, sie unter keinen Umständen zu überspringen.

Bevor sie mit dem kompilieren von Glibc beginnen, denken sie daran Glibc−linuxthreads in dem `glibc−2.3.2` Verzeichnis zu entpacken, und setzen sie alle Umgebungsvariablen zurück die die standard Optimierungen übergehen würden.

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

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

Dann wenden sie den gleichen Patch an den wir schon früher verwendet haben:

```
patch −Np1 −i ../glibc−2.3.2−sscanf−1.patch
```

Die Glibc Dokumentation empfiehlt, Glibc nicht im Quellenverzeichnis, sondern stattdessen in einem dedizierten Kompilierverzeichnis zu bauen:

```
mkdir ../glibc−build
cd ../glibc−build
```

Bereiten sie nun Glibc zum kompilieren vor:

```
../glibc−2.3.2/configure −−prefix=/usr \
    −−disable−profile −−enable−add−ons \
    −−libexecdir=/usr/bin −−with−headers=/usr/include
```

Die Bedeutung der neuen configure Optionen:

- **`−−libexecdir=/usr/bin`**: Das wird das `pt_chown` Programm in `/usr/bin` installieren.
- **`−−with−headers=/usr/include`**: Das stellt sicher, das 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 identisch sein sollten). Diese Option hat auch den Vorteil das sie sofort merken, wenn sie vergessen haben sollten, die Kernel Header in `/usr/include` zu installieren.

Kompilieren sie das Paket:

```
make
```

Testen sie das Ergebnis:

```
make check
```

Die Anmerkungen zur Test−suite aus dem Abschnitt namens *Installieren von Glibc−2.3.2* in Kapitel 5 gelten natürlich auch hier. Schlagen sie dort nach falls sie irgendwelche Zweifel haben.

Und installieren sie das Paket:

```
make install
```

Die locales wurden durch das obige Kommando nicht installiert. Holen sie das nach:

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

Eine alternative zum obigen Kommando wäre, nur die benötigten oder gewollten locales zu installieren. Das erreichen sie mit dem **`localedef`** Befehl. Informationen dazu finden sie in der `INSTALL` Datei im `glibc-2.3.2` Verzeichnisbaum. Einige locales sind allerdings essentiell für die Tests einiger nachfolgender Pakete. Die nachfolgenden Kommandos anstelle des obigen installieren nur ein Minimum an locales so das die Tests sauber durchlaufen:

```
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 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-2.3.2/linuxthreads/man
```

Und installieren die Pakete:

```
make -C ../glibc-2.3.2/linuxthreads/man install
```

Konfigurieren von Glibc

Wir müssen die Datei `/etc/nsswitch.conf` erstellen, denn obwohl Glibc Standardwerte vorgibt falls die Datei fehlt oder kaputt ist, funktionieren diese nicht gut mit Netzwerken. Und wir müssen die Zeitzone korrekt einrichten.

Erstellen sie die neue Datei `/etc/nsswitch.conf` indem sie das folgende ausführen:

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

passwd: files
group: files
shadow: files

publickey: files

hosts: files dns
networks: files

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

netgroup: db files
```

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

Um herauszufinden in welcher Zeitzone sie sind, führen sie das folgende Skript aus:

```
tzselect
```

Wenn sie ein paar Fragen zu ihrer Lokation 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, falls `/usr` auf einer separaten Partition liegt. Das könnte z. B. wichtig 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

Standardmässig such der dynamisch 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` einfügen damit der dynamische Lader diese finden kann. Zwei Verzeichnisse die dafür bekannt sind, weitere Bibliotheken zu enthalten, sind `/usr/local/lib` und `/opt/lib`, also fügen wir diese Verzeichnisse 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
```

Erneutes anpassen der toolchain

Nun wo die neue C Bibliothek installiert ist, muss die toolchain erneut angepasst werden. Wir modifizieren sie so, das alle weiteren kompilierten Programme gegen die neue C Bibliothek gelinkt werden. Im Grunde ist das genau das Gegenteil von dem was wir im vorigen Kapitel beim einhängen der Glibc gemacht haben.

Als erstes wird der Linker angepasst. Aus diesem Grunde haben wir die Quell− und Kompilierverzeichnisse aus dem zweiten Durchlauf von Binutils bestehen lassen. Installieren sie den angepassten Linker aus dem `binutils−build` Verzeichnis:

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

> **Anmerkung:** Falls sie aus irgendeinem Grund die Warnung, das Binutils Verzeichnis zu behalten, übersehen haben oder es vielleicht versehentlich gelöscht haben, es ist noch nichts verloren. Ignorieren sie einfach das obige Kommando. Das Ergebnis ist dann, das 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 das 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 Symlink `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 Kompilierverzeichnisse löschen.

Als nächstes passen sie die GCC Specs Datei an, so das sie auf den neuen dynamischen Linker zeigt. 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 zu überprüfen ob die Änderungen tatsächlich gemacht wurden.

> **Wichtig:** Wenn sie an einer Plattform arbeiten wo 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 namens *Technische Anmerkungen zur toolchain* in Kapitel 5 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
gcc dummy.c
readelf −l a.out | grep ': /lib'
```

Wenn alles richtig funktioniert sollte es keine Fehlermeldungen geben und die Ausgabe des letzten Kommandos sollte dann so aussehen:

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

Wenn sie eine andere Ausgabe erhalten oder überhaupt keine Ausgabe erscheint, 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, das etwas beim anpassen der Specs Datei weiter oben nicht funktioniert hat. Achten sie besonders darauf, das `/lib` nun als Prefix zu unserem dynamischen Linker angezeigt wird. Wenn sie an einer Plattform arbeiten an der der Name des dynamischen Linkers nicht `ld-linux.so.2` ist, sieht die Ausgabe natürlich ein klein wenig anders aus.

Wenn sie mit dem Ergebnis zufrieden sind, löschen sie die Test Dateien:

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

Installieren von Binutils−2.14

```
Geschätzte Kompilierzeit:              1.4 SBU
Ungefähr benötigter Festplattenplatz:  167 MB
```

Inhalt von Binutils

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

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]

Binutils Installationsabhängigkeiten

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

Installation von Binutils

Jetzt ist ein guter Zeitpunkt um zu überprüfen das 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 PTYs 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 namens *Einhängen des proc− und devpts Dateisystems* und im Abschnitt namens *Erstellen der Gerätedateien*

(Makedev−1.7) nach und führen sie die empfohlenen Schritte durch um das Problem zu beseitigen.

> **Anmerkung:** Die Binutils Test−suite in diesem Abschnitt wird als *kritisch* eingestuft. Wir raten ihnen, die Tests unter keinen Umständen zu überspringen.

Es ist bekannt, das dieses Paket nicht gut funktioniert wenn sie die standard Optimierungen (inklusive der −march und −mcpu Optionen) geändert haben. Wenn sie Umgebungsvariablen wie CFLAGS oder CXXFLAGS gesetzt haben, empfehlen wir, diese zum installieren von Binutils zurückzusetzen.

Die Binutils Dokumentation empfiehlt, Binutils nicht im Quell−, sondern in einem dedizierten Kompilierverzeichnis zu kompilieren:

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

Bereiten sie Binutils jetzt zum kompilieren vor:

```
../binutils-2.14/configure \
    --prefix=/usr --enable-shared
```

Kompilieren sie das Paket:

```
make tooldir=/usr
```

Normalerweise ist *tooldir* (das Verzeichnis wo die ausführbaren Dateien hineininstalliert werden) auf $(exec_prefix)/$(target_alias) gesetzt, welches dann zum Beispiel zu `/usr/i686-pc-linux-gnu` erweitert wird. Da wir aber nur für unser eigenes System installieren brauchen wir dieses spezielle Verzeichnis 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).

Testen sie das Ergebnis:

```
make check
```

Die Anmerkungen zur Test−suite aus dem Abschnitt namens *Installieren von Binutils−2.14 − Durchlauf 2* in Kapitel 5 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
```

Installieren von GCC−3.3.1

```
Geschätzte Kompilierzeit:             11.7 SBU
Ungefähr benötigter Festplattenplatz:  294 MB
```

Inhalt von GCC

Das Paket GCC enthält die Gnu Compiler Sammlung, inklusive dem C und C++ Compiler.

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

GCC Installationsabhängigkeiten

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

Installation von GCC

> **Anmerkung:** Die Test−suite für GCC in diesem Abschnitt wird als *kritisch* betrachtet. Unser Rat ist, sie unter keinen Umständen zu überspringen.

Es ist bekannt, das dieses Paket nicht sauber funktioniert wenn die standard Optimierungseinstellungen (inklusive der −march und −mcpu Optionen) verändert wurden. Deshalb sollten sie event. gesetzte Umgebungsvariablen die die Standard Optimierung überschreiben − zum Beispiel CFLAGS und CXXFLAGS − für den Kompiliervorgang von GCC zurücksetzen oder entsprechend abändern.

Dieses mal werden wir sowohl den C als auch den C++ Compiler erzeugen, sie müssen also die GCC−core *und* GCC−g++ Tar−Archive entpacken −− sie entpacken sich automatisch in das selbe Verzeichnis. Genauso sollten sie auch das GCC−testsuite Archiv entpacken. Das vollständige GCC Paket enthält noch mehr Compiler. Eine Anleitung wie sie diese bauen können finden sie unter http://www.linuxfromscratch.org/blfs/view/stable/general/gcc.html.

```
patch −Np1 −i ../gcc−3.3.1−no_fixincludes−2.patch
patch −Np1 −i ../gcc−3.3.1−suppress−libiberty.patch
```

Der zweite Patch verhindert die Installation von libiberty von GCC, weil wir stattdessen die Version von Binutils verwenden. Geben sie acht das sie hier *nicht* den GCC specs Patch aus Kapitel 5 anwenden.

Die GCC Dokumentation empfiehlt, GCC nicht im Quellenverzeichnis sondern in einem dedizierten Kompilierverzeichnis zu kompilieren:

```
mkdir ../gcc−build
cd ../gcc−build
```

Bereiten sie nun GCC zum kompilieren vor:

```
../gcc−3.3.1/configure −−prefix=/usr \
    −−enable−shared −−enable−threads=posix \
    −−enable−__cxa_atexit −−enable−clocale=gnu \
    −−enable−languages=c,c++
```

Kompilieren sie das Paket:

```
make
```

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

```
make -k check
```

Die Test−suite Anmerkungen aus dem Abschnitt namens *Installieren von GCC−3.3.1 − Durchlauf 2* in Kapitel 5 gelten auch hier noch. Schlagen sie dort nach falls sie irgendwelche Zweifel haben.

Und installieren sie das Paket:

```
make install
```

Einige Pakete erwarten das der C PreProzessor im `/lib` Verzeichnis 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 diese Pakete zufriedenzustellen 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 Abschnitt namens *Erneutes anpassen der toolchain* nach und wiederholen sie den Test. Wenn das Ergebnis negativ ist haben sie möglichweise versehentlich den GCC Specs Patch aus Kapitel 5 angewendet.

Installieren von Coreutils−5.0

```
Geschätzte Kompilierzeit:              0.9 SBU
Ungefähr benötigter Festplattenplatz:  69 MB
```

Inhalt von Coreutils

Das Paket Coreutils enthält eine große Anzahl von Shell Werkzeugen.

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, kill, 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

Coreutils Installationsabhängigkeiten

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

Installation von Coreutils

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

```
patch −Np1 −i ../coreutils−5.0−uname.patch
```

Wir möchten nicht, das 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−5.0−hostname−2.patch
```

Bereiten sie Coreutils zum kompilieren vor:

```
./configure −−prefix=/usr
```

Kompilieren sie das Paket:

```
make
```

Das Programm **su** aus Coreutils wurde in Kapitel 5 nicht installiert weil man dazu *root* Rechte benötigt. Wir brauchen es aber gleich für die Test−suite. Daher installieren wir es nun:

```
make install−root
```

Dieses Paket hat eine Test−suite mit der sie überprüfen können ob alles korrekt kompiliert wurde. Diese Test−suite macht einige Annahmen in Hinsicht auf die Existenz von nicht−root Benutzern und Gruppen und diese sind in in unserem LFS System noch nicht vorhanden. Wir erstellen nun einen dummy System Benutzer und zwei dummy Gruppen damit die Tests sauber durchlauen können. Falls sie diese Test−suite nicht ausführen möchten, überspringen sie sie und fahren sie mit "Installieren sie das Paket" fort. Das folgende Kommando bereitet uns auf das ausführen der Test−suite vor. Erstellen sie zwei dummy Gruppen und einen dummy Benutzernamen:

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

Ein paar Tests müssen als *root* ausgeführt werden:

```
make check−root
```

Die verbleibenden Tests werden als *dummy* Benutzer ausgeführt:

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

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/{dir,dircolors,du,date,echo,false,head} /bin
mv /usr/bin/{install,ln,ls,mkdir,mkfifo,mknod,mv,pwd} /bin
mv /usr/bin/{rm,rmdir,shred,sync,sleep,stty,su,test} /bin
mv /usr/bin/{touch,true,uname,vdir} /bin
mv /usr/bin/chroot /usr/sbin
```

Schliesslich erstellen sie noch ein paar nötige symbolische Links:

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

Installieren von Zlib−1.1.4

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  1.5 MB
```

Inhalt von Zlib

Zlib enthält die Bibliothek libz. Sie wird von vielen Programmen zum komprimieren und dekomprimieren genutzt.

Installierte Bibliotheken: libz[a,so]

Zlib Installationsabhängigkeiten

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

Installation von Zlib

Zlib enthält einen potentiellen Pufferüberlauf in der *gzprintf()* Funktion, welcher, wenn er auch schwierig auszunutzen ist, durch den folgenden Patch behoben werden sollte:

```
patch −Np1 −i ../zlib−1.1.4−vsnprintf.patch
```

Bereiten sie Zlib zum kompilieren vor:

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

Vorsicht: Zlib baut seine gemeinsamen Bibliotheken falsch wenn die CFLAGS Umgebungsvariable gesetzt ist. Wenn sie die CFLAGS Umgebungsvariable verwenden, fügen sie ihr während dem bauen und der Installation von Zlib *-fPIC* an und entfernen sie es später wieder.

Kompilieren sie das Paket:

```
make
```

Installieren sie die gemeinsamen Bibliotheken:

```
make install
```

Kompilieren sie nun die nicht-gemeinsamen Bibliotheken:

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

Dieses Paket enthält eine Test-suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make test
```

Und installieren sie das Paket:

```
make install
```

Die gemeinsame Zlib Bibliothek sollte in `/lib` installiert werden. Auf diese Weise haben beim booten, während `/usr` möglicherweise noch nicht verfübar ist, Systemprogramme trotzdem Zugriff zu dieser Bibliothek:

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

Der symbolische Link `/usr/lib/libz.so` zeigt 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
```

Zlib installiert seine Man-pages nicht. Holen sie das mit dem folgenden Kommando nach:

```
cp zlib.3 /usr/share/man/man3
```

Installieren von Lfs-Utils-0.3

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  1.1 MB
```

Inhalt von Lfs−Utils

Das Lfs−Utils Paket enthält ein paar Programme die von verschiedenen Paketen gebraucht werden, aber nicht gross genug sind um ein eigenes Paket zu rechtfertigen.

Installierte Programme: mktemp, tempfile, http−get und iana−net

Installierte Dateien: protocols, services

Lfs−Utils Installationsabhängigkeiten

(No dependencies checked yet.)

Installation von Lfs−Utils

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Kopieren sie noch zwei mitgelieferte Hilfsdateien aus dem Lfs−Utils Archiv an die richtige Stelle:

```
cp etc/{services,protocols} /etc
```

Die Datei `/etc/services` wird zum auflösen von Service Nummern in lesbare Namen verwendet, und `/etc/protocols` erfüllt den gleichen Zweck für Protokollnummern.

Installieren von Findutils−4.1.20

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  7.5 MB
```

Inhalt von Findutils

Das Findutils Paket enthält Programme zum auffinden von Dateien, entweder on−the−fly (indem ein Verzeichnisbaum live durchsucht wird) oder durch Suche in einer Datenbank.

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

Findutils Installationsabhängigkeiten

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

Installation von Findutils

Bereiten sie Findutils zum kompilieren vor:

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

Standardmässig liegt die updatedb Datenbank in `/usr/var`. Um FHS Konform zu sein wird die Datei aber in `/var/lib/misc/locatedb` abgelegt. Daher wird die configure Option *--localstatedir=/var/lib/misc* angegeben.

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test-suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie die Tests durchlaufen lassen möchten, führen sie dieses Kommando aus:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Gawk-3.1.3

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  17 MB
```

Inhalt von Gawk

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

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

Gawk Installationsabhängigkeiten

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

Installation von Gawk

Als erstes wenden sie einen Patch an um die folgenden Probleme zu beseitigen:

- Gawk's Standard Platz für einige seiner ausführbaren Dateien ist `$prefix/libexec/awk`. Dieser Pfad ist nicht konform mit dem FHS, welches ein Verzeichnis namens `libexec` noch nicht einmal erwähnt. Der Patch ermöglicht es, die *--libexecdir* Option an das configure Skript zu übergeben, sodass wir einen passenderen Ort für **`grcat`** und **`pwcat`** verwenden können: `/usr/bin`.
- Gawk's Standard Datenverzeichnis ist `$prefix/share/awk`. Aber Paketspezifische Verzeichnisse sollten den Paketnamen und die Versionsnummer enthalten (zum Beispiel `gawk-7.7.2.`) und nicht einfach nur den Paketnamen, denn es könnten verschiedene Versionen dieses Pakets auf dem System installiert sein. Der Patch ändert den Namen des Datenverzeichnisses auf den korrekten Namen `$prefix/share/gawk-3.1.3`.
- Der Patch stellt ausserdem sicher, das dieses Datenverzeichnis beim Aufruf von *make uninstall* wieder entfernt wird.

```
patch -Np1 -i ../gawk-3.1.3-libexecdir.patch
```

Bereiten sie nun Gawk zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test-suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Ncurses-5.3

```
Geschätzte Kompilierzeit:              0.6 SBU
Ungefähr benötigter Festplattenplatz:  27 MB
```

Inhalt von Ncurses

Ncurses enthält Bibliotheken zur Verwendung von Zeichen und Terminals, inklusive Schaltflächen und Menüs.

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]

Ncurses Installationsabhängigkeiten

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

Installation von Ncurses

Beheben sie zuerst zwei kleine Fehler:

```
patch -Np1 -i ../ncurses-5.3-etip-2.patch
patch -Np1 -i ../ncurses-5.3-vsscanf.patch
```

Der erste Patch korrigiert die Header Datei `etip.h`, und der zweite Patch verhindert einige Compiler Warnung über alte, missbilligte Header Dateien.

Bereiten sie Ncurses nun zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Installieren sie das Paket:

```
make install
```

Geben sie der Ncurses Bibliothek Ausführrechte:

```
chmod 755 /usr/lib/*.5.3
```

Und korrigieren sie eine Bibltiothek die nicht ausführbar sein sollte:

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

Verschieben sie die Bibltiotheken in das `/lib` Verzeichnis, denn es wird erwartet das sie sich dort befinden:

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

Da die Bibliotheken nach `/lib` verschoben wurden, zeigen ein paar symbolische Links ins Leere. Ertellen sie diese Links neu:

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

Installieren von Vim−6.2

```
Geschätzte Kompilierzeit:              0.4 SBU
Ungefähr benötigter Festplattenplatz:  34 MB
```

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.

Inhalt von Vim

Vim enthält einen sehr anpassungsfähigen Text Editor, extra programmiert zum effizienten bearbeiten von Text.

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

Vim Installationsabhängigkeiten

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

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
```

Und installieren sie das Paket:

```
make install
```

Vim kann im klassischen *vi* Modus laufen indem sie einen symbolischen Link erzeugen:

```
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 nochmal neu 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

Standardmässig läuft vim im vi Kompatibilitätsmodus. Einige Leute mögen das so, aber wir würden vim gern im vim Modus ausführen (sonst hätten wir vim nicht in diesem Buch installiert, sondern gleich vi). Erstellen sie die Datei `/root/.vimrc` mit dem folgenden Kommando:

```
cat > /root/.vimrc << "EOF"
" Begin /root/.vimrc

set nocompatible
set bs=2

" End /root/.vimrc
EOF
```

Installieren von M4−1.4

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  3.0 MB
```

Inhalt von M4

M4 ist ein Makroprozessor. Er kopiert die Eingabe zur Ausgabe und führt dabei Makros aus. Die Makros können entweder vordefiniert oder selbstgeschrieben sein und können beliebige Argumente übernehmen. Neben der Fähigkeit Makros auszuführen hat M4 eingaute Funktionen um benannte Dateien einzufügen, Unix Kommandos auszuführen, Integer Berechnungen durchzuführen, Text zu manipulieren, Rekursionen zu behandeln usw. M4 kann entweder als Front−end zu einem Compiler oder als eigenständiger Makroprozessor genutzt werden.

Installierte Programme: m4

M4 Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Bison−1.875

```
Geschätzte Kompilierzeit:              0.6 SBU
Ungefähr benötigter Festplattenplatz:  10.6 MB
```

Inhalt von Bison

Bison ist ein Programm zum erstellen von Analysatoren, ein Ersatz für yacc. Bison erstellt ein Programm welches die Struktur einer Textdatei analysiert.

Installierte Programme: bison und yacc

Installierte Bibliotheken: liby.a

Bison Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Falls sie diese Tests ausführen möchten, benutzen sie dieses Kommando:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Less−381

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  3.4 MB
```

Inhalt von Less

Less ist ein Textanzeigeprogramm. Es zeigt den Inhalt von Dateien oder Datenströmen an. Less hat einige Merkmale die **`more`** nicht hat, wie zum Beispiel die Möglichkeit rückwärts zu scrollen.

Installierte Programme: less, lessecho und lesskey

Less Installationsabhängigkeiten

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, das die in diesem Paket installierten Programme ihre Konfigurationsdateien in `/etc` suchen.

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Installieren von Groff−1.19

```
Geschätzte Kompilierzeit:              0.5 SBU
Ungefähr benötigter Festplattenplatz:  43 MB
```

Inhalt von Groff

Groff enthält verschiedene Programme zur verarbeitung und formatierung von Text. Groff konvertiert normalen Text mit speziellen Kommandos in eine formatierte Ausgabe, so wie man es zum Beispiel in den Hilfeseiten (man−pages) sieht.

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 and zsoelim (Link auf soelim)

Groff Installationsabhängigkeiten

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

Installation von Gross

Groff erwartet, das die Umgebungsvariable PAGE die standard Papiergröße enthält. Für alle in den Vereinigten Staaten ist das untenstehende Kommando so korrekt. Wenn sie woanders leben ersetzen sie besser *PAGE=letter* durch *PAGE=A4*.

Bereiten sie Groff zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Eine 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
```

Installieren von Sed−4.0.7

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  5.2 MB
```

Inhalt von Sed

Sed ist ein Stream Editor. Mit einem Stream Editor kann man einfache Textmanipulationen an einem Eingabestream vornehmen (z. B. einer Datei oder einer Pipe).

Installiertes Programm: sed

Sed Installationsabhängigkeiten

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

Installation von Sed

Bereiten sie Sed zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Flex−2.5.4a

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  3.4 MB
```

Inhalt von Flex

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

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

Installierte Bibliothek: libfl.a

Flex Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make bigcheck
```

Und 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 diese Programme dennoch zu unterstützen erzeugen sie ein kleine Shell Skript mit dem Namen `lex` das **`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
```

Installieren von Gettext−0.12.1

```
Geschätzte Kompilierzeit:              6.9 SBU
Ungefähr benötigter Festplattenplatz:  55 MB
```

Inhalt von Gettext

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

Installierte Programme: autopoint, config.charset, config.rpath, 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]

Gettext Installationsabhängigkeiten

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

Installation von Gettext

Bereiten sie Gettext zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie (das benötigt sehr viel Zeit):

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Net−tools−1.60

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  9.4 MB
```

Inhalt von Net−tools

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

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

ypdomainname (Link auf hostname)

Net−tools Installationsabhängigkeiten

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 sollten, dann akzeptieren sie einfach den Vorgabewert. Das ist in den meisten Fällen richtig. Sie werden einige Fragen gestellt bekommen 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 zur Verfügung zu stellen.

Beheben sie zuerst ein kleines Problem in den Quellen des mii Hilfsprogramm:

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

Bereiten sie nun Net−tools zum kompilieren vor:

```
make config
```

Wenn sie generell die Standardwerte annehmen möchten die ihnen von *make config* vorgeschlagen werden, dann können sie stattdessen auch **`yes "" | make config`** ausführen.

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make update
```

Installieren von Inetutils−1.4.2

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  11 MB
```

Inhalt von Inetutils

Inetutils enthält verschiedene Netzwerk Clients und Server.

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

Inetutils Installationsabhängigkeiten

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

Installation von Inetutils

Bereiten sie Inetutils zum kompilieren vor:

```
./configure --prefix=/usr --disable-syslogd \
    --libexecdir=/usr/sbin --disable-logger \
    --sysconfdir=/etc --localstatedir=/var \
    --disable-whois --disable-servers
```

Die Bedeutung der configure Optionen:

- **`--disable-syslogd`**: Diese Option verhindert die Installation des System Log Dämonen, weil wir einen mit dem Sysklogd Paket installieren.
- **`--disable-logger`**: Das verhindert die Installation des logger Programmes, welches Nachrichten an den System Log Dämonen übergibt. Wir installieren ihn nicht, weil Util−Linux etwas später eine bessere Version installiert.
- **`--disable-whois`**: Das 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, das es für fast alle dieser Netzwerkserver einen besseren Ersatz gibt.

Kompilieren sie das Paket:

```
make
```

Installieren sie es:

```
make install
```

Und verschieben sie das **`ping`** Programm an seine korrekte Stelle:

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

Installieren von Perl−5.8.0

```
Geschätzte Kompilierzeit:              2.9 SBU
Ungefähr benötigter Festplattenplatz:  143 MB
```

Inhalt von Perl

Das Perl Paket enthält die Skriptsprache Perl (Practical Extraction and Report Language). Perl kombiniert einige der besten Merkmale von C, sed, awk und sh in einer einzigen, mächtigen Skriptsprache.

Installierte Programme: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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 aufzulisten)

Perl Installationsabhängigkeiten

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

Installation von Perl

Bereiten sie Perl zum kompilieren vor:

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

Wenn sie mehr Kontrolle über die Art haben möchten wie Perl sich selbst zum installieren konfiguriert, können sie stattdessen das interaktive **`Configure`** Skript benutzen. Wenn sie mit den (sinnvollen) Standardwerten zufrieden sind die Perl automatisch erkennt, dann benutzen sie einfach das obige Kommando.

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, müssen sie erst eine Basisversion der `/etc/hosts` Datei 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
```

Installieren von Texinfo−4.6

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  17 MB
```

Inhalt von Texinfo

Texinfo enthält Programme zum lesen, schreiben und konvertieren von Info Dokumenten (System Dokumentation).

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

Texinfo Installationsabhängigkeiten

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

Installation von Texinfo

Bereiten sie Texinfo zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Installieren sie das Paket:

```
make install
```

Optional können sie die Komponenten installieren die zu einer TeX Installation dazugehören:

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

Die Bedeutung des make Parameters:

- **`TEXMF=/usr/share/texmf`**: Die TEXMF makefile Variable enthält den Standort ihres TeX Verzeichnisbaumes, falls sie zum Beispiel planen später ein TeX Paket zu installieren.

Installieren von Autoconf−2.57

```
Geschätzte Kompilierzeit:              2.9 SBU
Ungefähr benötigter Festplattenplatz:  7.7 MB
```

Inhalt von Autoconf

Autoconf erstellt Shell Skripte die Quelltexte automatisch konfigurieren.

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

Autoconf Installationsabhängigkeiten

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
```

Dieses Paket hat eine Test−suite mit der sie überprüfen können ob alles korrekt kompiliert wurde. Wenn sie sie laufen lassen möchten, führen sie dieses Kommando aus:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Automake−1.7.6

```
Geschätzte Kompilierzeit:              5.3 SBU
Ungefähr benötigter Festplattenplatz:  6.8 MB
```

Inhalt von Automake

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

Installierte Programme: acinstall, aclocal, aclocal−1.7, automake, automake−1.7, compile, config.guess, config.sub, depcomp, elisp−comp, install−sh, mdate−sh, missing, mkinstalldirs, py−compile, ylwrap

Automake Installationsabhängigkeiten

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
```

Dieses Paket hat eine Test−suite mit der sie prüfen können ob alles korrekt kompiliert wurde. Benutzen sie dieses Kommando wenn sie sie auszuführen möchten:

```
make check
```

Installieren sie das Paket:

```
make install
```

Und erzeugen sie einen nötigen symbolischen Link:

```
ln -s automake-1.7 /usr/share/automake
```

Installieren von Bash−2.05b

```
Geschätzte Kompilierzeit:              1.2 SBU
Ungefähr benötigter Festplattenplatz:  27 MB
```

Inhalt von Bash

Bash ist die "Bourne Again SHell", ein auf Unix Systemen weit verbreiteter Befehlsinterpreter. Die Bash liest Befehle von der Standard Eingabe (der Tastatur). Ein Anwender gibt etwas ein und die Bash wertet die Eingabe aus. Je nach Eingabe reagiert die Bash entsprechend und führt zum Beispiel ein Programm aus.

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

Bash Installationsabhängigkeiten

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

Installation 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 zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket hat eine Test−suite mit der sie prüfen können ob alles korrekt kompiliert wurde. Wenn sie die Tests durchlaufen möchten, führen sie dieses Kommando aus:

```
make tests
```

Installieren sie das Paket:

```
make install
```

Und starten sie die frisch installierte **bash**:

```
exec /bin/bash −−login +h
```

Installieren von File−4.04

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  6.3 MB
```

Inhalt von File

File ist ein kleines Werkzeug zum identifizieren von Dateitypen.

Installiertes Programm: file

Installierte Bibliotheken: libmagic.[a,so]

File Installationsabhängigkeiten

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 die das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Installieren von Libtool−1.5

```
Geschätzte Kompilierzeit:              1.5 SBU
Ungefähr benötigter Festplattenplatz:  20 MB
```

Inhalt von Libtool

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

Installierte Programme: libtool und libtoolize

Installierte Bibliotheken: libltdl.[a,so].

Libtool Installationsabhängigkeiten

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
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Bzip2−1.0.2

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  3.0 MB
```

Inhalt von Bzip2

Bzip2 ein Block−sortierendes Kompressionsprogramm und erreicht üblicherweise bessere Kompressionsraten als das herkömmliche **`gzip`**.

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

Bzip2 Installationsabhängigkeiten

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

Installation 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
```

Installieren sie es:

```
make install
```

Und installieren sie die ausführbare **bzip2** Datei 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
```

Installieren von Diffutils-2.8.1

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  7.5 MB
```

Inhalt von Diffutils

Die Programme dieses Pakets können die Unterschiede zwischen zwei Dateien oder Verzeichnissen anzeigen. Die häufigste Anwendung ist das erstellen von Software-Patches.

Installierte Programme: cmp, diff, diff3 und sdiff

Diffutils Installationsabhängigkeiten

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

Installation von Diffutils

Bereiten sie Diffutils zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Installieren von Ed−0.2

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  3.1 MB
```

Inhalt von Ed

GNU ed ist ein 8bit−fähiger, POSIX−konformer Editor.

Installierte Programme: ed und red (Link auf ed)

Ed Installationsabhängigkeiten

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 Patch Programm 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 übr 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.

Wenden sie den Patch an:

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

Bereiten sie Ed nun zum kompilieren vor:

```
./configure −−prefix=/usr
```

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Installieren sie das Paket:

```
make install
```

Und verschieben sie die Programme nach `/bin` damit sie auch dann verwendet werden können, wenn die Partition für `/usr` nicht verfügbar sein sollte.

```
mv /usr/bin/{ed,red} /bin
```

Installieren von Kbd−1.08

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  12 MB
```

Inhalt von Kbd

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

Installiete 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

Kbd Installationsabhängigkeiten

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

Installation von Kbd

Standardmässig werden einige von Kbd's Hilfprogrammen (**`setlogcons`**, **`setvesablank`** und **`getunimap`**) nicht installiert. Aktivieren sie erst die Installation dieser Programme:

```
patch −Np1 −i ../kbd−1.08−more−programs.patch
```

Bereiten sie Kbd zum kompilieren vor:

```
./configure
```

Kompilieren sie das Paket:

```
make
```

Und installieren es:

```
make install
```

Installieren von E2fsprogs−1.34

```
Geschätzte Kompilierzeit:              0.6 SBU
Ungefähr benötigter Festplattenplatz:  48.4 MB
```

Inhalt von E2fsprogs

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

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]

E2fsprogs Installationsabhängigkeiten

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 Quellverzeichnisses zu kompilieren:

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

Bereiten sie E2fsprogs zum kompilieren vor:

```
../e2fsprogs-1.34/configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs
```

Die Bedeutung der configure Optionen:

- **`--with-root-prefix=""`**: Bestimmte Programme (so wie z. B. e2fsck) sind absolut essentielle Programme. Wenn zum Beispiel `/usr` nicht gemountet ist müssen diese Programme trotzdem verfügbar sein. Sie gehören in Verzeichnisse wie `/lib` und `/sbin`. Wenn diese Option nicht an E2fsprogs configure Skript übergeben wird, würden die Programme entgegen unserem Willen im `/usr` Verzeichnis installiert werden.
- **`--enable-elf-shlibs`**: Das erzeugt die gemeinsamen Bibliotheken die einige Programme in diesem Paket verwenden.

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Falls sie sie auführen wollen, führen sie dieses Kommando aus:

```
make check
```

Installieren sie das meiste aus dem Paket:

```
make install
```

Und installieren sie auch die gemeinsamen Bibliotheken:

```
make install−libs
```

Installieren von Grep−2.5.1

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  5.8 MB
```

Inhalt von Grep

Grep zeigt alle Zeilen einer Datei an die auf ein bestimmtes Muster passen.

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

Grep Installationsabhängigkeiten

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

Installation von Grep

Bereiten sie Grep zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Grub−0.93

```
Geschätzte Kompilierzeit:               0.2 SBU
Ungefähr benötigter Festplattenplatz:  10 MB
```

Inhalt von Grub

Das Paket Grub enthält den Grub Bootloader.

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

Grub Installationsabhängigkeiten

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

Installation von Grub

Es ist bekannt, das dieses Paket nicht sauber funktioniert wenn die standard Optimierungseinstellungen (inklusive der −march und −mcpu Optionen) verändert wurden. Deshalb sollten sie event. gesetzte Umgebungsvariablen die die Standard Optimierung überschreiben − zum Beispiel CFLAGS und CXXFLAGS − für den Kompiliervorgang von Grub zurücksetzen oder entsprechend abändern.

Beheben sie zuerst ein kompilierproblem mit GCC−3.3.1:

```
patch −Np1 −i ../grub−0.93−gcc33−1.patch
```

Bereiten sie nun Grub zum kompilieren vor:

```
./configure −−prefix=/usr
```

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
mkdir /boot/grub
cp /usr/share/grub/i386−pc/stage{1,2} /boot/grub
```

Ersetzen sie `i386−pc` durch das für ihre Plattform korrekte Verzeichnis.

Das `i386−pc` Verzeichnis enthält auch einige `*stage1_5` Dateien, jeweils für verschiedene Dateisysteme. Schauen sie nach welche zur Verfügung stehen und kopieren sie die notwendigen nach `/boot/grub`. Die meisten werden `e2fs_stage1_5` und/oder `reiserfs_stage1_5` kopieren.

Installieren von Gzip−1.3.5

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  2.6 MB
```

Inhalt von Gzip

Gzip enthält Programme für die Dateikompression und −dekompression mit Hilfe des Lempel−Ziv Algorythmus (LZ77).

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

Gzip Installationsabhängigkeiten

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, das 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 das `/bin` Verzeichnis:

```
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
```

Installieren von Man−1.5m2

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  1.9MB
```

Inhalt von Man

Man ist das Programm zum Seitenweisen anzeigen von Hilfeseiten (man−pages).

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

Man Installationsabhängigkeiten

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 Man vor:

Der erste Patch kommentiert die "MANPATH /usr/man" Zeile in `man.conf` aus. Das verhindert redundante Ergebnisse wenn Programme wie zum Beispiel **`whatis`** verwendet werden:

```
patch −Np1 −i ../man−1.5m2−manpath.patch
```

Der zweite Patch fügt der *PAGER* Variable die *−R* Option hinzu. Dadurch werden Escape Sequenzen korrekt behandelt:

```
patch −Np1 −i ../man−1.5m2−pager.patch
```

Der dritte und letzte 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
```

Bereiten sie Man nun zum kompilieren vor:

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

Die Bedeutung der configure Optionen:

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

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.

Installieren von Make-3.80

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  8.8 MB
```

Inhalt von Make

Make erkennt automatisch, welche Teile eines grossen Programmes erneut kompiliert werden müssen und welche Kommandos dazu nötig sind.

Installiertes Programm: make

Make Installationsabhängigkeiten

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

Installation von Make

Bereiten sie Make zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test-suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Modutils−2.4.25

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  2.9 MB
```

Inhalt von Modutils

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

Installierte Programme: depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (Link auf insmod), kernelversion, ksyms (Link aus insmod), lsmod (Link auf insmod), modinfo, modprobe (Link auf insmod) und rmmod (Link auf insmod)

Modutils Installationsabhängigkeiten

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
```

Und installieren es:

```
make install
```

Installieren von Patch−2.5.4

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  1.9 MB
```

Inhalt von Patch

Patch manipuliert Dateien auf Basis einer Patch−Datei. Eine Patch−Datei ist üblicherweise eine Liste mit Änderungsanweisungen die mit Hilfe des Programms diff erzeugt wurde.

Installiertes Programm: patch

Patch Installationsabhängigkeiten

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

Installation von Patch

Bereiten sie Patch zum kompilieren vor:

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

Auch hier wird die Preprozessor Option **`-D_GNU_SOURCE`** nur auf PowerPC benötigt. Auf anderen Architekturen können sie sie weglassen.

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Installieren von Procinfo-18

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  0.2 MB
```

Inhalt von Procinfo

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

Installierte Programme: lsdev, procinfo und socklist

Procinfo Installationsabhängigkeiten

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.

Und installieren sie das Paket:

```
make install
```

Installieren von Procps−3.1.11

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  6.2 MB
```

Inhalt von Procps

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

Installierte Programme: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w und watch

Installierte Bibliothek: libproc.so

Procps Installationsabhängigkeiten

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

Installation von Procps

Beheben sie ein Problem das **`w`** unter bestimmten Bedingungen zum Absturz bringen könnte:

```
patch -Np1 -i ../procps-3.1.11-locale-fix.patch
```

Kompilieren sie nun Procps:

```
make
```

Installieren sie es:

```
make install
```

Und entfernen sie einen toten symolischen Link auf eine Bibliothek:

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

Installieren von Psmisc−21.3

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  2.2 MB
```

Inhalt von Psmisc

Das Paket Psmisc enthält drei Programme zur Verwaltung des `/proc` Verzeichnisses.

Installierte Programme: fuser, killall und pstree

Psmisc Installationsabhängigkeiten

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 `/lib`, 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 gemountet ist.

Kompilieren sie das Paket:

```
make
```

Und installieren sie es:

```
make install
```

Standardmässig wird Psmisc's **`pidof`** Programm nicht mit 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 Installtion von Psmisc durch erstellen dieses Links komplettieren:

```
ln -s killall /bin/pidof
```

Installieren von Shadow−4.0.3

```
Geschätzte Kompilierzeit:              0.4 SBU
Ungefähr benötigter Festplattenplatz:  11 MB
```

Inhalt von Shadow

Das Shadow Paket wurde zur verstärkung der Sicherheit von System Passwörtern eingeführt.

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

Shadow Installationsabhängigkeiten

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

Installation von Shadow

Die Programme **`login`**, **`getty`** und **`init`** (ein einige andere) pflegen eine Reihe Logdateien in denen sie aufzeichnen wer in das System eingloggt ist und wer eingeloggt war. Diese Programme erzeugen die Logdateien jedoch nicht wenn sie nicht existieren, wenn sie also diese Art Protokollierung möchten, müssen sie die Dateien selber anlegen. Das Shadow Paket muss diese Dateien an der richtigen Stelle finden, deshalb erstellen wir sie nun mit den korrekten Rechten:

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

Die Datei `/var/run/utmp` listet die gerade angemeldeten Benutzer auf, `/var/log/wtmp` wer eingloggt *war* und wann. `/var/log/lastlog` speichert für jeden Benutzer wann er das letzte Mal angemeldet war und `/var/log/btmp` listet die fehlgeschlagenen Login Versuche auf.

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, nimm das Paket an, das es nach `/bin/passwd` gehört, installiert es dann aber nach `/usr/bin/passwd`. Das führt zu dem Fehler das `/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
```

Die aktuelle Shadow suite hat ein Problem das dazu führt das **`newgrp`** fehlschlägt. Der folgende Patch (der auch in Shadow's CVS Code vorhanden ist) behebt dieses Problem:

```
patch −Np1 −i ../shadow−4.0.3−newgrp−fix.patch
```

Bereiten sie Shadow nun zum kompilieren vor:

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

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 Methode auf MD5 ändern, welche theoretisch sicherer ist als die standard "crypt" Methode, und ausserdem erlaubt sie Passwörter mit mehr als 8 Zeichen. Ausserdem müssen wir den alten Ort für die Benutzermailboxen `/var/spool/mail` nach `/var/mail` ändern. Das erledigen wir indem wir die Konfigurationsdatei beim kopieren an die richtige Stelle gleich ändern:

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

> **Anmerkung:** Seien sie besonders vorsichtig wenn sie das obige abschreiben. Es ist sicherer wenn sie es kopieren und einfügen anstatt es alles abzuschreiben.

Laut der Man-page von **`vipw`** sollte auch ein **`vigr`** Programm existieren. Da das Installationsprogramm dieses Programm nicht erzeugt, erstellen wir den symbolischen Link selber:

```
ln -s vipw /usr/sbin/vigr
```

Weil der symbolische Link `/bin/vipw` redundant ist (und zusätlich auch noch ins Leere zeigt), entfernen sie ihn:

```
rm /bin/vipw
```

Verschieben sie nun das Programm **`sg`** an die richtige Stelle:

```
mv /bin/sg /usr/bin
```

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
```

Coreutils hat bereits ein **groups** Programm 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 im entpackten Shadow Verzeichnisbaum. Eines gilt es allerdings zu beachten: Programme die Passwörter überprüfen müssen (z. B. xdm, ftp und pop3 Server) müssen shadow−konform sein, sdas heisst sie müssen mit shadow Passwörtern umgehen können.

Um shadow Passwörter zu aktivieren, benutzen sie das folgende Kommando:

```
/usr/sbin/pwconv
```

Und um shadow Gruppenpasswörter zu aktivieren, benutzen sie das folgende Kommando:

```
/usr/sbin/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.

Installieren von Sysklogd−1.4.1

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  0.5 MB
```

Inhalt von Sysklogd

Die in Sysklogd enthaltenen Programme dienen zum aufzeichnen von System Logs, zum Beispiel die des Kernels.

Installierte Programme: klogd und syslogd

Sysklogd Installationsabhängigkeiten

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

Installation von Sysklogd

Kompilieren sie Sysklogd:

```
make
```

Und installieren 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
```

Installieren von Sysvinit−2.85

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  0.9 MB
```

Inhalt von Sysvinit

Das Sysvinit Paket enhält Programme mit denen sie das starten, ausführen und beenden aller anderen Programme kontrollieren können.

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

Sysvinit Installationsabhängigkeiten

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

Installation von Sysvinit

Wenn Runlevel gewechselt werden (zum Beispiel beim herunterfahren des Systems), sendet init die Signale TERM und KILL and 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 allen laufenden Programmen sendet. Um diese Verwirrung zu vermeiden können sie die Datei init.c modifizieren so das es sich so liest: "Sending processes started by init the TERM signal".

Bearbeiten sie die Nachricht:

```
cp src/init.c{,.backup}
sed 's/Sending processes/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
```

Installieren von Tar−1.13.25

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  10 MB
```

Inhalt von Tar

Tar ist ein Programm zum speichern und extrahieren von Dateien aus sog. Tar−Archiven.

Installierte Programme: rmt und tar

Tar Installationsabhängigkeiten

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

Installation von Tar

Bereiten sie Tar zum kompilieren vor:

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

Kompilieren sie das Paket:

```
make
```

Dieses Paket enthält eine Test−suite um zu prüfen ob alles korrekt kompiliert wurde. Wenn sie sie ausführen möchten, erledigt dies das folgende Kommando für sie:

```
make check
```

Und installieren sie das Paket:

```
make install
```

Installieren von Util−linux−2.12

```
Geschätzte Kompilierzeit:              0.2 SBU
Ungefähr benötigter Festplattenplatz:  16 MB
```

Inhalt von Util−linux

Util−linux enthält verschiedene Werkzeuge. Einige der etwas bekannteren Programme werden z. B. zum mounten, entmounten, formatieren, partitionieren und verwalten von Festplatten, öffnen von tty Ports und zum auslesen von Kernel Meldungen genutzt.

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, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, 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, test.bash, test.tcsh, tunelp, ul, umount, vidmode (Link auf rdev), whereis und write

Util−linux Installationsabhängigkeiten

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 `/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
```

Installatin von Util−linux

Bereiten sie Util−linux zum kompilieren vor:

```
./configure
```

Kompilieren sie das Paket:

```
make HAVE_SLN=yes
```

Die Bedeutung des make Parameter:

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

Und installieren sie das Paket:

```
make HAVE_SLN=yes install
```

Installieren von GCC−2.95.3

```
Geschätzte Kompilierzeit:              1.5 SBU
Ungefähr benötigter Festplattenplatz:  130 MB
```

Installation von GCC

Es ist bekannt, das dieses Paket nicht sauber funktioniert wenn die standard Optimierungseinstellungen (inklusive der −march und −mcpu Optionen) verändert wurden. Deshalb sollten sie event. gesetzte Umgebungsvariablen die die Standard Optimierung überschreiben − zum Beispiel CFLAGS und CXXFLAGS − für den Kompiliervorgang von GCC zurücksetzen oder entsprechend abändern.

Dies ist eine ältere Version von GCC die wir nur installieren um damit den Linux Kernel in Kapitel 8 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 diese Version hier 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 Quellverzeichnis sondern in einem dafür dedizierten Kompilierverzeichnis 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
```

Ein neues chroot Kommando

Wenn sie von nun an die chroot Umgebung verlassen und wieder eintreten 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, das wir keine Programme mehr aus dem `/tools` Verzeichnis benötigen. Trotzdem sollten wir das `/tools` Verzeichnis noch nicht löschen; wir brauchen es noch bis zum Ende des Buches.

Installieren von LFS Bootscripts-1.12

```
Geschätzte Kompilierzeit:              0.1 SBU
Ungefähr benötigter Festplattenplatz:  0.3 MB
```

Inhalt der LFS-Bootskripte

Die LFS-Bootskripte enthalten Shell Skripte im SysV init Stil. Diese Skripte erfüllen verschiedene Aufgaben wie zum Beispiel Dateisystemprüfungen beim Systemstart, das laden von Tastaturmappings, Netzwerkeinrichtung oder das Beenden von Prozessen beim herunterfahren des Systems.

Installierte Skripte: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, swap, sysklogd und template

LFS-Bootscripts Installationsabhängigkeiten

Bzip2 ist abhängig von: Bash, Coreutils.

Installation der LFS-Bootscripts

Wir benutzen Init Skripte im SysV Stil. Wir haben diesen Stil ausgewählt 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 BSD-Stil Init Skripte oder etwas ganz anderes entschieden haben können sie das nachfolgende Kapitel überspringen und direkt weitermachen bei Kapitel 8.

Installieren sie die Boot Skripte:

```
cp -a rc.d sysconfig /etc
```

Geben sie *root* die Besitzrechte auf die Skripte:

```
chown -R root:root /etc/rc.d /etc/sysconfig
```

Konfigurieren der Systemkomponenten

Jetzt wo alle Software installiert ist müssen wir nur noch ein paar Konfigurationen vornehmen.

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 standard Tastaturlayout zu ändern, erstellen sie mit dem folgenden Kommando den symbolischen Link `/usr/share/kbd/keymaps/defkeymap.map.gz`:

```
ln -s pfad/zum/tastaturlayout /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 das Tastaturlayout zu setzen, ist die keymap in den Kernel einzukompilieren. Das stellt sicher das 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.

Führen sie das folgende Kommando aus wenn sie die jetzige standard 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 > \
    /usr/src/linux-2.4.22/drivers/char/defkeymap.c
```

Setzen des root Passworts

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

```
passwd root
```

Kapitel 7. Aufsetzen der System Boot Skripte

Einführung

Dieses Kapitel setzt die Boot Skripte auf, die sie in Kapitel 6 installiert haben. Die meisten der Skripte funktionieren ohne Anpassungen, aber ein paar benötigen eine Grundkonfiguration weil sie zum Beispiel Hardwareabhängig sind.

Wie funktioniert der Bootvorgang mit diesen Skripten?

Linux benutzt eine spezielle Booteinrichtung mit dem Namen SysVinit. Es basiert auf dem Konzept der *Runlevel*. Dieses kann von System zu System sehr 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 Verzeichnisse 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 beginnen 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 Verzeichnissen rc0.d und rc6.d starten keine Dienste. Sie werden

stattdessen mit dem stop Parameter aufgerufen um etwas zu beenden. Die Logik dahinter ist, das 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 mithilfe 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* in eine *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 wir 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 am Ende von Kapitel 6 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, das 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 *−m 0* Option. Diese Option schaltet die periodische Marke ab, die syslogd standardmässig 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 Aufgabe des localnet Skript ist das setzen des System Hostnamen. Das 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 durch den Namen für ihren Computer ersetzt werden. Geben sie hier 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
```

Sie sollten sicherstellen, das 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.

Auch wenn sie keine Netzwerkkarte verwenden brauchen sie trotzdem 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>.mydomain.com <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.localdomain 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 Verzeichnissen 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 konfiguieren. 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 sodass 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. Dieses Verzeichnis soll Dateien der Form ifconfig.xyz enthalten, wobei xyz der Name eines Netzwerkgerätes ist (wie zum Beispiel eth0 oder eth0:1)

Wenn sie das Verzeichnis /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 mit dem folgenden Inhalt. Das folgende Kommando erzeugt eine Beispieldatei ifconfig.eth0:

```
cat > /etc/sysconfig/network−devices/ifconfig.eth0 << "EOF"
ONBOOT=yes
```

```
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.

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 Boot loader 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

# filesystem  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 ihre System ersetzen -- zum Beispiel `hda2`, `hda5` und `reiserfs`. Die Details zu den sechs Feldern in dieser Tabelle finde 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, das zur Zeit nur sehr wenig Software POSIX shared memory verwendet. Daher können sie den `/dev/shm` Mountpunkt als optional betrachten. Mehr Informationen dazu finden sie in `Documentation/filesystems/tmpfs.txt` im Quellverzeichnisbaum 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.

Installieren von Linux−2.4.22

```
Geschätzte Kompilierzeit:              All default options: 4.20 SBU
Ungefähr benötigter Festplattenplatz:  All default options: 181 MB
```

Inhalt von Linux

Der Linux Kernel ist der Kern eines jeden Linux Systems. Er ist sozusagen der Herzschlag von Linux. Wenn der Computer eingeschaltet wird und ein Linux System startet, dann ist der Kernel das erste Stück Software das gestartet wird. Der Kernel initialisiert die Geräte und Hardware Komponenten: serielle Schnittstellen, parallele Schnittstellen, Soundkarten, Netzwerkkarten, IDE und SCSI Controller und vieles mehr. Zusammenfassend kann man sagen, der Kernel stellt dem System die Hardware zur Verfügung, so das die Software damit laufen kann.

Installierte Dateien: Der Kernel und die Kernel Header

Linux Installationsabhängigkeiten

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 Quellenverzeichnis nach alternativen Methoden.

Bereiten sie den Kompiliervorgang mit dem folgenden Kommando vor:

```
make mrproper
```

Damit wird sichergestellt, das der Kernel absolut sauber ist. Das Kernel Team empfiehlt, dieses Kommando vor *jeder* Kernel Installation auszuführen. Sie sollten sich nicht blindlings darauf verlassen, das der Kernel−Verzeichnisbaum nach dem entpacken sauber ist.

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 Kernel Konfiguration überspringen und einfach die Kernel Konfigurationsdatei, `.config`, von ihrem Host−system nach `linux−2.4.22` 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 Kernel Konfiguration im allgemeinen finden sie in der Kernel Dokumentation, sie finden sie im Verzeichnis `linux-2.4.22/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
```

Weil ohne Dokumentation nichts komplett ist, erzeugen sie die Man-pages die mit dem Kernel kommen:

```
make mandocs
```

Und installieren sie diese:

```
cp -a Documentation/man /usr/share/man/man9
```

Kernel kompilieren ist nun abgeschlossen, aber einige der erzeugten Dateien befinden sich noch im Quellverzeichnis. Um die Installation abzuschliessen müssen zwei Dateien in das `/boot` Verzeichnis 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
```

Das LFS System bootfähig machen

Ihr frisches LFS System ist beinahe komplett. Eines der letzten Dinge ist, sicherzustellen das es booten kann. Die untenstehende Anleitung gilt nur auf Computern mit IA−32 Architektur, dazu gehören alle handelsüblichen PCs. Informationen zum "boot loading" auf anderen Architekturen finden sie and den üblichen Dokumentationsquellen zu diesen Architekturen.

Das booten kann ein komplexes Thema sein. Hier erstmal ein paar warnende Worte. Sie sollten mit ihrem jetzigen Boot loader 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 Boot loader als Vorbereitung für diesen Schritt installiert. In dieser Prozedur müssen ein paar Grub Dateien an spezielle Orte auf der Festplatte kopiert werden. Bevor wir dazu kommen empfehlen wir, das 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 legen sie sie an einem sicheren Ort ab. 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, das zum Beispiel die Partition `hda1` für Grub (hd0,0) ist, und `hdb2` ist (hd1,1). Anders als Linux betrachet 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 das 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 Boot loader. Wenn das nicht ist was sie wollen führen sie das Kommando nicht aus. Zum Beispiel wenn sie einen Boot loader von einem Dritthersteller benutzen möchten um ihren MBR zu verwalten (Master Boot Record). In dem Fall würde es Sinn machen, Grub in den "Boot Sektor" ihrer LFS Partition zu installieren, das Kommando würde dann lauten: **`setup (hd0,3)`**.

Sagen sie Grub nun das er sich in den MBR von `hda` installieren soll:

```
setup (hd0)
```

Wenn alles in Ordnung ist wird Grub nun berichten das er seine Dateien in `/boot/grub` findet. Das ist alles soweit:

```
quit
```

Nun müssen wir die Datei "menu.lst" erstellen, welche das Grub Boot Menü 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 5.0
root (hd0,3)
kernel /boot/lfskernel root=/dev/hda4 ro
EOF
```

Vielleicht möchten sie einen weiteren Eintrag für ihr Host−System vornehmen. Dieser könnte so aussehen:

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

Jetzt könnte ein guter Zeitpunkt sein alle debug Symbole aus den Binärdateien auf ihrem LFS System zu entfernen. Wenn sie kein Programmierer sind und nicht planen die Software zu debuggen, dann wird es sie erfreuen zu hören das sie einige Megabytes an Festplattenspeicher zurückgewinnen können indem sie debugging Symbole entfernen. Dadurch haben sie keine Nachteile, ausser das sie diese Programme dann nicht mehr vollständig debuggen können. Wenn sie nicht wissen was debuggen ist, ist das für sie höchstwahrscheinlich nicht von Bedeutung.

Hinweis: 98% der Leute die das untenstehende Kommando ausführen haben keine Probleme. Aber machen sie trotzdem ein Backup ihres Systems bevor sie den Befehl ausführen. Es besteht immer noch eine kleine Chance das es schiefgeht und ihr System unbenutzbar macht (zum Beispiel durch zerstören der Kernel Module und der dynamischen & gemeinsamen Bibliotheken). Soetwas wird meist durch Tippfehler anstatt durch das Kommando ansich verursacht.

Die −−strip−debug Option die wir benutzen ist unter normalen Umständen völlig harmlos. Sie entfernt keine wichtigen Dinge von den Dateien. Es ist sogar relativ sicher, −−strip−all auf normalen Programmen anzuwenden (verwenden sie das nicht auf Bibliotheken − sie werden dadurch zerstört), aber es ist nicht ganz so sicher und der dadurch eingesparte Platz ist das Risiko nicht wert. Schlagen sie in der Man−page zu strip nach um die weiteren möglichen Optionen kennenzulernen, generell sollte man strip nicht auf Bibliotheken anwenden (ausser mit der −−strip−debup Option).

Wenn sie strip ausführen möchten, ist besondere Vorsicht geboten damit sie strip nicht auf Programme anwenden die gerate ausgeführt werden −− inklusive der bash shell. Daher müssen sie die chroot Umgebung verlassen und mit einem modifizierten Kommando erneut betreten:

```
logout
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:

```
/tools/bin/find /{,usr/,usr/local/}{bin,sbin,lib} -type f \
   -exec /tools/bin/strip --strip-debug '{}' ';'
```

Es werden einige Dateien gemeldet deren Format nicht erkannt wurde. Die meisten dieser Dateien sind Skripte und keine Binärdateien. Die Warnungen können einfach ignoriert werden.

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 5.0 > /etc/lfs-release
```

Werden sie gezählt

Möchten sie nun wo sie das Buch durchgearbeitet haben als LFS Benutzer gezählt werden? Dann gehen sie zu http://linuxfromscratch.org/cgi-bin/lfscounter.cgi und registrieren sie sich als LFS Benutzer indem sie ihren Namen und die Versionsnummer ihre 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, die chroot Umgebung zu verlassen und den Computer neu zu starten. Bevor wir die chroot Umgebung verlassen sollten wir noch die eingehängten virtuellen Dateisystem entmounten:

```
umount /proc
umount /dev/pts
```

Verlassen sie die chroot Umgebung:

```
logout
```

Desweiteren können sie nun nach der Installation das `/tools` Verzeichnis löschen. Da das auch die temporären Kopien von Tcl, Expect und DejaGnu löscht, die zum ausführen der toolchain test installiert wurden, müssen sie diese neu installieren wenn sie sie weiter verwenden möchten.

Vielleicht möchten sie auch den Inhalt von `/sources` nach `/usr/src/packages` verschieben (oder sie vielleicht auf CD brennen und dann löschen).

Bevor sie neu starten lassen sie uns noch die LFS Partition selbst aushängen:

```
umount $LFS
```

Wenn sie sich für mehrere Partitionen entschieden haben, müssen sie die anderen Partitionen aushängen bevor sie wie folgt $LFS entmounten:

```
umount $LFS/usr
umount $LFS/home
umount $LFS
```

Und jetzt können sie ihren Computer neu starten indem sie eingeben:

```
/sbin/shutdown -r now
```

Unter der Annahme das der Grub Boot loader wie vorgeschlagen installiert wurde, sollte das standard Bootmenü *LFS 5.0* 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 das sie das LFS Buch gelesen haben. Wir hoffen, das sie das Buch nützlich fanden und das es seine Zeit wert war.

Jetzt wo sie mit der Installation von LFS fertig sind, fragen sie sich vielleich "Was nun?". Um diese Frage zu beantworten haben wir eine Reihe von Links 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.
- Mailing Listen

  Es gibt einige Mailing Listen die sie abonnieren können wenn sie mal Hilfe benötigen. Schauen sie für weitere Informationen unter Kapitel 1 − Mailinglisten.
- Das Linux Documentation Project

  Das Ziel des Linux Documentation Project ist es, in allen Fragen zu Linux zusammenzuarbeiten. Das LDP verfügt über jede Menge an HOWTOs, Anleitungen und Man−pages. Sie finden es unter http://www.tldp.org/.

IV. Teil IV − Anhänge

Inhaltsverzeichnis

Anhang A. Paketbeschreibungen und −abhängigkeiten

Einführung

Dieser Anhang beschreibt zu jedem installierten Paket die folgenden Details:

- die offizielle Download Adresse für das Paket,
- was das Paket beinhaltet,
- für was die Programme in dem Paket zuständig sind,
- die Voraussetzungen um das Paket kompilieren zu können.

Die meisten Informationen über die Pakete (vor allem die Beschreibungen) wurden den man−pages entnommen. Wir geben nur die Eckpunkte mit an, nicht die gesamte man−page, so das man einen Überblick erhält was jedes Programm macht. Die restlichen Details entnehmen sie bitte den man−pages selbst.

Manche Pakete werden ausführlicher dokumentiert als andere. Das liegt daran das wir einfach über bestimmte Pakete mehr Informationen haben als über andere. Wenn sie meinen, das zu den nachfolgenden Beschreibungen etwas wichtiges hinzugefügt werden sollte, zögern sie nicht eine Email an die Mailing Liste zu schreiben. Die Paketbeschreibungen sollen für jedes Paket so ausführlich wie möglich sein, aber das schaffen wir nicht ohne Hilfe.

Bitte beachten sie, das zur Zeit nur beschrieben wird was ein Paket macht und nicht warum es installiert wird. Möglicherweise werden wir die Liste später diesbezüglich erweitern.

Auch die Abhängigkeiten aller installierten Pakete sind aufgeführt. Die Liste gibt wieder, welche Programme benötigt werden um ein anderes Programm installieren zu können.

Es handelt sich hier nicht um die Laufzeit−Abhängigkeiten, das bedeutet sie erfahren nicht welche Programme sie zum ausführen des Pakets brauchen, sondern nur welche Programme zum erfolgreichen kompilieren benötigt werden.

Die Liste der Abhängigkeiten kann von Zeit zu Zeit veraltet sein, vor allem in Bezug auf sehr aktuelle Paket Versionen. Das prüfen der Abhängigkeiten ist viel Arbeit, so das wir manchmal zeitlich dem Paket Update ein wenig hinterher sind. Doch die meisten kleineren Paket Updates beeinflussen die Abhängigkeiten nicht, so das die Liste zumindest in den meisten Fällen aktuell sein sollte. Beim Update auf die nächste Hauptversion stellen wir jeweils sicher, das die Abhängigkeiten überprüft und ggf. korrigiert werden.

Autoconf

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Autoconf−2.57* in Kapitel 6.

Offizielle Download Adresse

Autoconf (2.57):
ftp://ftp.gnu.org/gnu/autoconf/

Inhalt von Autoconf

Autoconf erstellt Shell Skripte die Quelltexte automatisch konfigurieren.

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

Kurze Beschreibungen

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 unabhängig 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 Macro Prozessor.

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

autoscan kann beim erzeugen einer `configure.in` Datei für ein Software Paket behilflich sein. Es untersucht die Quelldateien in einem Verzeichnis 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 Preprozessor 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.

Autoconf Installationsabhängigkeiten

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

Automake

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Automake−1.7.6* in Kapitel 6.

Offizielle Download Adresse

Automake (1.7.6):
ftp://ftp.gnu.org/gnu/automake/

Inhalt von Automake

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

Installierte Programme: acinstall, aclocal, aclocal−1.7, automake, automake−1.7, compile, config.guess, config.sub, depcomp, elisp−comp, install−sh, mdate−sh, missing, mkinstalldirs, py−compile, ylwrap

Kurze Beschreibungen

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 Hauptverzeichnis 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 das versucht, kanonische Triplets 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 Code.

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

mdate−sh ist ein Skript, welches die Veränderungszeigt einer Datei oder eines Verzeichnisses ausgibt.

missing ist ein Skript für fehlende GNU Programme während der Installation.

mkinstalldirs ist ein Skript zum erzeugen eines Verzeichnisbaumes.

py−compile kompiliert ein Python Programm.

ylwrap ist ein Wrapper für lex und yacc.

Automake Installationsabhängigkeiten

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

Bash

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Bash−2.05b* in Kapitel 6.

Offizielle Download Adresse

Bash (2.05b):
ftp://ftp.gnu.org/gnu/bash/

Bash Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/bash−2.05b−2.patch

Inhalt von Bash

Bash ist die "Bourne Again SHell", ein auf Unix Systemen weit verbreiteter Befehlsinterpreter. Die Bash liest Befehle von der Standard Eingabe (der Tastatur). Ein Anwender gibt etwas ein und die Bash wertet die Eingabe aus. Je nach Eingabe reagiert die Bash entsprechend und führt zum Beispiel ein Programm aus.

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

Kurze Beschreibungen

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 bash Programm. Wenn die bash als sh aufgerufen wird, versucht sie das Verhalten der historischen Versionen von sh so gut wie möglich zu simulieren und bleibt dabei trotzdem POSIX Standardkonform.

Bash Installationsabhängigkeiten

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

Binutils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Binutils−2.14* in Kapitel 6.

Offizielle Download Adresse

Binutils (2.14):
ftp://ftp.gnu.org/gnu/binutils/

Inhalt von Binutils

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

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

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

Kurze Beschreibungen

addr2line übersetzt Programmadressen in Dateinamen und Zeilennummern. Mithilfe der Adresse und dem Namen der ausführbaren Datei benutzt es die debugging Informationen in der Datei um die korrespondierende Quelldatei und Zeilennummer zu der Adresse herauszufinden.

ar erzeugt, bearbeitet und extrahiert von Archiven. Ein Archiv ist eine einzige Datei die eine Sammlung anderer Dateien enthält. Die Struktur des Archivs macht es möglich, einzelne originale Dateien aus dem Archiv zu erhalten (auch Member eines Archivs genannt).

as ist ein Assembler. Er assembliert die Ausgabe von GCC zu Objektdateien.

c++filt wird vom Linker benutzt um verstümmelte C++ und Java Symbole zu reparieren und so Konflikte mit überladenen Funktionen zu verhindern.

gprof zeigt profiling Daten zu Aufrufdiagrammen (call graphs) an.

ld ist ein Linker. Er fügt eine vielzahl von Objekten und Archiven zu einer einzigen Datei zusammen, replaziert ihre Daten und fügt Symbol referenzen zusammen.

nm listet die vorhandenen Symbole in einer angegebenen Objektdatei auf.

objcopy wird verwendet um einen Objekttyp in einen anderen umzuwandeln.

objdump zeigt ausgewählte Informationen zu einer Objektdatei an. Die Informationen sind hauptsächlich für Programmierer nützlich, die an den Kompilierwerkzeugen arbeiten.

ranlib erzeugt einen Index vom Inhalt eines Archivs und speichert ihn in dem Archiv. Der Index enthält eine Liste aller Symbole von replazierbaren Objektdateien im Archiv.

readelf zeigt Informationen über elf−Binärdateien an.

size listet Bereichsgrössen −− und die Summe −− für angegebene Objektdateien auf.

strings gibt zu jeder angegebenen Datei Zeichenketten von druckbaren Zeichen aus, wobei die Sequenzen (als Vorgabe) mindestens 4 Zeichen lang sein müssen. Für Objektdateien gibt es standardmässig nur die Zeichenketten aus den Initialisierungs− und Ladebereichen aus. Bei anderen Dateitypen durchsucht es jeweils die gesamte Datei.

strip entfernt Symbole aus Objektdateien.

libiberty enthält Routinen die von verschiedenen GNU Programmen genutzt werden, z. B. getopt, obstack, strerror, strtol und strtoul.

libbfd ist die Bibliothek für Binärdateibeschreibungen.

libopcodes ist eine Bibliothek die sog. Opcodes behandelt. Sie wird zum erstellen von Werkzeugen wie objdump verwendet. Opcodes sind die "lesbaren" Versionen der Prozessorinstruktionen.

Binutils Installationsabhängigkeiten

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

Bison

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Bison−1.875* in Kapitel 6.

Offizielle Download Adresse

Bison (1.875):
ftp://ftp.gnu.org/gnu/bison/

Bison Attribut Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/bison−1.875−attribute.patch

Inhalt von Bison

Bison ist ein Programm zum erstellen von Analysatoren, ein Ersatz für yacc. Bison erstellt ein Programm welches die Struktur einer Textdatei analysiert.

Installierte Programme: bison und yacc

Installierte Bibliotheken: liby.a

Kurze Beschreibungen

bison erzeugt aus einer Reihe von Regeln ein Programm zum analysieren der Struktur von Textdateien. Bison ist ein Ersatz zu yacc (Yet Another Compiler Compiler).

yacc 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 Hauptfunktionen enthält. Diese Bibliothek ist normalerweise nicht sehr hilfreiche, aber sie wird von POSIX vorausgesetzt.

Bison Installationsabhängigkeiten

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

Bzip2

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Bzip2−1.0.2* in Kapitel 6.

Offizielle Download Adresse

Bzip2 (1.0.2):
http://sources.redhat.com/bzip2/

Inhalt von Bzip2

Bzip2 ein Block−sortierendes Kompressionsprogramm und erreicht üblicherweise bessere Kompressionsraten als das herkömmliche **`gzip`**.

Installierte Programme: bunzip2 (Link auf bzip2), bzcat (Link auf bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless und bzmore

Installierte Bibliotheken: libbz2.a, libbz2.so (Link auf libbz2.so.1.0), libbz2.so.1.0 (Link auf libbz2.so.1.0.2) und libbz2.so.1.0.2

Kurze Beschreibungen

bunzip2 dekomprimiert bzip2 Dateien.

bzcat dekomprimiert zur Standard Ausgabe.

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.

Bzip2 Installationsabhängigkeiten

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

Coreutils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Coreutils−5.0* in Kapitel 6.

Offizielle Download Adresse

Coreutils (5.0):
ftp://ftp.gnu.org/gnu/coreutils/

Coreutils Hostname Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−hostname−2.patch

Coreutils Uname Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−uname.patch

Inhalt von Coreutils

Das Paket Coreutils enthält eine große Anzahl von Shell Werkzeugen.

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, kill, 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 Beschreibungen

basename entfernt Pfade und ein angegebenes Suffix von einem Dateinamen.

cat fügt Dateien an die Standard Ausgabe an.

chgrp ändert die Gruppenzugehörigkeit jeder angegebenen Datei zu der angegebenen Gruppe. Die Gruppe kann entweder ein Name oder eine numerische ID sein.

chmod ändert die Rechte auf jeder angegebenen Datei auf den angegebenen Modus. Der Modus kann entweder symbolische oder als Oktale Nummer angegeben werden.

chown ändern Benutzer und/oder Gruppenzugehörigkeit jeder angegebenen Datei auf das angegebene Benutzer:Gruppe Paar.

chroot führt ein Kommando mit dem angegebenen Verzeichnis als / Verzeichnis aus. Das Kommando kann eine interaktive Shell sein. Auf den meisten Systemen kann nur *root* dies tun.

cksum gibt die CRC Prüfsumme und anzahl der Bytes einer Datei aus.

comm vergleicht zwei sortierte Dateien und gibt in drei Spalten die gemeinsamen und die einmaligen Vorkommnisse von Zeilen aus.

cp kopiert Dateien.

csplit teil eine Datei in mehrere neue Dateien auf und schneidet entsprechend eines Musters oder anhand von Zeilennummern. Gibt die Byte−grösse jeder neuen Datei aus.

cut gibt Teile aus einer Zeile aus und wählt die auszugebenden Teil anhand von Feldern oder Positionsangaben aus.

date zeigt die aktuelle Zeit im vorgegebenen Format an oder setzt sie neu.

dd kopiert eine Datei mit der angegebenen Blockgröße und Anzahl während es optional diverse Konvertierungen durchführt.

df berichtet über den verfügbaren (und benutzten) Festplattenplatz auf allen eingehängten Dateisystem oder nur auf den Dateisystemen die bestimmte Dateien enthalten.

dir ist das selbe wie ls.

dircolors gibt die Kommandos aus um die LS_COLOR Umgebungsvariable zu setzen. Mit deren Hilfe wird das Farbschema von ls verändert.

dirname entfernt den nicht−Verzeichnis Suffix von einem Dateinamen.

du berichtet die Menge Festplattenspeicher die vom aktuellen Verzeichnis oder von allen Unterverzeichnissen verbraucht wird.

echo gibt den angegebenen String aus.

env führt ein Kommando in einer geänderten Umgebung aus.

expand konvertiert Tabulatoren in Leerzeichen.

expr wertet Ausdrücke aus.

factor gibt den Primfaktor aller angegebenen integer Werte aus.

false tut nichts, ist immer nicht−erfolgreich. Es beendet immer mit einem Status Code der auf einen Fehler hindeutet.

fmt formatiert Absätze in einer Datei neu.

fold bricht Zeilen in Dateien um.

groups gibt die Gruppenzugehörigkeit des Benutzers aus.

head gibt die ersten zehn (oder die angegebene Anzahl) Zeilen einer Datei aus.

hostid gibt den numerischen Bezeichner des Hosts aus (hexadezimal).

hostname gibt den Hostnamen aus bzw. setzt ihn.

id gibt die effektive Benutzer ID, Gruppen ID, und Gruppenzugehörigkeit des aktuellen Benutzers oder eines angegebenen Benutzers aus.

install kopiert Dateien und setzt währenddessen ihre Berechtigungen, und wenn möglich ihren Benutzer und Gruppe.

join fügt Zeilen aus zwei Dateien zusammen, die identische join Felder haben.

kill beendet den angegebenen Prozess.

link erzeugt einen harten Link von einer Datei auf eine andere.

ln erzeugt harte oder symbolische Links zwischen Dateien.

logname gibt den login Namen des aktuellen Benutzers aus.

ls listet den Inhalt eines Verzeichnisses auf. Standardmässig sortiert es alphanumerisch.

md5sum prüft MD5 Prüfsummen oder gibt sie aus.

mkdir erzeugt Verzeichnisse mit dem angegebenen Namen.

mkfifo erzeugt FIFO's mit den angegebenen Namen.

mknod erzeugt Gerätedateien mit den angegebenen Namen. Eine Gerätedatei ist eine spezielle Zeichen− oder Blockdatei oder ein FIFO.

mv verschiebt oder bennent Dateien und Verzeichnisse um.

nice führt ein Programm mit veränderter Priorität aus.

nl numeriert die Zeilen in einer Datei.

nohup führt ein Kommando so aus, das es von Hangups unabhängig ist, die Ausgabe wird in eine Logdatei umgeleitet.

od gibt Dateien oktal oder in anderen Formaten aus.

paste fügt Dateien zusammen, verkettet sequentiell zusammengehörige Zeilen durch Tabulatoren getrennt nebeneinander.

pathchk prüft, ob Dateinamen gültig oder portabel sind..

pinky ist ein leichtgewichtiges finger Programm. Es gibt ein paar Informationen zu einem Benutzer aus.

pr bereitet Dateien zum Seiten− oder Spaltenweisen drucken vor.

printenv gibt die Umgebung aus.

printf gibt die angegebenen Argumente in einem bestimmten Format aus − 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 Verzeichnisses aus.

readlink gibt den Wert eines symbolischen Links aus.

rm löscht Dateien oder Verzeichnisse.

rmdir löscht Verzeichnisses, wenn sie leer sind.

seq gibt eine Reihe von Zahlen in einem bestimmten Bereich 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 teil eine Datei in Stücke, nach grösse oder nach Zeilennummern.

stty setzt oder zeigt Terminal Einstellungen an.

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

sum gibt Prüfsumme und Blockanzahl einer Datei aus.

sync schreibt den Dateisystempuffer. Es zwingt veränderte Blöcke auf die Festplatte und aktualisiert den Superblock.

tac fügt Dateien rückwärts zusammen.

tail gibt die letzten zehn (oder die angegebene Anzahl) von Zeilen aus einer Datei aus.

tee liest von der Standard Eingabe während gleichzeitig auf die Standard Ausgabe 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 null−Länge erzeugt.

tr übersetzt, quetscht oder entfernt Zeichen von der Standard Eingabe.

true macht nichts, ist immer erfolgreich. Beendet immer mit einem Status Code der Erfolg bedeutet.

tsort führt eine topologische Sortierung durch. Schreibt eine vollkommen sortierte Liste entsprechend der teilweisen Sortierung in einer Datei.

tty gibt den Dateinamen des Terminals aus das mit der Standard Eingabe 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.

Coreutils Installationsabhängigkeiten

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

DejaGnu

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von DejaGnu−1.4.3* in Kapitel 5.

Offizielle Download Adresse

DejaGnu (1.4.3):
ftp://ftp.gnu.org/gnu/dejagnu/

Inhalt von DejaGnu

Das Paket DejaGnu enthält ein Grundgerüst zum testen anderer Programme.

Installiertes Programm: runtest

Kurze Beschreibung

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

DejaGnu Installationsabhängigkeiten

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

Diffutils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Diffutils−2.8.1* in Kapitel 6.

Offizielle Download Adresse

Diffutils (2.8.1):
ftp://ftp.gnu.org/gnu/diffutils/

Inhalt von Diffutils

Die Programme dieses Pakets können die Unterschiede zwischen zwei Dateien oder Verzeichnissen anzeigen. Die häufigste Anwendung ist das erstellen von Software−Patches.

Installierte Programme: cmp, diff, diff3 und sdiff

Kurze Beschreibungen

cmp vergleicht zwei Dateien und berichtet, ob, und an welchen Bytes sie sich unterscheiden.

diff vergleicht zwei Dateien oder Verzeichnisse und berichtet, welche Zeilen sich in den Dateien unterscheiden.

diff3 vergleicht drei Dateien Zeile für Zeile.

sdiff verschmelzt interaktiv zwei Dateien und gibt das Ergebnis aus.

Diffutils Installationsabhängigkeiten

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

E2fsprogs

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von E2fsprogs−1.34* in Kapitel 6.

Offizielle Download Adresse

E2fsprogs (1.34):
ftp://download.sourceforge.net/pub/sourceforge/e2fsprogs/
http://download.sourceforge.net/e2fsprogs/

Inhalt von E2fsprogs

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

Installierte Programme: badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs und uuidgen.

Installierte Bibliotheken: libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so] und libuuid.[a,so]

Kurze Beschreibungen

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

blkid ist ein Kommandozeilenprogramm zum auffinden und ausgeben von Blockgerät Eigenschaften.

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 Dateisystem 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. Standardmässig prüft es die in `/etc/fstab` aufgelisteten Dateisysteme.

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 Quellcode, 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 ein `lost+found` Verzeichnis auf einem second extended Dateisystem zu erzeugen. Es fürt eine Vorzuweiseung von Disk Blocks zu diesem Verzeichnis durch um damit e2fsck die Arbeit zu erleichtern.

resize2fs kann zum vergrössern oder verkleinern einest 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 zur identifikation von 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 die hinter dem lokalen System verfügbar sein könnten.

E2fsprogs Installationsabhängigkeiten

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

Ed

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Ed−0.2* in Kapitel 6.

Offizielle Download Adresse

Ed (0.2):
ftp://ftp.gnu.org/gnu/ed/

Ed Mkstemp Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/ed−0.2−mkstemp.patch

Inhalt von Ed

GNU ed ist ein 8bit−fähiger, POSIX−konformer Editor.

Installierte Programme: ed und red (Link auf ed)

Kurze Beschreibungen

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 Verzeichnis bearbeiten und keine Shell Kommandos ausführen.

Ed Installationsabhängigkeiten

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

Expect

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Expect−5.39.0* in Kapitel 5.

Offizielle Download Adresse

Expect (5.39.0):
http://expect.nist.gov/src/

Expect Spawn Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/expect−5.39.0−spawn.patch

Inhalt von Expect

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

Installierte Programme: expect

Installierte Bibliotheken: libexpect5.39.a

Kurze Beschreibung

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

Expect Installationsabhängigkeiten

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

File

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von File−4.04* in Kapitel 6.

Offizielle Download Adresse

File (4.04):
ftp://ftp.gw.com/mirrors/pub/unix/file/

Alternative Download Adresse:
ftp://gaosu.rave.org/pub/linux/lfs/

Inhalt von File

File ist ein kleines Werkzeug zum identifizieren von Dateitypen.

Installiertes Programm: file

Installierte Bibliotheken: libmagic.[a,so]

Kurze Beschreibung

file versucht, Dateien zu klassifizieren. Dazu führt es verschiedene Tests durch: Dateisystem Tests, Tests mit magische Nummern, und Sprachtests. Der erste erfolgreiche Test entscheidet über das Ergebnis.

libmagic enthält Routinen zur erkennung von magischen Nummern; wird vom file Programm verwendet.

File Installationsabhängigkeiten

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

Findutils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Findutils−4.1.20* in Kapitel 6.

Offizielle Download Adresse

Findutils (4.1.20):
ftp://alpha.gnu.org/gnu/findutils/

Inhalt von Findutils

Das Findutils Paket enthält Programme zum auffinden von Dateien, entweder on−the−fly (indem ein Verzeichnisbaum live durchsucht wird) oder durch Suche in einer Datenbank.

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

Kurze Beschreibungen

bigram wurde früher benutzt um locate Datenbanken zu erzeugen.

code wurde früher benutzt um locate Datenbanken zu erzeugen.

find durchsucht einen Verzeichnisbaum nach Dateien die einem bestimmten Kriterium entsprechen.

frcode wird von updatedb aufgerufen um die Liste der Dateinamen zu komprimieren. Es benutzt sog. front−Kompression, welche die Datenbankgröße um den Faktor 4 bis 5 verkleinert.

locate sucht durch eine Datenbank von 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 es ihm nicht anders gesagt wurde) und trägt jeden gefundenen Dateinamen in die Datenbank ein.

xargs kann benutzt werden, um ein bestimmtes Kommando mit einer Liste von Dateien auszuführen.

Findutils Installationsabhängigkeiten

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

Flex

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Flex−2.5.4a* in Kapitel 6.

Offizielle Download Adresse

Flex (2.5.4a):
ftp://ftp.gnu.org/non−gnu/flex/

Inhalt von Flex

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

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

Installierte Bibliothek: libfl.a

Kurze Beschreibungen

flex ist ein Werkzeug das Programme erzeugt die Muster in Text erkennen können. Mustererkennung ist in vielen Programmen nützlich. Flex erzeugt aus einem Set an Regeln nach denen es suchen soll ein Programm, das nach diesen Mustern sucht. Der Grund warum man Flex nimmt, ist, 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.

Flex Installationsabhängigkeiten

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

Gawk

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Gawk−3.1.3* in Kapitel 6.

Offizielle Download Adresse

Gawk (3.1.3):
ftp://ftp.gnu.org/pub/gnu/gawk/

Gawk Libexecdir Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/gawk−3.1.3−libexecdir.patch

Inhalt von Gawk

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

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

Kurze Beschreibungen

gawk ist ein Programmen zum manipulieren von Textdateien. Es ist die GNU Implementierung von awk.

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

igawk gibt gawk die Möglichkeit, Dateien einzubinden.

pgawk ist die profiling Version von gawk.

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

Gawk Installationsabhängigkeiten

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

GCC

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von GCC−3.3.1* in Kapitel 6.

Offizielle Download Adresse

GCC (3.3.1):
ftp://ftp.gnu.org/pub/gnu/gcc/

GCC No−Fixincludes Patch:

http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−no_fixincludes−2.patch

GCC Specs Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−specs−2.patch

GCC Suppress−Libiberty Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−suppress−libiberty.patch

GCC−2 (2.95.3):
ftp://ftp.gnu.org/pub/gnu/gcc/

GCC−2 Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−2.patch

GCC−2 No−Fixincludes Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−no−fixinc.patch

GCC−2 Return−Type Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−returntype−fix.patch

Inhalt von GCC

Das Paket GCC enthält die Gnu Compiler Sammlung, inklusive dem C und C++ Compiler.

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 Beschreibungen

cpp ist der C Preprozessor. Er wird von dem Compiler benutzt um #include und #define Anweisungen in Quellcodedateien zu expandieren.

g++ ist der C++ Compiler.

gcc ist der C Compiler. Er wird benutzt um Quellcode eines Programms in Assembler Code zu übersetzen.

gccbug ist ein Shell Skript das helfen soll, gute Fehlerberichte zu erzeugen.

gcov ist ein Berichterstattungs Hilfsmittel. Es wird verwendet, um Programme zu analysieren und herauszufinden, wo Optimierungen die meiste Wirkung zeigen.

libgcc* enthält Laufzeitunterstützung für gcc.

libstdc++ ist die Standard C++ Bibliothek. Sie enthält viele häufig benutzte Funktionen.

libsupc++ stellt hilfreiche Routinen für die C++ Programmiersprache zur Verfügung.

GCC Installationsabhängigkeiten

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

Gettext

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Gettext−0.12.1* in Kapitel 6.

Offizielle Download Adresse

Gettext (0.12.1):
ftp://ftp.gnu.org/gnu/gettext/

Inhalt von Gettext

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

Installierte Programme: autopoint, config.charset, config.rpath, 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 Beschreibungen

autopoint kopiert die Dateien einer standard 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.

gettext übersetzt Nachrichten in natürlicher Sprache in die Sprache des Anwenders. Dafür benutzt es einen Übersetzungsnachrichten Katalog.

gettextize kopiert alle standard Gettext Dateien in das Hauptverzeichnis eines Pakets um so den Beginn der internationalisierung zu erleichtern.

hostname zeigt den Netzwerk Hostnamen in verschiedenen Forman 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.

Gettext Installationsabhängigkeiten

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

Glibc

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Glibc−2.3.2* in Kapitel 6.

Offizielle Download Adresse

Glibc (2.3.2):
ftp://ftp.gnu.org/gnu/glibc/

Glibc−linuxthreads (2.3.2):
ftp://ftp.gnu.org/gnu/glibc/

Glibc Sscanf Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/glibc−2.3.2−sscanf−1.patch

Inhalt von Glibc

Glibc ist die C Bibliothek, sie stellt Systemaufrufe und grundlegende Funktionen wie open, malloc, printf usw. zur Verfügung. Die C Bibliothek wird von allen dynamisch gelinkten Programmen verwendet.

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 Beschreibungen

catchsegv kann benutzt werden um einen Stack Trace zu erzeugen, wenn ein Programm mit einem Speicherzugriffsfehler abbricht.

gencat erzeugt Nachrichtenkataloge.

getconf zeigt Systemspezifische Konfigurationswerte zu Dateisystemspezifischen Variablen an.

getent liest Einträge aus einer administrativen Datenbank.

glibcbug erzeugt einen Fehlerbericht und mailt ihn an die Bug−Emailadresse.

iconv führt Zeichensatzkonvertierungen durch.

iconvconfig erzeugt schnell ladende iconv Modulkonfigurationsdateien.

ldconfig konfiguriert die Laufzeit Bindungen des dynamischen Linkers.

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

lddlibc4 hilft ldd mit Objektdateien.

locale ist ein Perl Programm das dem Compiler mitteilt, ob er POSIX Locales für eingebaute Operationen benutzen oder nicht benutzen soll.

localedef kompiliert Locale−Spezifikationen.

mtrace...

nscd ist ein Namensdienst Cache Dämon. Er stellt einen Cache für die gängigsten Namensdienstanfragen zur Verfügung.

nscd_nischeck prüft, ob der sichere Modus für NIS+ Abfragen nötig ist.

pcprofiledump zeigt Informationen an die durch PC profiling erzeugt wurden.

pt_chown ist ein Hilfsprogramm für grantpt um Besitzer, Gruppe und Zugriffsreche auf einem Slave Pseudo Terminal zu setzen.

rpcgen erzeugt C Code zum implementieren des RPC Protokolls.

rpcinfo generiert einen RPC Call an einen RPC Server.

sln wird zum erzeugen von symbolischen Links benutzt. Das Programm ist statisch gelinkt, das ist nützliche um symbolische Links zu dymamischen Bibliotheken zu erstellen, wenn das dynamische Linkersystem aus irgendwelchen Gründen nicht funktioniert.

sprof liest und zeigt profiling Daten von gemeinsamen Objekten an.

tzselect fragt den Benutzer nach der Lokation des System und zeigt dann die dazugehörige Zeitzonenbeschreibung an.

xtrace erzeugt einen Trace der Auführung eines Programmes indem es die gerade ausgeführte Funktion ausgibt.

zdump gibt Zeitzonen aus.

zic ist der Zeitzonencompiler.

ld.so ist das Hilfsprogramm für gemeinsame ausführbare Bibliotheken.

libBrokenLocale wird von Programmen wie Mozilla verwendet um kaputte Locales zu beheben.

libSegFault ist ein Handler für Speicherzugriffsfehler. Er versucht Speicherzugriffsfehler aufzufangen.

libanl ist eine Bibliothek für asynchrone Namensauflösung.

libbsd−compat stellt Portabilität zur Verfügung die benötigt wird, damit einige BSD Programme unter Linux laufen.

libc ist die Haupt C Bibliothek −− eine Sammlung von gemeinsam benutzten Funktionen.

libcrypt ist die Kryptographiebibliothek.

libdl ist die Bibliothek zur Schnittstelle des dynamischen Linkers.

libg ist eine Laufzeitbibliothek für g++.

libieee ist die IEEE Fliesskomma Bibliothek.

libm ist die mathematische Bibliothek.

libmcheck enthält Code der beim booten ausgeführt wird.

libmemusage wird von memusage verwendet und hilft bei der Sammlung von Informationen über den Speicherverbrauch 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 benutzt werden um die verbrauchte Menge an CPU Zeit in bestimmten Codezeilen zurückzuverfolgen.

libpthread ist die POSIX threads Bibliothek.

libresolv enthält Funktionen zum erzeugen, senden und interpretieren von Paketen zu Internet Domain Name Servern.

librpcsvc enthält Funktionen die diverse RPC Dienste zur Verfügung stellen.

librt enthält Funktionen zu den Schnittstellen, die von der POSIX.1b Echtzeit Erweiterung spezifiziert werden.

libthread_db enthält nützliche Funktionen zum erzeugen von debuggern für multi−thread Programme.

libutil enthält Code für "standard" Funktionen die in vielen verschiedenen Unix Werkzeugen verwendet werden.

Glibc Installationsabhängigkeiten

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

Grep

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Grep−2.5.1* in Kapitel 6.

Offizielle Download Adresse

Grep (2.5.1):
ftp://ftp.gnu.org/gnu/grep/

Inhalt von Grep

Grep zeigt alle Zeilen einer Datei an die auf ein bestimmtes Muster passen.

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

Kurze Beschreibungen

egrep gibt die Zeilen aus, die auf einen regulären Ausdruck passen.

fgrep gibt die Zeilen aus, die auf eine Liste von festen Zeichenketten passen.

grep gibt die Zeilen aus, die auf einen einfachen regulären Ausdruck passen.

Grep Installationsabhängigkeiten

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

Groff

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Groff−1.19* in Kapitel 6.

Offizielle Download Adresse

Groff (1.19):
ftp://ftp.gnu.org/gnu/groff/

Inhalt von Groff

Groff enthält verschiedene Programme zur verarbeitung und formatierung von Text. Groff konvertiert normalen Text mit speziellen Kommandos in eine formatierte Ausgabe, so wie man es zum Beispiel in den Hilfeseiten (man−pages) sieht.

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 and zsoelim (Link auf soelim)

Kurze Beschreibungen

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

afmtodit erzeugt eine 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 in ein beschnittenes Bild.

grn ist ein groff Preprozessor für gremlin Dateien.

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

groff ist eine Benutzerschnittstelle zu dem 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 rät, 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 (Laser Drucker 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 schreibmaschinen−ähnliche 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 bilbiographische Datenbanken nach Referenzen die bestimmte Schlüssel enthalten und gibt die gefundenen Referenzen aus.

lookbib gibt einen Prompt auf die standard Fehlerausgabe (solange die standard Eingabe kein Terminal ist), liest eine Zeile mit Stichwörtern von der standard Eingabe, 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 Preprozessor 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 Preprozessoren und Postprozessoren in der richtigen Reihenfolge und mit den richtigen Optionen aufruft.

zsoelim ist die GNU Implementierung von soelim.

Groff Installationsabhängigkeiten

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

Grub

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Grub−0.93* in Kapitel 6.

Offizielle Download Adresse

Grub (0.93):
ftp://alpha.gnu.org/pub/gnu/grub/

Grub Gcc33 Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/grub−0.93−gcc33−1.patch

Inhalt von Grub

Das Paket Grub enthält den Grub Bootloader.

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

Kurze Beschreibungen

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.

Grub Installationsabhängigkeiten

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

Gzip

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Gzip−1.3.5* in Kapitel 6.

Offizielle Download Adresse

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

Inhalt von Gzip

Gzip enthält Programme für die Dateikompression und −dekompression mit Hilfe des Lempel−Ziv Algorythmus (LZ77).

Installierte Programme: gunzip (Link auf gzip), gzexe, gzip, uncompress (Link auf gunzip), zcat (Link auf gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore und znew

Kurze Beschreibungen

gunzip dekomprimiert gzip Dateien.

gzexe wird zum erzeugen von selbstentpackenden ausführbaren Dateien verwendet.

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

zcat dekomprimiert gzip Dateien zur standard Ausgabe.

zcmp führt cmp auf gzip Dateien aus.

zdiff führt diff auf gzip Dateien aus.

zegrep führt egrep auf gzip Dateien aus.

zfgrep runs fgrep on gzipped files.

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.

Gzip Installationsabhängigkeiten

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

Inetutils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Inetutils−1.4.2* in Kapitel 6.

Official Download Location

Inetutils (1.4.2):
http://freshmeat.net/projects/inetutils/

Inhalt von Inetutils

Inetutils enthält verschiedene Netzwerk Clients und Server.

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

Kurze Beschreibungen

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.

Inetutils Installationsabhängigkeiten

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

Kbd

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Kbd−1.08* in Kapitel 6.

Offizielle Download Adresse

Kbd (1.08):
ftp://ftp.win.tue.nl/pub/linux−local/utils/kbd/

Kbd More−Programs Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/kbd−1.08−more−programs.patch

Inhalt von Kbd

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

Installiete Programme: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, getunimap, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (Link auf psfxtable), psfgettable (Link auf psfxtable), psfstriptable (Link auf psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start und unicode_stop

Kurze Beschreibungen

chvt ändert das vordergründige Virtuelle Terminal.

deallocvt zieht zugewiesene unbenutzte Virtuelle Terminals zurück.

dumpkeys gibt Tastaturübersetzungstabellen aus.

fgconsole gitb die Nummer des aktiven Virtuellen Terminals aus.

getkeycodes gitb 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 Zeichenausgabezuweisungstabelle in den Konsoletreiber lädt. Das 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 größe 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, NumLock standardmässig eingeschaltet zu haben, mit setleds +num kann man das 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 Toaste, nur ein einfacher schwarzer Schirm).

showconsolefont zeigt die aktutelle EGA/VGA Konsole Schrift an.

showkey zeigt Scancode und Keycode und ASCII Code der Taste an die auf der Tastatur gedrückt wurde.

unicode_start versetzt Tastatur und Konsole in den unicode Modus.

unicode_stop schaltet den unicode Modus von Tastatur und Konsole wieder aus.

Kbd Installationsabhängigkeiten

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

Less

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Less−381* in Kapitel 6.

Offizielle Download Adresse

Less (381):
ftp://ftp.gnu.org/gnu/less/

Inhalt von Less

Less ist ein Textanzeigeprogramm. Es zeigt den Inhalt von Dateien oder Datenströmen an. Less hat einige Merkmale die `more` nicht hat, wie zum Beispiel die Möglichkeit rückwärts zu scrollen.

Installierte Programme: less, lessecho und lesskey

Kurze Beschreibungen

less ist ein Dateibetrachter. Er zeigt den Inhalt einer Datei an und lässt sie darin scrollen, Zeichenketten finden und zu Markierungen springen.

lessecho wird zum expandieren von Metazeichen in Unix Dateinamen benötigt, sowie * und ?.

lesskey wird zum angeben der Tastenbindungen für less benutzt.

Less Installationsabhängigkeiten

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

LFS−Bootskripte

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von LFS Bootscripts−1.12* in Kapitel 6.

Offizielle Download Adresse

LFS−Bootskripte (1.12):
http://downloads.linuxfromscratch.org/

Inhalt der LFS−Bootskripte

Die LFS−Bootskripte enthalten Shell Skripte im SysV init Stil. Diese Skripte erfüllen verschiedene Aufgaben wie zum Beispiel Dateisystemprüfungen beim Systemstart, das laden von Tastaturmappings, Netzwerkeinrichtung oder das Beenden von Prozessen beim herunterfahren des Systems.

Installierte Skripte: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, swap, sysklogd und template

Kurze Beschreibungen

Das **checkfs** Skript prüft Dateisystem bevor sie gemountet werden (mit der Ausnahme von journalisierenden 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 eine eventuell vorhandene `/etc/nologin`, `/fastboot` und `/forcefsck` Dateien.

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 mit Netzwerkgeräten.

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 sind nicht Netzwerkbasiert sind.

Das **mountproc** Skript wird zum einhängen des proc Dateisystems benutzt.

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, das 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 **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.

LFS−Bootscripts Installationsabhängigkeiten

Bzip2 ist abhängig von: Bash, Coreutils.

Lfs−Utils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Lfs−Utils−0.3* in Kapitel 6.

Official Download Location

Lfs−utils (0.3):
http://www.linuxfromscratch.org/~winkie/downloads/lfs−utils/

Inhalt von Lfs−Utils

Das Lfs−Utils Paket enthält ein paar Programme die von verschiedenen Paketen gebraucht werden, aber nicht gross genug sind um ein eigenes Paket zu rechtfertigen.

Installierte Programme: mktemp, tempfile, http−get und iana−net

Installierte Dateien: protocols, services

Kurze Beschreibungen

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

tempfile erzeugt temporäre Dateien auf weniger sichere Weise als **`mktemp`**. Es ist nur aus Gründen der Rückwärtskompatibilität installiert.

Das **http−get** Skript nutzt ein nettes bekanntes Feature der **`bash`**, "net redirection" genannt. Damit kann man downloads von Webseiten ohne zusätzliche Programme machen.

iana−net benutzt das **`http−get`** Skript um die Beschaffung von IANA's Dienst− und Protokollkonfigurationsdateien zu vereinfachen.

Lfs−Utils Installationsabhängigkeiten

(No dependencies checked yet.)

Libtool

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Libtool−1.5* in Kapitel 6.

Offizielle Download Adresse

Libtool (1.5):
ftp://ftp.gnu.org/gnu/libtool/

Inhalt von Libtool

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

Installierte Programme: libtool und libtoolize

Installierte Bibliotheken: libltdl.[a,so].

Kurze Beschreibungen

libtool stellt vereinheitlichte Dienste zum erstellen von Bibliotheken zur Verfügung.

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

libltdl versteckt die verschiedenen Schwierigkeiten mit Bibliotheken die dlopen verwenden.

Libtool Installationsabhängigkeiten

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

Linux (der Kernel)

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Linux−2.4.22* in Kapitel 8.

Offizielle Download Adresse

Linux (2.4.22):
ftp://ftp.kernel.org/pub/linux/kernel/

Inhalt von Linux

Der Linux Kernel ist der Kern eines jeden Linux Systems. Er ist sozusagen der Herzschlag von Linux. Wenn der Computer eingeschaltet wird und ein Linux System startet, dann ist der Kernel das erste Stück Software das gestartet wird. Der Kernel initialisiert die Geräte und Hardware Komponenten: serielle Schnittstellen, parallele Schnittstellen, Soundkarten, Netzwerkkarten, IDE und SCSI Controller und vieles mehr. Zusammenfassend kann man sagen, der Kernel stellt dem System die Hardware zur Verfügung, so das die Software damit laufen kann.

Installierte Dateien: Der Kernel und die Kernel Header

Kurze Beschreibungen

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 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` Verzeichnis ihres Systems sollten *immer* diejenigen sein mit denen die Glibc kompiliert wurde und sollten daher bei einem Kernelupgrade *nicht* ersetzt werden.

Linux Installationsabhängigkeiten

Linux ist abhängig von: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

M4

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von M4−1.4* in Kapitel 6.

Offizielle Download Adresse

M4 (1.4):
ftp://ftp.gnu.org/gnu/m4/

Inhalt von M4

M4 ist ein Makroprozessor. Er kopiert die Eingabe zur Ausgabe und führt dabei Makros aus. Die Makros können entweder vordefiniert oder selbstgeschrieben sein und können beliebige Argumente übernehmen. Neben der Fähigkeit Makros auszuführen hat M4 eingaute Funktionen um benannte Dateien einzufügen, Unix Kommandos auszuführen, Integer Berechnungen durchzuführen, Text zu manipulieren, Rekursionen zu behandeln usw. M4 kann entweder als Front−end zu einem Compiler oder als eigenständiger Makroprozessor genutzt werden.

Installierte Programme: m4

Kurze Beschreibungen

m4 kopiert Dateien und expandiert währenddessen Makros die darin enthalten sind. Diese Makros können entweder eingebaut oder benutzerdefiniert sein und können eine beliebige Anzahl von Argumenten übernehmen. Neben der Makrofunktionalität kann m4 benannte Dateien einfügen, Unix Kommandos ausführen, integer Berechnungen durchführen, Textmanipulationen in verschiedenster Weise, und Rekursionen bearbeiten. Das m4 Programm kann entweder als Front−end zu einem Compiler oder als eigenständiger Makro Prozessor benutzt werden.

M4 Installationsabhängigkeiten

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

Make

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Make−3.80* in Kapitel 6.

Offizielle Download Adresse

Make (3.80):
ftp://ftp.gnu.org/gnu/make/

Inhalt von Make

Make erkennt automatisch, welche Teile eines grossen Programmes erneut kompiliert werden müssen und welche Kommandos dazu nötig sind.

Installiertes Programm: make

Kurze Beschreibung

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

Make Installationsabhängigkeiten

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

MAKEDEV

Eine Installationsanleitung finden sie im Abschnitt namens *Erstellen der Gerätedateien (Makedev−1.7)* in Kapitel 6.

Offizielle Download Adresse

MAKEDEV (1.7):
http://downloads.linuxfromscratch.org/

Inhalt von MAKEDEV

Das Skript MAKEDEV erstellt statische Gerätedateien. Diese liegen normalerweise im `/dev` Verzeichnis. Detaillierte Informationen über die Gerätedateien finden sie in der Datei `Documentation/devices.txt` in den Linux Kernel Quellen.

Installiertes Skript: MAKEDEV

Kurze Beschreibung

MAKEDEV ist ein Skript zum erstellen benötigter statische Gerätedateien. Diese liegen normalerweise im `/dev` Verzeichnis.

MAKEDEV Installationsabhängigkeiten

Make ist abhängig von: Bash, Coreutils.

Man

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Man−1.5m2* in Kapitel 6.

Offizielle Download Adresse

Man (1.5m2):
ftp://ftp.win.tue.nl/pub/linux−local/utils/man/

Man 80−Columns Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−80cols.patch

Man Manpath Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−manpath.patch

Man Pager Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−pager.patch

Inhalt von Man

Man ist das Programm zum Seitenweisen anzeigen von Hilfeseiten (man−pages).

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

Kurze Beschreibungen

apropos durchsucht die whatis Datenbank und gibt kurze Beschreibungen zu den Kommandos aus, die die angegebene Zeichenkette enthalten.

makewhatis erstellt die whatis Datenbank. Es liest alle Man−pages und schreibt für jedes Paket den Namen und eine kurze Beschreibung in die whatis Datenbank.

man formatiert die angefragte 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.

Man Installationsabhängigkeiten

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

Man−pages

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren der Man−pages−1.60* in Kapitel 6.

Offizielle Download Adresse

Man−pages (1.60):
ftp://ftp.kernel.org/pub/linux/docs/manpages/

Inhalt von Man−pages

Die Hilfeseiten (man−pages) enthalten über 1200 Seiten Hilfetexte. Die Dokumentation beschreibt sehr detailliert C und C++ Funktionen, einige wichtige Gerätedateien und enthält Dokumente die in anderen Paketen sonst fehlen würden.

Installierte Dateien: verschiedene Hilfeseiten

Kurze Beschreibung

Beispiele für die enthaltenen *Hilfeseiten* sind die Seiten zu den C und C++ Funktionen, wichtigen Gerätedateien und wichtigen Konfigurationsdateien.

Man−pages Installationsabhängigkeiten

Man ist abhängig von: Bash, Coreutils, Make.

Modutils

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Modutils−2.4.25* in Kapitel 6.

Offizielle Download Adresse

Modutils (2.4.25):
ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils/

Inhalt von Modutils

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

Installierte Programme: depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (Link auf insmod), kernelversion, ksyms (Link aus insmod), lsmod (Link auf insmod), modinfo, modprobe (Link auf insmod) und rmmod (Link auf insmod)

Kurze Beschreibungen

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

genksyms erzeugt 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 debugginzwecken 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 Abhängigkeitsdatei, die von depmod erzeugt wurde, um benötigte Module automatisch nachzuladen.

rmmod entlädt ein Modul aus dem laufenden Kernel.

Modutils Installationsabhängigkeiten

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

Ncurses

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Ncurses−5.3* in Kapitel 6.

Offizielle Download Adresse

Ncurses (5.3):
ftp://ftp.gnu.org/gnu/ncurses/

Ncurses Etip Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−etip−2.patch

Ncurses Vsscanf Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−vsscanf.patch

Inhalt von Ncurses

Ncurses enthält Bibliotheken zur Verwendung von Zeichen und Terminals, inklusive Schaltflächen und Menüs.

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 Beschreibungen

captoinfo konvertiert termcap Beschreibungen zu terminfo Beschreibungen.

clear löscht den Bildschirminhalt..

infocmp vergleicht terminfo Beschriebungen oder gibt sie aus.

infotocap konvertiert terminfo Beschreibungen zu termcap Beschreibungen.

reset reinitialisiert ein Terminal mit seinen Standardwerten.

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, welche von den ncurses Bibliotheksroutinen benötigt wird. Eine terminfo Datei enthält Informationen über die Fähigkeiten eines bestimmten Terminals.

toe listest alle verfügbaren Terminaltypen auf und gibt zu jedem seinen Namen und seine Beschreibung aus.

tput macht der Shell die Werte von Terminalabhängigen Fähigkeiten verfügbar. 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 Formen.

libmenu* enthält Funktionen zum implementieren von Memüs.

libpanel* enthält Funktionen zum implementieren von Schaltflächen.

Ncurses Installationsabhängigkeiten

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

Net−tools

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Net−tools−1.60* in Kapitel 6.

Offizielle Download Adresse

Net−tools (1.60):
http://www.tazenda.demon.co.uk/phil/net−tools/

Net−tools Mii−Tool−Gcc33 Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/net−tools−1.60−miitool−gcc33−1.patch

Inhalt von Net−tools

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

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

Kurze Beschreibungen

arp wird zum manipulieren des ARP Cache des Kernels verwendet. Normalerweise zum hinzufügen oder entfernen eines Eintrags 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, Routing Tabellen und Gerätestatistiken an.

nisdomainname tut das selbe wie domainname.

plipconfig wird zur Feinkonfiguration eines PLIP Geräts 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 tut das selbe wie domainname.

Net−tools Installationsabhängigkeiten

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

Patch

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Patch−2.5.4* in Kapitel 6.

Offizielle Download Adresse

Patch (2.5.4):
ftp://ftp.gnu.org/gnu/patch/

Inhalt von Patch

Patch manipuliert Dateien auf Basis einer Patch−Datei. Eine Patch−Datei ist üblicherweise eine Liste mit Änderungsanweisungen die mit Hilfe des Programms diff erzeugt wurde.

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 diff Programm 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.

Patch Installationsabhängigkeiten

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

Perl

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Perl−5.8.0* in Kapitel 6.

Offizielle Download Adresse

Perl (5.8.0):
http://www.perl.com/

Perl Libc Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/perl−5.8.0−libc−3.patch

Inhalt von Perl

Das Perl Paket enthält die Skriptsprache Perl (Practical Extraction and Report Language). Perl kombiniert einige der besten Merkmale von C, sed, awk und sh in einer einzigen, mächtigen Skriptsprache.

Installierte Programme: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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 aufzulisten)

Kurze Beschreibungen

a2p übersetzt awk nach 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 mailen 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 korrent 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 durch "cc −g −S" erzeugt wurden.

s2p übersetzt sed zu perl.

splain wird zur Analyse von Warnungen in Perl benutzt.

xsubpp konvertiert Perl XS Code zu C Code.

Perl Installationsabhängigkeiten

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

Procinfo

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Procinfo−18* in Kapitel 6.

Offizielle Download Adresse

Procinfo (18):
ftp://ftp.cistron.nl/pub/people/svm/

Inhalt von Procinfo

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

Installierte Programme: lsdev, procinfo und socklist

Kurze Beschreibungen

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.

Procinfo Installationsabhängigkeiten

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

Procps

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Procps−3.1.11* in Kapitel 6.

Offizielle Download Adresse

Procps (3.1.11):
http://procps.sourceforge.net/

Procps Locale Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/procps−3.1.11−locale−fix.patch

Inhalt von Procps

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

Installierte Programme: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w und watch

Installierte Bibliothek: libproc.so

Kurze Beschreibungen

free gibt die Menge and freiem und benutzten Arbeitsspeicher aus, sowohl physischen 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 das System läuft, wieviele Benutzer eingeloggt sind, und wie hoch die durchschnittliche 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, 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.

Procps Installationsabhängigkeiten

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

Psmisc

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Psmisc−21.3* in Kapitel 6.

Offizielle Download Adresses

Psmisc (21.3):
http://download.sourceforge.net/psmisc/
ftp://download.sourceforge.net/pub/sourceforge/psmisc/

Inhalt von Psmisc

Das Paket Psmisc enthält drei Programme zur Verwaltung des `/proc` Verzeichnisses.

Installierte Programme: fuser, killall und pstree

Kurze Beschreibungen

fuser zeigt die PIDs von Prozessen an, die gerde eine bestimmte Datei oder Dateisystem verwenden.

killall tötet Prozesse aufgrund ihres Namens. Es sendet ein Signal an alle Prozesse die ein bestimmtes Kommando ausführen.

pidof gibt die PIDs eines bestimmten Programms aus.

pstree zeigt laufende Prozesse als Baum an.

Psmisc Installationsabhängigkeiten

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

Sed

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Sed−4.0.7* in Kapitel 6.

Offizielle download Adresse

Sed (4.0.7):
ftp://ftp.gnu.org/gnu/sed/

Inhalt von Sed

Sed ist ein Stream Editor. Mit einem Stream Editor kann man einfache Textmanipulationen an einem Eingabestream vornehmen (z. B. einer Datei oder einer Pipe).

Installiertes Programm: sed

Kurze Beschreibung

sed wird zum filtern und transformieren von Dateien in einem einzigen Durchlauf verwendet.

Sed Installationsabhängigkeiten

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

Shadow

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Shadow−4.0.3* in Kapitel 6.

Offizielle download Adresse

Shadow (4.0.3):
ftp://ftp.pld.org.pl/software/shadow/

Shadow Newgrp Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/shadow−4.0.3−newgrp−fix.patch

Inhalt von Shadow

Das Shadow Paket wurde zur verstärkung der Sicherheit von System Passwörtern eingeführt.

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 Beschreibungen

chage ändert die maximale Anzahl von Tagen zwischen zwei nötigen Passwortänderungen.

chfn wird benutzt um den vollständigen Namen und ein paar andere Informationen eines Benutzers zu ändern.

chpasswd wird benutzt um das Password mehrerer Benutzer in einem Durchlauf zu ändern.

chsh wird benutzt um die Standardshell 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, um 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 die letzten Anmeldungen aller Benutzer 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 password 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, Heimatverzeichnis und ähnliches für einen Benutzer.

vigr kann zum editieren von `/etc/group` oder `/etc/gshadow` Dateien benutzt werden.

vipw kann zum editieren von `/etc/passwd` oder `/etc/shadow` Dateien benutzt werden.

libmisc...

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

Shadow Installationsabhängigkeiten

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

Sysklogd

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Sysklogd−1.4.1* in Kapitel 6.

Offizielle download Adresse

Sysklogd (1.4.1):
http://www.infodrom.org/projects/sysklogd/

Inhalt von Sysklogd

Die in Sysklogd enthaltenen Programme dienen zum aufzeichnen von System Logs, zum Beispiel die des Kernels.

Installierte Programme: klogd und syslogd

Kurze Beschreibungen

klogd ist ein Systemdämon zum abfangen und loggen von Kernelnachrichten.

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

Sysklogd Installationsabhängigkeiten

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

Sysvinit

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Sysvinit−2.85* in Kapitel 6.

Offizielle download Adresse

Sysvinit (2.85):
ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/

Inhalt von Sysvinit

Das Sysvinit Paket enhält Programme mit denen sie das starten, ausführen und beenden aller anderen Programme kontrollieren können.

Installierte Programme: halt, init, killall5, last, lastb (Link auf last), mesg, pidof (Link auf killall5), poweroff (Link auf halt), reboot (Link auf halt), runlevel, shutdown, sulogin, telinit (Link auf init), utmpdump und wall

Kurze Beschreibungen

halt ruft üblicherweise shutdown mit dem −h Schalter auf, ausser wenn der aktuelle Runlevel 0 ist, dann teilt es dem Kernel mit, das System anzuhalten. Vorher notiert es in `/var/log/wtmp` das 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 hochgefahren werden sollen.

killall5 sendet ein Signal an alle Prozesse, ausser denen in der eigenen Sitzung −− so tötet es nicht die Programme die das Skript ausführen welches es aufgerufen hat.

last zeigt, welcher Benutzer als letztes eingeloggt und ausgeloggt hat, indem es durch die Datei `/var/log/wtmp` sucht. Es kann auch Systemstarts und Shutdowns sowie Runlevel−wechsel 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 zu rebooten. 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 Einzelbenutzermodus 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.

Sysvinit Installationsabhängigkeiten

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

Tar

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Tar−1.13.25* in Kapitel 6.

Offizielle download Adresse

Tar (1.13.25):
ftp://alpha.gnu.org/gnu/tar/

Inhalt von Tar

Tar ist ein Programm zum speichern und extrahieren von Dateien aus sog. Tar−Archiven.

Installierte Programme: rmt und tar

Kurze Beschreibungen

rmt wird zum entfernten manipulieren von magnetischen Bandlaufwerken verwendet und benutzt das Interprozesskommunikation.

tar wird zum erzeugen und extrahieren von Dateien aus einem Archiv verwendet..

Tar Installationsabhängigkeiten

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

Tcl

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Tcl−8.4.4* in Kapitel 5.

Offizielle download Adresse

Tcl (8.4.4):
http://download.sourceforge.net/tcl/
ftp://download.sourceforge.net/pub/sourceforge/tcl/

Inhalt von Tcl

Das Tcl Paket enthält die sog. Tool Command Language.

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.

Tcl Installationsabhängigkeiten

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

Texinfo

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Texinfo−4.6* in Kapitel 6.

Offizielle download Adresse

Texinfo (4.6):
ftp://ftp.gnu.org/gnu/texinfo/

Inhalt von Texinfo

Texinfo enthält Programme zum lesen, schreiben und konvertieren von Info Dokumenten (System Dokumentation).

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

Kurze Beschreibungen

info wird zum lesen von Info Dokumenten benutzt. Info Dokumente sind ähnlich wie Man−pages, aber gehen oft tiefer in die Materie als einfach nur die möglichen Parameter zu beschreiben. Vergleichen sie zum Beispiel man 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.

Texinfo Installationsabhängigkeiten

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

Util−linux

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Util−linux−2.12* in Kapitel 6.

Offizielle download Adresse

Util−linux (2.12):
http://ftp.cwi.nl/aeb/util−linux/

Inhalt von Util−linux

Util−linux enthält verschiedene Werkzeuge. Einige der etwas bekannteren Programme werden z. B. zum mounten, entmounten, formatieren, partitionieren und verwalten von Festplatten, öffnen von tty Ports und zum auslesen von Kernel Meldungen genutzt.

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, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, 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, test.bash, test.tcsh, tunelp, ul, umount, vidmode (Link auf rdev), whereis und write

Kurze Beschreibungen

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

arch gibt die Systemarchitektur aus.

blockdev ermöglicht die Verwendung von Blockgerät−ioctls in 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 Tastetnkombination 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 Kernel Bootmeldungen 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.

kill beendet einen Prozess.

line kopiert eine einzelne Zeile.

logger gibt eine Nachricht in das Logsystem ein.

look zeigt die Zeilen an die mit einer bestimmten Zeichenkette beginnen.

losetup konfiguriert und kontrolliert das Loopback Gerät.

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 bindet ein Dateisystem auf einem Gerät an ein Verzeichnis im Verzeichnisbaum.

namei zeigt die symbolischen Links ein Pfadnamen an.

pg zeigt eine Textdatei seitenweise an.

pivot_root macht ein Dateisystem zu dem neuen root−Dateisystem für einen bestimmten Prozess.

ramsize kann benutzt werden um die Größe einer RAM Disk in einem bootbaren Abbild zu setzen.

rdev kann ein root−Gerät abfragen und setzen und noch weitere Dinge in einem bootfähigen Abbild.

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−Schalter eines bootfähigen Abbilds setzen.

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 Escape−Sequenzen zum unterstreichen, die ein verwendetes Terminal versteht.

umount löst ein Dateisystem aus dem System−Verzeichnisbaum.

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 solange der Benutzer solche Nachrichten nich deaktiviert hat.

Util−linux Installationsabhängigkeiten

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

Vim

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Vim−6.2* in Kapitel 6.

Offizielle download Adresse

Vim (6.2):
ftp://ftp.vim.org/pub/editors/vim/unix/

Inhalt von Vim

Vim enthält einen sehr anpassungsfähigen Text Editor, extra programmiert zum effizienten bearbeiten von Text.

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

---

Kurze Beschreibungen

efm_filter.pl 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 Rechtschreibung von Argumenten.

rview ist eine eingeschränkte Version von view: keine Shell Kommandos und vim kann nicht pausiert werden.

rvim ist eine eingeschränkte Version von vim: keine Shell Kommandos und vim kann nicht pausiert werden.

shtags.pl erzeugt eine Markup Datei für Perl Skripte.

tcltags erzeugt eine Markup Datei für TCL Code.

vi startet vim im vi Modus.

view startet vim im nur−lesen Modus.

vim ist der Editor.

vim132 startet vim im 132−spalten Terminalmodus.

vim2html.pl konvertiert vim Dokumentation in 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 Tasten und Kommandos von vim bei.

xxd erzeugt eine hex−Ausgabe einer Datei. Das geht auch umgekehrt, und kann zum patchen von Binärdateien benutzt werden.

Vim Installationsabhängigkeiten

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

Zlib

Eine Installationsanleitung finden sie im Abschnitt namens *Installieren von Zlib−1.1.4* in Kapitel 6.

Official Download Location

Zlib (1.1.4):
http://www.gzip.org/zlib/

Zlib Vsnprintf Patch:
http://www.linuxfromscratch.org/patches/lfs/5.0/zlib−1.1.4−vsnprintf.patch

Inhalt von Zlib

Zlib enthält die Bibliothek libz. Sie wird von vielen Programmen zum komprimieren und dekomprimieren genutzt.

Installierte Bibliotheken: libz[a,so]

Kurze Beschreibung

libz* enthält Funktionen zum komprimieren und dekomprimieren die von einigen Programmen benutzt werden.

Zlib Installationsabhängigkeiten

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

Anhang B. Index aller Programme und Bibliotheken

Dies ist eine Liste aller Programme und Bibliotheken die in diesem Buch installiert werden, inklusive der Angabe zu welchem Paket aus Anhang A es/sie gehört.

- a2p : Perl
- acinstall : Automake
- aclocal : Automake
- addftinfo : Groff
- addr2line : Binutils
- afmtodit : Groff
- agetty : Util−linux
- apropos : Man
- ar : Binutils
- arch : Util−linux
- arp : Net−tools
- as : Binutils
- attrs : Perl
- autoconf : Autoconf
- autoheader : Autoconf
- autom4te : Autoconf
- automake : Automake
- autopoint : Gettext
- autoreconf : Autoconf
- autoscan : Autoconf
- autoupdate : Autoconf
- awk : Gawk
- badblocks : E2fsprogs
- basename : Coreutils
- bash : Bash
- bashbug : Bash
- bigram : Findutils
- bison : Bison
- blkid : E2fsprogs
- blockdev : Util−linux
- bunzip2 : Bzip2
- bzcat : Bzip2
- bzcmp : Bzip2
- bzdiff : Bzip2
- bzegrep : Bzip2
- bzfgrep : Bzip2
- bzgrep : Bzip2
- bzip2 : Bzip2
- bzip2recover : Bzip2
- bzless : Bzip2
- bzmore : Bzip2
- c++ : GCC
- c++filt : Binutils