Linux From Scratch

Version 6.7

製作： Gerard Beekmans
編集： Matthew Burgess 、Bruce Dubbs

Linux From Scratch: Version 6.7

: 製作： Gerard Beekmans 、 編集： Matthew Burgess 、Bruce Dubbs

製作著作 © 1999-2010 Gerard Beekmans

Copyright © 1999-2010, Gerard Beekmans

All rights reserved.

本書は クリエイティブコモンズライセンス に従います。

本書のインストール手順のコマンドを抜き出したものは MIT ライセンス に従ってください。

Linux® は Linus Torvalds の登録商標です。

目次

序文 vii
i. はしがき vii
ii. 対象読者 vii
iii. LFS が対象とする CPU アーキテクチャ viii
iv. LFS と各種標準 viii
v. 各パッケージを用いる理由 x
vi. 必要な知識 xiii
vii. ホストシステム要件 xiv
viii. 本書の表記 xv
ix. 本書の構成 xvi
x. 正誤情報 xvii
xi. 日本語訳について xvii
I. はじめに 1
1. はじめに 2
1.1. LFS をどうやって作るか 2
1.2. 前版からの変更点 2
1.3. 変更履歴 4
1.4. 変更履歴（日本語版） 8
1.5. 情報源 8
1.6. ヘルプ 8
II. ビルド作業のための準備 11
2. 新しいパーティションの準備 12
2.1. はじめに 12
2.2. 新しいパーティションの生成 12
2.3. ファイルシステムの生成 13
2.4. 新しいパーティションのマウント 14
3. パッケージとパッチ 16
3.1. はじめに 16
3.2. 全パッケージ 17
3.3. 必要なパッチ 22
4. 準備作業の仕上げ 24
4.1. $LFSについて 24
4.2. $LFS/tools ディレクトリの生成 24
4.3. LFS ユーザーの追加 24
4.4. 環境設定 25
4.5. SBU 値について 26
4.6. テストスイートについて 27
5. 一時的環境の構築 28
5.1. はじめに 28
5.2. ツールチェーンの技術的情報 28
5.3. 全般的なコンパイル手順 29
5.4. Binutils-2.20.1 - 1回め 31
5.5. GCC-4.5.1 - 1回め 33
5.6. Linux-2.6.35.4 API ヘッダ 35
5.7. Glibc-2.12.1 36
5.8. ツールチェーンの調整 38
5.9. Binutils-2.20.1 - 2回め 39
5.10. GCC-4.5.1 - 2回め 40
5.11. Tcl-8.5.8 43
5.12. Expect-5.44.1.15 44
5.13. DejaGNU-1.4.4 46
5.14. Ncurses-5.7 47
5.15. Bash-4.1 48
5.16. Bzip2-1.0.5 49
5.17. Coreutils-8.5 50
5.18. Diffutils-3.0 51
5.19. File-5.04 52
5.20. Findutils-4.4.2 53

5.21. Gawk-3.1.8 54
5.22. Gettext-0.18.1.1 55
5.23. Grep-2.6.3 56
5.24. Gzip-1.4 57
5.25. M4-1.4.14 58
5.26. Make-3.82 59
5.27. Patch-2.6.1 60
5.28. Perl-5.12.1 61
5.29. Sed-4.2.1 62
5.30. Tar-1.23 63
5.31. Texinfo-4.13a 64
5.32. ストリップ 65
5.33. 所有者の変更 65
III. LFSシステムの構築 66
6. 基本的なソフトウェアのインストール 67
6.1. はじめに 67
6.2. 仮想カーネルファイルシステムの準備 67
6.3. パッケージ管理 68
6.4. Chroot 環境への移行 70
6.5. ディレクトリの生成 71
6.6. 基本的なファイルとリンクの生成 72
6.7. Linux-2.6.35.4 API ヘッダ 74
6.8. Man-pages-3.25 75
6.9. Glibc-2.12.1 76
6.10. ツールチェーンの再調整 82
6.11. Zlib-1.2.5 84
6.12. Binutils-2.20.1 85
6.13. GMP-5.0.1 87
6.14. MPFR-3.0.0 89
6.15. MPC-0.8.2 90
6.16. GCC-4.5.1 91
6.17. Sed-4.2.1 95
6.18. Pkg-config-0.25 96
6.19. Ncurses-5.7 97
6.20. Util-linux-ng-2.18 99
6.21. E2fsprogs-1.41.12 103
6.22. Coreutils-8.5 106
6.23. Iana-Etc-2.30 110
6.24. M4-1.4.14 111
6.25. Bison-2.4.3 112
6.26. Procps-3.2.8 113
6.27. Grep-2.6.3 114
6.28. Readline-6.1 115
6.29. Bash-4.1 117
6.30. Libtool-2.2.10 119
6.31. GDBM-1.8.3 120
6.32. Inetutils-1.8 121
6.33. Perl-5.12.1 123
6.34. Autoconf-2.67 126
6.35. Automake-1.11.1 127
6.36. Bzip2-1.0.5 128
6.37. Diffutils-3.0 130
6.38. Gawk-3.1.8 131
6.39. File-5.04 132
6.40. Findutils-4.4.2 133
6.41. Flex-2.5.35 134
6.42. Gettext-0.18.1.1 136
6.43. Groff-1.20.1 138
6.44. GRUB-1.98 140
6.45. Gzip-1.4 141
6.46. IPRoute2-2.6.35 142

6.47. Kbd-1.15.2 144
6.48. Less-436 146
6.49. Make-3.82 147
6.50. Man-DB-2.5.7 148
6.51. Module-Init-Tools-3.12 151
6.52. Patch-2.6.1 152
6.53. Psmisc-22.12 153
6.54. Shadow-4.1.4.2 154
6.55. Sysklogd-1.5 157
6.56. Sysvinit-2.88dsf 158
6.57. Tar-1.23 161
6.58. Texinfo-4.13a 162
6.59. Udev-161 164
6.60. Vim-7.3 166
6.61. デバッグシンボルについて 169
6.62. 再度のストリップ 169
6.63. 仕切り直し 169
7. ブートスクリプトの設定 171
7.1. はじめに 171
7.2. LFS-ブートスクリプト-20100627 172
7.3. ブートスクリプトはどのようにして動くのか 174
7.4. Setclock スクリプトの設定 175
7.5. Linux コンソールの設定 175
7.6. Sysklogd スクリプトの設定 178
7.7. /etc/inputrc ファイルの生成 178
7.8. Bash シェルの初期起動ファイル 179
7.9. LFS システムにおけるデバイスとモジュールの扱い 181
7.10. デバイスへのシンボリックリンクの生成 184
7.11. localnet スクリプトの設定 185
7.12. /etc/hosts ファイルの設定 186
7.13. ネットワークスクリプトの設定 186
8. LFS システムのブート設定 189
8.1. はじめに 189
8.2. /etc/fstab ファイルの生成 189
8.3. Linux-2.6.35.4 191
8.4. GRUB を用いたブートプロセスの設定 194
9. 作業終了 198
9.1. 作業終了 198
9.2. ユーザー登録 198
9.3. システムの再起動 198
9.4. 今度は何? 199
IV. 付録 200
A. 略語と用語 201
B. 謝辞 203
C. パッケージの依存関係 205
D. ブートスクリプトと sysconfig スクリプト version-20100627 215
D.1. /etc/rc.d/init.d/rc 215
D.2. /etc/rc.d/init.d/functions 216
D.3. /etc/rc.d/init.d/mountkernfs 229
D.4. /etc/rc.d/init.d/consolelog 230
D.5. /etc/rc.d/init.d/modules 231
D.6. /etc/rc.d/init.d/udev 232
D.7. /etc/rc.d/init.d/swap 234
D.8. /etc/rc.d/init.d/setclock 234
D.9. /etc/rc.d/init.d/checkfs 235
D.10. /etc/rc.d/init.d/mountfs 238
D.11. /etc/rc.d/init.d/udev_retry 238
D.12. /etc/rc.d/init.d/cleanfs 239
D.13. /etc/rc.d/init.d/console 241
D.14. /etc/rc.d/init.d/localnet 243
D.15. /etc/rc.d/init.d/sysctl 244

D.16. /etc/rc.d/init.d/sysklogd 244
D.17. /etc/rc.d/init.d/network 245
D.18. /etc/rc.d/init.d/sendsignals 247
D.19. /etc/rc.d/init.d/reboot 248
D.20. /etc/rc.d/init.d/halt 248
D.21. /etc/rc.d/init.d/template 249
D.22. /etc/sysconfig/rc 250
D.23. /etc/sysconfig/modules 250
D.24. /etc/sysconfig/createfiles 250
D.25. /etc/sysconfig/network-devices/ifup 251
D.26. /etc/sysconfig/network-devices/ifdown 252
D.27. /etc/sysconfig/network-devices/services/ipv4-static 254
D.28. /etc/sysconfig/network-devices/services/ipv4-static-route 255
E. Udev 設定ルール 258
E.1. 55-lfs.rules 258
F. LFS ライセンス 259
F.1. クリエイティブコモンズライセンス 259
F.2. MIT ライセンス (The MIT License) 262
項目別もくじ 263

序文

はしがき

私が Linux の学習と理解を深め始めたのは 1998年頃からです。 Linux ディストリビューションのインストールを行ったのはその時が初めてです。 そして即座に Linux 全般の考え方や原理について興味を抱くようになったのです。

何かの作業を完成させるには多くの方法があるものです。 同じことは Linux ディストリビューションについても言えます。 この数年の間に数多くのディストリビューションが登場しました。 あるものは今も存在し、あるものは他のものへと形を変え、そしてあるものは記憶の彼方へ追いやられたりもしました。 それぞれが利用者の求めに応じて、様々に異なる形でシステムを実現してきたわけです。 最終ゴールが同じものなのに、それを実現する方法がたくさんあるため、私は一つのディストリビューションにとらわれることが不要だと思い始めました。 Linux が登場する以前であれば、オペレーティングシステムに何か問題があったとしても、他に選択肢はなくそのオペレーティングシステムで満足する以外にありませんでした。 それはそういうものであって、好むと好まざるは関係がなかったのです。 それが Linux になって ”選ぶ”という考え方が出てきたわけです。 何かが気に入らなかったら、いくらでも変えたら良いし、そうすることがむしろ当たり前なのです。

数多くのディストリビューションを試してみましたが、これという1つに決定できるものがありませんでした。 個々のディストリビューションは優れたもので、それぞれを見てみれば正しいものです。 ただこれは正しいとか間違っているとかの問題ではなく、個人的な趣味の問題へと変化しているのです。 こうしたさまざまな状況を通じて明らかになってきたのは、私にとって完璧なシステムは1つもないということです。 そこで私は自分自身の Linux を作り出して、自分の好みを満足させるものを目指したのです。

本当に自分自身のシステムを作り出すため、私はすべてをソースコードからコンパイルすることを目指し、コンパイル済のバイナリパッケージは使わないことにしました。 この 「完璧な」 Linux システムは、他のシステムが持つ弱点を克服し、逆にすべての強力さを合わせ持つものです。 当初は気の遠くなる思いがしていましたが、そのアイデアは今も持ち続けています。

パッケージが相互に依存している状況やコンパイル時にエラーが発生するなどを順に整理していく中で、私はカスタムメイドの Linux を作り出したのです。 この Linux は今日ある他の Linux と比べても、十分な機能を有し十分に扱いやすいものとなっています。 これは私自身が作り出したものです。 いろいろなものを自分で組み立てていくのは楽しいものです。 後は個々のソフトウェアまでも自分で作り出せれば、もっと楽しいものになるのでしょうが、それは次の目標とします。

私の求める目標や作業経験を他の Linux コミュニティの方々とも共有する中で、私の Linux への挑戦は絶えることなく続いていくことを実感しています。 このようなカスタムメイドの Linux システムを作り出せば、独自の仕様や要求を満たすことができるのはもちろんですが、さらにはプログラマーやシステム管理者の Linux 知識を引き伸ばす絶好の機会となります。 壮大なこの意欲こそが Linux From Scratch プロジェクト　誕生の理由なのです。

Linux From Scratch ブックは関連プロジェクトの中心に位置するものです。 皆さんご自身のシステムを構築するために必要となる基礎的な手順を提供します。 本書が示すのは正常動作するシステム作りのための雛形となる手順ですので、皆さんが望んでいる形を作り出すために手順を変えていくことは自由です。 それこそ、本プロジェクトの重要な特徴でもあります。 そうしたとしても手順を踏み外すものではありません。我々は皆さんが挑戦する旅を応援します。

あなたの LFS システム作りが素晴らしいひとときとなりますように。 そしてあなた自身のシステムを持つ楽しみとなりますように。

--
Gerard Beekmans
gerard@linuxfromscratch.org

対象読者

本書を読む理由は様々にあると思いますが、よく挙がってくる質問として以下があります。 「既にある Linux をダウンロードしてインストールすれば良いのに、 どうして苦労してまで手作業で Linux を構築しようとするのか。」

本プロジェクトを提供する最大の理由は Linux システムがどのようにして動作しているのか、これを学ぶためのお手伝いをすることです。 LFS システムを構築してみれば、様々なものが連携し依存しながら動作している様子を知ることができます。 そうした経験をした人であれば Linux システムを自分の望む形に作りかえる手法も身につけることができます。

LFS の重要な利点として、他の Linux システムに依存することなく、システムをより適切に制御できる点が挙げられます。 LFS システムではあなたが運転台に立って、システムのあらゆる側面への指示を下していきます。

さらに非常にコンパクトな Linux システムを作る方法も身につけられます。 通常の Linux ディストリビューションを用いる場合、多くのプログラムをインストールすることになりますが、たいていのプログラムは使わないものですし、その内容もよく分からないものです。 それらのプログラムはハードウェアリソースを無駄に占有することになります。今日のハードドライブや CPU のことを考えたら、リソース消費は大したことはないと思うかもしれません。 しかし問題がなくなったとしても、サイズの制限だけは気にかける必要があることでしょう。 例えばブータブル CD、USB スティック、組み込みシステムなどのことを思い浮かべてください。 そういったものに対して LFS は有用なものとなるでしょう。

カスタマイズした Linux システムを構築するもう一つの利点として、セキュリティがあります。 ソースコードからコンパイルしてシステムを構築するということは、あらゆることを制御する権限を有することになり、セキュリティパッチは望みどおりに適用できます。 他の人がセキュリティホールを修正しバイナリパッケージを提供するのを待つ必要がなくなるということです。 他の人がパッチとバイナリパッケージを提供してくれたとしても、それが本当に正しく構築され、問題を解決してくれているかどうかは、調べてみなければ分からないわけですから。

Linux From Scratch の最終目標は、実用的で完全で、基盤となるシステムを構築することです。 Linux システムを一から作り出すつもりのない方は、本書から得られるものはないかもしれません。

LFS を構築する理由は様々ですから、すべてを列記することはできません。 学習こそ、理由を突き詰める最大最良の手段です。 LFS 構築作業の経験を積むことによって、情報や知識を通じてもたらされる意義が十二分に理解できるはずです。

LFS が対象とする CPU アーキテクチャ

LFS が対象としている CPU アーキテクチャは 32ビットインテル CPU が主となります。 LFS システムの構築に初めて取りかかる方は、おそらくこのアーキテクチャを用いることでしょう。 32ビットアーキテクチャは Linux システムが最も広くサポートしているもので、このアーキテクチャなら、オープンソースも製品ソフトウェアも互換性があります。

本書の作業手順は、多少の変更を加えれば Power PC や 64ビット AMD/インテル CPU でも動作することは検証されています。 その CPU を使ったシステムをビルドするには、これ以降の数ページで説明している条件以外に必要となることがあります。 LFS システムそのものや Ubuntu、Red Hat/Fedora、SuSE などのディストリビューションをホストとするわけですが、それは 64ビットシステムである必要があるということです。 ホストが 64ビット AMD/インテルによるシステムであったとしても 32ビットシステムは問題なくインストールできます。

64ビットシステムにて明らかなことをここに記しておきます。 32ビットシステムに比べると、実行プログラムのサイズは多少大きくなり、実行速度は若干速くなります。 例えば Core2Duo CPU をベースとするシステム上に、LFS 6.5 をビルドしてみたところ、以下のような情報が得られました。

アーキテクチャ	ビルド時間	ビルドサイズ
32ビット	198.5 分	648 MB
64ビット	190.6 分	709 MB

ご存知かと思いますが 64ビットによってビルドを行っても、 32ビットのときのビルドに比べて 4% 早くなるだけで 9% は大きなものになります。 つまり 64ビットシステムによって得られることは比較的小さいということです。 もちろん 4GB 以上のRAM を利用していたり、4GB を超えるデータを取り扱いたいならば、 64ビットシステムを用いるメリットが大きいのは間違いありません。

LFS の手順に従って作り出す 64ビットシステムは、”純粋な”64ビットシステムと言えます。 つまりそのシステムは 64ビット実行モジュールのみをサポートするということです。 ”複数のライブラリ”によるシステムをビルドするのなら、多くのアプリケーションを二度ビルドしなければなりません。 一度は 32ビット用であり、一度は 64ビット用です。 現時点にて本書はこの点をサポートしませんが、後々のリリースに向けて検討中です。 さしあたりそのような応用的なトピックに関しては Cross Linux From Scratch プロジェクトを参照してください。

最後に 64ビットシステムについてもう一つ述べておきます。 パッケージの中には現時点にて ”純粋な”64ビットシステム上でビルドできないものがあり、あるいは特別なビルド手順を必要とするものがあります。 一般的に言えば、そのようなパッケージには 32ビット固有のアセンブリ言語の命令が含まれるからであり、 だから 64ビットシステムでのビルドに失敗するということです。 例としては Beyond Linux From Scratch (BLFS) にある Xorg ドライバの一部分などです。 このような問題はたいていは解消していくことができますが、中には特別なビルド手順やパッチを要するものとなるかもしれません。

LFS と各種標準

LFS の構成は出来る限り Linux の各種標準に従うようにしています。 主な標準は以下のものです。

- The Single UNIX Specification Version 3 (POSIX).（登録操作が必要です。無料。）
- Filesystem Hierarchy Standard (FHS)
- Linux Standard Base (LSB) Core Specification 4.0

LSB はさらに以下の５つの標準から構成されます。 コア (Core)、C++、デスクトップ (Desktop)、ランタイム言語 (Runtime Languages)、印刷 (Printing) です。 また一般的な要求事項に加えて、アーキテクチャに固有の要求事項もあります。 LFS では前節にて示したように、各アーキテクチャに適合することを目指します。

注記

LSB の要求に対しては異論のある方も多いでしょう。 LSB を定義するのは、私有ソフトウェア (proprietary software) をインストールした場合に、要求事項を満たしたシステム上にて問題なく動作することを目指すためです。 LFS はソースコードから構築するシステムですから、どのパッケージを利用するかをユーザー自身が完全に制御できます。 また LSB にて要求されているパッケージであっても、インストールしない選択をとることもできます。

LFS の構築にあたっては LSB に適合していることを確認するテスト (certifications tests) をクリアするように構築することも可能です。 ただし LFS の範囲外にあるパッケージ類を追加しなければ実現できません。 そのような追加パッケージ類については BLFS にて導入手順を説明しています。

LFS 提供のパッケージで LSB 要求に従うもの

LSB コア:	Bash, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux-ng, Zlib
LSB C++:	Gcc
LSB デスクトップ:	なし
LSB ランタイム言語:	Perl
LSB 印刷:	なし
LSB マルチメディア:	なし

BLFS 提供のパッケージで LSB 要求に従うもの

LSB コア:	Bc, Cpio, Ed, Fcrontab, PAM, Sendmail（あるいは Postfix または Exim）
LSB C++:	なし
LSB デスクトップ:	ATK, Cairo, Desktop-file-utils, Freetype, Fontconfig, Glib2, GTK+2, Icon-naming-utils, Libjpeg, Libpng, Libxml2, MesaLib, Pango, Qt3, Qt4, Xorg
LSB ランタイム言語:	Python
LSB 印刷:	CUPS
LSB マルチメディア:	Alsa 関連ライブラリ, NSPR, NSS, OpenSSL, Java

LFS, BLFS で提供しないパッケージで LSB 要求に従うもの

LSB コア:	At, Batch, Install_initd, Lsb_release, Remove_initd, Test
LSB C++:	なし
LSB デスクトップ:	なし
LSB ランタイム言語:	なし
LSB 印刷:	なし

LSB マルチメディア: Xdg-utils

各パッケージを用いる理由

既に説明しているように LFS が目指すのは、完成した形での実用可能な基盤システムを構築することです。 LFS に含まれるパッケージ群は、パッケージの個々を構築していくために必要となるものばかりです。 そこからは最小限の基盤となるシステムを作り出します。 そしてユーザーの望みに応じて、より完璧なシステムへと拡張していくものとなります。 LFS は極小システムを意味するわけではありません。 厳密には必要のないパッケージであっても、重要なものとして含んでいるものもあります。 以下に示す一覧は、本書内の各パッケージの採用根拠について説明するものです。

- Autoconf

  このパッケージは、以下に示すようなシェルスクリプトを生成するプログラムを提供します。 つまり開発者が意図しているテンプレートに基づいて、ソースコードを自動的に設定する（configure する）ためのシェルスクリプトです。 特定のパッケージのビルド方法に変更があった場合は、パッケージ再構築を行うことになるため、その場合に本パッケージが必要となります。

- Automake

  このパッケージは、テンプレートとなるファイルから Makefile を生成するためのプログラムを提供します。 特定のパッケージのビルド方法に変更があった場合は、パッケージ再構築を行うことになるため、その場合に本パッケージが必要となります。

- Bash

  このパッケージは、システムとのインターフェースを実現する Bourne シェルを提供し、LSB コア要件を満たします。 他のシェルを選ばずにこれを選ぶのは、一般的に多用されていることと、基本的なシェル関数においての拡張性が高いからです。

- Binutils

  このパッケージは、リンカ、アセンブラのような、オブジェクトファイルを取り扱うプログラムを提供します。 各プログラムは LFS における他のパッケージをコンパイルするために必要となり、さらに LFS にて示される以外のパッケージでも必要となります。

- Bison

  このパッケージは yacc（Yet Another Compiler Compiler）の GNU バージョンを提供します。 LFS において利用するプログラムの中に、これを必要とするものがあります。

- Bzip2

  このパッケージは、ファイルの圧縮、伸張（解凍）を行うプログラムを提供します。 これは LFS パッケージの多くを伸張（解凍）するために必要です。

- Coreutils

  このパッケージは、ファイルやディレクトリを参照あるいは操作するための基本的なプログラムを数多く提供します。 各プログラムはコマンドラインからの実行によりファイル制御を行うために必要です。 また LFS におけるパッケージのインストールに必要となります。

- DejaGNU

  このパッケージは、他のプログラムをテストするフレームワークを提供します。 これは一時的なツールチェーンプログラムをインストールする際にだけ必要となります。

- Diffutils

  このパッケージは、ファイルやディレクトリ間の差異を表示するプログラムを提供します。 各プログラムはパッチを生成するために利用されます。 したがってパッケージのビルド時に利用されることが多々あります。

- Expect

  このパッケージは、スクリプトで作られた対話型プログラムを通じて、他のプログラムとのやりとりを行うプログラムを提供します。 通常は他のパッケージをテストするために利用します。 本書では一時的なツールチェーンの構築時にしかインストールしません。

- E2fsprogs

  このパッケージは ext2, ext3, ext4 の各ファイルシステムを取り扱うユーティリティを提供します。 各ファイルシステムは Linux がサポートする一般的なものであり、十分なテストが実施されているものです。

- File

このパッケージは、指定されたファイルの種類を判別するユーティリティプログラムを提供します。 他のパッケージにおいて、ビルド時にこれを必要とするものもあります。

- Findutils

  このパッケージは、ファイルシステム上のファイルを検索するプログラムを提供します。 これは他のパッケージにて、ビルド時のスクリプトにおいて利用されています。

- Flex

  このパッケージは、テキスト内の特定パターンの認識プログラムを生成するユーティリティを提供します。 これは lex（字句解析; lexical analyzer）プログラムの GNU 版です。 LFS 内の他のパッケージの中にこれを必要としているものがあります。

- Gawk

  このパッケージはテキストファイルを操作するプログラムを提供します。 プログラムは GNU 版の awk（Aho-Weinberg-Kernighan）です。 これは他のパッケージにて、ビルド時のスクリプトにおいて利用されています。

- Gcc

  これは GNU コンパイラコレクションパッケージです。 C コンパイラと C++ コンパイラを含みます。また LFS ではビルドしないコンパイラも含まれています。

- GDBM

  このパッケージは GNU データベースマネージャライブラリを提供します。 LFS が扱う Man-DB パッケージがこれを利用しています。

- Gettext

  このパッケージは、各種パッケージが国際化を行うために利用するユーティリティやライブラリを提供します。

- Glibc

  このパッケージは C ライブラリです。Linux 上のプログラムはこれがなければ動作させることができません。

- GMP

  このパッケージは数値演算ライブラリを提供するもので、任意精度演算（arbitrary precision arithmetic）についての有用な関数を含みます。 これは GCC をビルドするために必要です。

- Grep

  このパッケージはファイル内を検索するプログラムを提供します。 これは他のパッケージにて、ビルド時のスクリプトにおいて利用されています。

- Groff

  このパッケージは、テキストを処理し整形するプログラムをいくつか提供します。 重要なものプログラムとして man ページを生成するものを含みます。

- GRUB

  これは Grand Unified Boot Loader です。 ブートローダーとして利用可能なものの中でも、これが最も柔軟性に富むものです。

- Gzip

  このパッケージは、ファイルの圧縮と伸張（解凍）を行うプログラムを提供します。 LFS において、パッケージを伸張（解凍）するために必要です。

- Iana-etc

  このパッケージは、ネットワークサービスやプロトコルに関するデータを提供します。 ネットワーク機能を適切に有効なものとするために、これが必要です。

- Inetutils

  このパッケージは、ネットワーク管理を行う基本的なプログラム類を提供します。

- IProute2

  このパッケージは、IPv4、IPv6 による基本的な、あるいは拡張したネットワーク制御を行うプログラムを提供します。 IPv6 への対応があることから、よく使われてきたネットワークツールパッケージ（net-tools）に変わって採用されました。

- Kbd

このパッケージは、米国以外のキーボードに対してのキーテーブルファイルやキーボードユーティリティを提供します。 また端末上のフォントも提供します。

- Less

このパッケージはテキストファイルを表示する機能を提供するものであり、表示中にスクロールを可能とします。 また Man-DB において man ページを表示する際にも利用されます。

- Libtool

このパッケージは GNU の汎用的なライブラリに対してのサポートスクリプトを提供します。 これは、複雑な共有ライブラリの取り扱いを単純なものとし、移植性に優れた一貫した方法を提供します。 LFS パッケージのテストスイートにおいて必要となります。

- Linux Kernel

このパッケージは ”オペレーティングシステム” であり GNU/Linux 環境における Linux です。

- M4

このパッケージは汎用的なテキストマクロプロセッサであり、他のプログラムを構築するツールとして利用することができます。

- Make

このパッケージは、パッケージ構築を指示するプログラムを提供します。 LFS におけるパッケージでは、ほぼすべてにおいて必要となります。

- Man-DB

このパッケージは man ページを検索し表示するプログラムを提供します。 man パッケージではなく本パッケージを採用しているのは、その方が国際化機能が優れているためです。 このパッケージは man プログラムを提供しています。

- Man-pages

このパッケージは Linux の基本的な man ページを提供します。

- Module-Init-Tools

このパッケージは Linux カーネルモジュールを管理するのに必要なプログラムを提供します。

- MPC

このパッケージは複素数演算のための関数を提供します。 GCC パッケージがこれを必要としています。

- MPFR

このパッケージは倍精度演算（multiple precision）の関数を提供します。 GCC パッケージがこれを必要としています。

- Ncurses

このパッケージは、端末に依存せず文字キャラクタを取り扱うライブラリを提供します。 メニュー表示時のカーソル制御を実現する際に利用されます。 LFS の他のパッケージでは、たいていはこれを必要としています。

- Patch

このパッケージは、パッチ ファイルの適用により、特定のファイルを修正したり新規生成したりするためのプログラムを提供します。 パッチファイルは diff プログラムにより生成されます。 LFS パッケージの中には、構築時にこれを必要とするものがあります。

- Perl

このパッケージは、ランタイムに利用されるインタープリタ言語 PERL を提供します。 LFS の他のパッケージでは、インストール時やテストスイートの実行時にこれを必要とするものがあります。

- Pkg-config

このパッケージは、configure や make を行う際に、ビルドツールに対してインクルードパスやライブラリパスを受け渡すツールプログラムを提供します。 LFS パッケージでは、ほとんどがこれを必要としています。

- Procps

このパッケージは、プロセスの監視を行うプログラムを提供します。 システム管理にはこのパッケージが必要となります。 また LFS ブートスクリプトではこれを利用しています。

- Psmisc

このパッケージは、実行中のプロセスに関する情報を表示するプログラムを提供します。 システム管理にはこのパッケージが必要となります。

- Readline

  このパッケージは、コマンドライン上での入力編集や履歴管理を行うライブラリを提供します。 これは Bash が利用しています。

- Sed

  このパッケージは、テキストの編集を、テキストエディタを用いることなく可能とします。 LFS パッケージにおける configure スクリプトは、たいていこれを必要としています。

- Shadow

  このパッケージは、セキュアな手法によりパスワード制御を行うプログラムを提供します。

- Sysklogd

  このパッケージは、システムメッセージログを扱うプログラムを提供します。 例えばカーネルが出力するログや、デーモンプロセスが異常発生時に出力するログなどです。

- Sysvinit

  このパッケージは init プログラムを提供します。 これは Linux システム上のすべてのプロセスの基点となるものです。

- Tar

  このパッケージは、アーカイブや圧縮機能を提供するもので LFS が扱うすべてのパッケージにて利用されています。

- Tcl

  このパッケージはツールコマンド言語（Tool Command Language）を提供します。 LFS が扱うパッケージにてテストスイートの実行に必要となります。 これは一時的なツールチェーンの構築時にのみインストールします。

- Texinfo

  このパッケージは Info ページに関しての入出力や変換を行うプログラムを提供します。 LFS が扱うパッケージのインストール時には、たいてい利用されます。

- Udev

  このパッケージはデバイスノードの動的生成を行うプログラムを提供します。 /dev ディレクトリに、デバイスを静的にいくつも作り出す方法を取らないためのものです。

- Util-linux-ng

  このパッケージは数多くのユーティリティプログラムを提供します。 その中には、ファイルシステムやコンソール、パーティション、メッセージなどを取り扱うユーティリティがあります。

- Vim

  このパッケージはテキストエディタを提供します。 これを採用しているのは、従来の vi エディタとの互換性があり、しかも数々の有用な機能を提供するものだからです。 テキストエディタは個人により好みはさまざまですから、もし別のエディタを利用したいなら、そちらを用いても構いません。

- Zlib

  このパッケージは、圧縮や解凍の機能を提供するもので、他のプログラムがこれを利用しています。

必要な知識

LFS システムの構築作業は決して単純なものではありません。 ある程度の Unix システム管理の知識が必要です。 問題を解決したり、説明されているコマンドを正しく実行することが求められます。 ファイルやディレクトリのコピー、それらの表示確認、カレントディレクトリの変更、といったことは最低でも知っていなければなりません。 さらに Linux の各種ソフトウェアを使ったりインストールしたりする知識も必要です。

LFS ブックでは、最低でも そのようなスキルがあることを前提としていますので、数多くの LFS サポートフォーラムは、ひょっとすると役に立たないかもしれません。 フォーラムにおいて基本的な知識を尋ねたとしたら、誰も回答してくれないでしょう。 そうするよりも LFS に取り掛かる前に以下のような情報をよく読んでください。

LFS システムの構築作業に入る前に、以下の「ハウツー」を読むことをお勧めします。

- ソフトウェア構築のハウツー（Software-Building-HOWTO）http://www.tldp.org/HOWTO/Software-Building-HOWTO.html

これは Linux 上において 「一般的な」 Unix ソフトウェアを構築してインストールする方法を総合的に説明しています。 だいぶ前に書かれたものですが、ソフトウェアのビルドとインストールを行うために必要となる基本的な方法が程よくまとめられています。

- Linux ユーザーガイド (The Linux Users's Guide) http://www.linuxhq.com/guides/LUG/guide.html

  このガイドには Linux ソフトウェアの利用方法が分類され説明されています。 若干古いものですが内容に間違いはありません。

- 基本的な事前ヒント情報 (The Essential Pre-Reading Hint) http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

  これは Linux 初心者に向けて書かれた LFS ヒントです。 ここには非常に多くの有用なトピックへのリンクがあります。 LFS を構築しようとするなら、これらのヒントに示されている内容は、出来るだけ多く理解しておくことが必要でしょう。

ホストシステム要件

ホストシステムには以下に示すソフトウェアが必要であり、それぞれに示されているバージョン以降である必要があります。 最近の Linux ディストリビューションを利用するなら、あまり問題にはならないはずです。 ディストリビューションによっては、ソフトウェアのヘッダファイル群を別パッケージとして提供しているものが多々あります。 例えば「<パッケージ名>-devel」 であったり 「<パッケージ名>-dev」 といった具合です。 お使いのディストリビューションがそのような提供の仕方をしている場合は、それらもインストールしてください。

各パッケージにて、示しているバージョンより古いものでも動作するかもしれませんが、テストは行っていません。

- Bash-3.2 (/bin/sh が bash に対するシンボリックリンクまたはハードリンクである必要があります。)
- Binutils-2.17 (2.20.1 以上のバージョンは、テストしていないためお勧めしません。)
- Bison-2.3 (/usr/bin/yacc が bison へのリンクか、bison を実行するためのスクリプトである必要があります。)
- Bzip2-1.0.4
- Coreutils-6.9
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-3.1.5 (/usr/bin/awk が gawk へのリンクである必要があります。)
- Gcc-4.1.2 (4.5.1 以上のバージョンは、テストしていないためお勧めしません。)
- Glibc-2.5.1 (2.12.1 以上のバージョンは、テストしていないためお勧めしません。)
- Grep-2.5.1a
- Gzip-1.3.12
- Linux Kernel-2.6.22.5 (GCC-4.1.2 以上でコンパイルされたもの)

  カーネルのバージョンを指定しているのは、第6章にて glibc をビルドする際にバージョンを指定するからであり、開発者の勧めに従うためです。

  ホストシステムのカーネルバージョンが 2.6.22.5 より古い場合、あるいはカーネルをビルドした際の GCC のバージョンが 4.1.2 よりも古い場合は、ここに示した条件に合致するカーネルに置き換えることが必要です。 これを実施するには２つの方法があります。 お使いの Linux システムのベンダーが 2.6.22.5 以上のバージョンのカーネルを提供しているかを調べることです。 提供していれば、それをインストールします。 もしそれがない場合や、あったとしてもそれをインストールしたくない場合、カーネルをご自身でコンパイルする必要があります。 カーネルのコンパイルと (ホストシステムが GRUB を利用しているとして) ブートローダの設定方法については 第8章 を参照してください。
- M4-1.4.10
- Make-3.81
- Patch-2.5.4
- Perl-5.8.8
- Sed-4.1.5
- Tar-1.18
- Texinfo-4.9

上で示しているシンボリックリンクは、本書の説明を通じて LFS を構築するために必要となるものです。 シンボリックリンクが別のソフトウェア (例えば dash や mawk) を指し示している場合でもうまく動作するかもしれません。 しかしそれらに対して LFS 開発チームはテストを行っていませんしサポート対象としていません。 そのような状況に対しては作業手順の変更が必要となり、特定のパッケージに対しては追加のパッチを要するかもしれません。

ホストシステムに、上のソフトウェアの適切なバージョンがインストールされているかどうか、またコンパイルが適切に行えるかどうかは、以下のスクリプトを実行して確認することができます。

```
cat > version-check.sh << "EOF"
#!/bin/bash
export LC_ALL=C

# Simple script to list version numbers of critical development tools

bash --version | head -n1 | cut -d" " -f2-4
echo "/bin/sh -> 'readlink -f /bin/sh'"
echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-
bison --version | head -n1
if [ -e /usr/bin/yacc ];
  then echo "/usr/bin/yacc -> 'readlink -f /usr/bin/yacc'";
  else echo "yacc not found"; fi
bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1
if [ -e /usr/bin/awk ];
  then echo "/usr/bin/awk -> 'readlink -f /usr/bin/awk'";
  else echo "awk not found"; fi
gcc --version | head -n1
/lib/libc.so.6 | head -n1 | cut -d"," -f1
grep --version | head -n1
gzip --version | head -n1
cat /proc/version
m4 --version | head -n1
make --version | head -n1
patch --version | head -n1
echo Perl 'perl -V:version'
sed --version | head -n1
tar --version | head -n1
echo "Texinfo: 'makeinfo --version | head -n1'"
echo 'main(){}' > dummy.c && gcc -o dummy dummy.c
if [ -x dummy ]; then echo "Compilation OK";
  else echo "Compilation failed"; fi
rm -f dummy.c dummy

EOF

bash version-check.sh
```

本書の表記

本書では、特定の表記を用いて分かりやすく説明を行っていきます。 ここでは Linux From Scratch ブックを通じて利用する表記例を示します。

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

この表記は特に説明がない限りは、そのまま入力するテキストを示しています。 またコマンドの説明を行うために用いる場合もあります。

場合によっては、1行で表現される内容を複数行に分けているものがあります。 その場合は各行の終わりにバックスラッシュ（あるいは円記号）を表記しています。

```
CC="gcc -B/usr/bin/" ../binutils-2.18/configure \
  --prefix=/tools --disable-nls --disable-werror
```

バックスラッシュ（または円記号）のすぐ後ろには改行文字がきます。 そこに余計な空白文字やタブ文字があると、おかしな結果となるかもしれないため注意してください。

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

上の表記は固定幅フォントで示されており、たいていはコマンド入力の結果として出力される端末メッセージを示しています。 あるいは /etc/ld.so.conf といったファイル名を示すのに利用する場合もあります。

Emphasis

上の表記は様々な意図で用いています。 特に重要な説明内容やポイントを表します。

http://www.linuxfromscratch.org/

この表記は LFS コミュニティ内や外部サイトへのハイパーリンクを示します。 そこには「ハウツー」やダウンロードサイトなどが含まれます。

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

上の表記は設定ファイル類を生成する際に示します。 1行目のコマンドは $LFS/etc/group というファイルを生成することを指示しています。 そのファイルへは2行目以降 EOF が記述されるまでのテキストが出力されます。 したがってこの表記は通常そのままタイプ入力します。

<REPLACED TEXT>

上の表記は入力するテキストを仮に表現したものです。 これをそのまま入力するものではないため、コピー、ペースト操作で貼り付けないでください。

[OPTIONAL TEXT]

上の表記は入力しなくてもよいオプションを示しています。

passwd(5)

上の表記はマニュアルページ（man ページ）を参照するものです。 カッコ内の数字は man の内部で定められている特定のセクションを表しています。 例えば passwd コマンドには2つのマニュアルページがあります。 LFS のインストールに従った場合、2つのマニュアルページは /usr/share/man/man1/passwd.1 と /usr/share/man/man5/passwd.5 に配置されます。 passwd(5) という表記は /usr/share/man/man5/passwd.5 を参照することを意味します。 man passwd という入力に対しては 「passwd」 という語に合致する最初のマニュアルページが表示されるものであり /usr/share/man/man1/passwd.1 が表示されることになります。 特定のマニュアルページを見たい場合は man 5 passwd といった入力を行う必要があります。 マニュアルページが複数あるケースはまれですので、普通は man **<プログラム名>** と入力するだけで十分です。

本書の構成

本書は以下の部から構成されます。

第 I 部 - はじめに

第I部では LFS 構築作業を進めるための重要事項について説明します。 また本書のさまざまな情報についても説明します。

第 II 部 - ビルド作業のための準備

第II部では、パーティションの生成、パッケージのダウンロード、一時的なツールのコンパイルといった、システム構築の準備作業について説明します。

第 III 部 - LFSシステムの構築

第III部では LFS システムの構築作業を順に説明していきます。 そこでは全パッケージのコンパイルとインストール、ブートスクリプトの設定、カーネルのインストールを行います。 出来上がる Linux システムをベースとして、他のソフトウェアを必要に応じて導入し、このシステムを拡張していくことができます。 本書の終わりには、インストール対象のプログラム、ライブラリ、あるいは重要なファイル類についてのさくいんも示します。

正誤情報

LFS システムを構築するためのソフトウェアは日々拡張され更新されています。 LFS ブックがリリースされた後に、セキュリティフィックスやバグフィックスが公開されているかもしれません。 本版にて説明するパッケージや作業手順に対して、セキュリティフィックスやバグフィックス等が必要かどうか、ビルド作業を行う前に http://www.linuxfromscratch.org/lfs/errata/6.7/ を確認してください。 そして LFS ビルド作業を進めながら、対応する節においての変更を確認し適用してください。

日本語訳について

日本語訳情報

本節はオリジナルの LFS ブックにはないものです。 日本語訳に関する情報を示すために設けました。

はじめに

本書は LFS ブック 6.7 の日本語版-20100923 です。 オリジナルの LFS ブックと同様に DocBook を用いて構築しています。

日本語版の提供について

日本語版 LFS ブックは SourceForge.jp 内に開発の場を設け http://sourceforge.jp/projects/lfsbookja/ にて「LFSブック日本語版」 のプロジェクト名で提供するものです。

HTML ファイル類や日本語化のために構築しているソース類について、あるいはそれらの取り扱い（ライセンス）については上記サイトを参照してください。

日本語版の生成について

日本語版 LFS ブックの生成は、以下のようにして行っています。

- そもそも LFS ブックのソースは、LFS のサイト http://www.linuxfromscratch.org/ において、Static 版として公開されていると同時に Subversion により、日々開発更新されているソース（XMLソース）が公開されています。日本語版はその XML ソースに基づいて作成しています。
- XML ソースは DocBook XML DTD の書式に従ったファイル形式です。 日本語版では、ソースに記述された原文を日本語訳文に変えて、同様の処理により生成しています。 ソース内に含まれる INSTALL ファイルには、処理に必要となるツール類の詳細が示されています。 それらのツール類はすべて BLFS にてインストールする対象となっていますので、興味のある方は参照してください。
- 日本語訳にあたっては、原文にて 「地の文」 として表現されている文章を日本語化しています。 逆に各手順におけるコマンド説明（四角の枠囲いで示されている箇所）は、日本語化の対象とはしていません。 コマンド類や設定記述が英単語で行われるわけですから、これは当たり前のことです。 ただ厳密に言えば、その四角の枠囲いの中でシェルのコメント書きが含まれる場合があり、これは日本語化せずそのまま表記しています。

日本語版における注意点

日本語版 LFS ブックを参照頂く際には、以下の点に注意してください。

- 本ページの冒頭にあるように、原文にはない記述は 「日本語訳情報」 として枠囲い文章で示すことにします。
- 訳者は Linux に関する知識を隅から隅まで熟知しているわけではありません。 したがってパッケージのことや Linux の仕組みに関して説明されている原文の、真の意味が捉えられず、原文だけを頼りに訳出している箇所もあります。 もし誤訳、不十分な訳出、意味不明な箇所に気づかれた場合は、是非ご指摘、ご教示をお願いしたいと思います。
- 日本語訳にて表記しているカタカナ用語について触れておきます。 特に語末に長音符号がつく（あるいはつかない）用語です。 このことに関しては訳者なりに捉えているところがあるのですが、詳述は省略します。 例えば 「ユーザー（user)」 という用語は語末に長音符号をつけるべきと考えます。 一方 「コンピュータ（computer)」 という用語は、情報関連その他の分野では長音符号をつけない慣用があるものの、昨今これをつけるような流れもあり情勢が変わりつつあります。 このように用語表記については、大いに ”ゆれ” があるため、訳者なりに取り決めて表記することにしています。 なじみの表記とは若干異なるものが現れるかもしれませんが、ご了承いただきたいと思います。

第I部 はじめに

第1章 はじめに

1.1. LFS をどうやって作るか

LFS システムは、既にインストールされている Linux ディストリビューション（Debian、Mandriva、Red Hat、SUSE など）を利用して構築していきます。 この既存の Linux システム（ホスト）は、LFS 構築のために様々なプログラム類を利用する基盤となります。 プログラム類とはコンパイラ、リンカ、シェルなどです。 したがってそのディストリビューションのインストール時には 「開発（development）」 オプションを選択し、それらのプログラム類が利用できるようにしておく必要があります。

コンピュータ内にインストールされているディストリビューションを利用するのではなく、 Linux From Scratch LiveCD、あるいは他に提供されている LiveCD を利用することもできます。 LFS LiveCD はホストシステムとして利用することができ、本書の手順を実施するための必要なツール類がすべて含まれます。 LiveCD の開発は思うように進んでいませんが、ホストシステムとして利用することが可能です。 現時点にて本書によるビルド作業を進めるなら 「-nosrc」 や 「-min」 という名称が含まれている版を用いてください。 LFS LiveCD の詳細や LiveCD ダウンロード方法については http://www.linuxfromscratch.org/livecd/ を参照してください。

注記

LFS LiveCD は最近のハードウェア環境において、うまく動作しないかもしれません。 ブートに失敗したり SATA ハードドライブのようなデバイス検出に失敗したりすることがあります。

第2章 では、新しく構築する Linux のためのパーティションとファイルシステムの生成方法について説明します。 そのパーティション上にて LFS システムをコンパイルしインストールします。 第3章 では LFS 構築に必要となるパッケージとパッチについて説明します。 これらをダウンロードして新たなファイルシステム内に保存します。 第4章 では作業環境の準備について述べています。 この章では重要な説明を行っていますので 第5章 以降に進む前に是非注意して読んでください。

第5章 では数多くのパッケージをインストールします。 これらは基本的な開発ツール（ツールチェーン）を構成するものであり 第6章 において最終的なシステムを構築するために利用します。 パッケージの中には自分自身を循環的に必要とするような依存関係を持つものがあります。 例えばコンパイラをコンパイルするためにはコンパイラが必要となります。

第5章 ではツールチェーンの第1回めの構築方法を示します。 そこではまず Binutils と GCC を構築します。（第1回めと表現しているということは、つまりこれら2つのパッケージは後に再構築します。）次に C ライブラリである Glibc を構築します。 Glibc は第1回めのツールチェーンを用いてコンパイルされます。 そして第2回めのツールチェーン構築を行います。 この時のツールチェーンは新たに構築した Glibc をリンクします。 それ以降の 第5章 に示すパッケージは第2回めのツールチェーンプログラムを用いて構築します。 上の作業をすべて終えたら LFS のインストール作業はもはやホストディストリビューションに依存しません。 ただし作動させるカーネルだけは使い続けます。

ホストシステムのツール類から新しいシステムを切り離していくこの手順は、やり過ぎのように見えるかもしれません。 5.2. 「ツールチェーンの技術的情報」 にて詳細に説明しているので参照してください。

第6章 にて LFS システムが出来上がります。 chroot（ルートをチェンジする）プログラムを使って仮想的な環境に入り LFS パーティション内のディレクトリをルートディレクトリとしてシェルを起動します。 これは LFS パーティションをルートパーティションとするシステム再起動と同じことです。 ただ実際にはシステムを再起動はしません。 再起動できるシステムとするためにはもう少し作業を必要としますし、この時点ではまだそれが必要ではないので chroot を行う方法を取ります。 chroot を使うメリットは、LFS 構築作業にあたって引き続きホストシステムを利用できることです。 パッケージをコンパイルしている最中には、通常どおり別の作業を行うことができます。

インストールの仕上げとして 第7章 にて LFS ブートスクリプトを設定し、 第8章 にてカーネルとブートローダを設定します。 第9章 では LFS システム構築経験を踏まえて、その先に進むための情報を示します。 本書に示す作業をすべて実施すれば、新たな LFS システムを起動することが出来ます。

上はごく簡単な説明にすぎません。 各作業の詳細はこれ以降の章やパッケージの説明を参照してください。 内容が難しいと思っていても、それは徐々に理解していけるはずです。 読者の皆さんには、是非 LFS アドベンチャーに挑んで頂きたいと思います。

1.2. 前版からの変更点

以下に示すのは前版から変更されているパッケージです。

アップグレード:

- Autoconf 2.67

- Binutils 2.20.1
- Bison 2.4.3
- Diffutils 3.0
- E2fsprogs 1.41.12
- Expect 5.44.1.15
- Gawk 3.1.8
- GCC 4.5.1
- Gettext 0.18.1.1
- GMP 5.0.1
- Grep 2.6.3
- GRUB 1.98
- Inetutils 1.8
- IPRoute2 2.6.35
- Kbd 1.15.2
- LFS-Bootscripts 20100627
- Libtool 2.2.10
- Linux 2.6.35.4
- M4 1.4.14
- Make 3.82
- Man-DB 2.5.7
- Man-pages 3.25
- Module-Init-Tools 3.12
- MPC 0.8.2
- MPFR 3.0.0
- Perl 5.12.1
- Pkg-config 0.25
- Psmisc 22.12
- Tar 1.23
- Udev 161
- Util-Linux-NG 2.18
- Zlib 1.2.5

追加:

- bash-4.1-fixes-2.patch
- bzip2-1.0.5-version_fixes-1.patch
- expect-5.44.1.15-no_tk-1.patch
- glibc-2.12.1-makefile_fix-1.patch
- linux-2.6.35.4-mm_locking-1.patch
- MPC-0.8.2
- udev-161-testfiles.tar.bz2

削除:

- expect-5.43.0-spawn-1.patch
- expect-5.43.0-tcl_8.5.5_fix-1.patch
- gettext-0.17-upstream_fixes-2.patch
- grep-2.5.4-debian_fixes-1.patch
- make-3.81-upstream_fixes-1.patch
- perl-5.10.1-utf8-1.patch
- vim-7.2-fixes-5.patch

1.3. 変更履歴

本書は Linux From Scratch ブック、バージョン 6.7 です。 本書が 6ヶ月以上更新されていなければ、より新しい版が公開されているはずです。以下のミラーサイトを確認してください。 http://www.linuxfromscratch.org/mirrors.html

以下は前版からの変更点を示したものです。

変更履歴：

- 2010-09-18
  - [bdubbs] - LFS-6.7 リリース。
  - [matthew] - GCC テストにてスタックサイズを増やす手順を追加。
- 2010-09-17
  - [bdubbs] - 'テスト スイート'の表現を統一。 #2756 を Fix に。
  - [bdubbs] - Psmisc でのシンボリックリンク作成を削除。 Sysvinit のインストール時に上書きされなかったら、init スクリプトが正常動作しなくなるため。
  - [bdubbs] - grub.conf の修正。 #2748 を Fix に。
- 2010-09-06
  - [matthew] - GCC ２回目の処理にて LD_LIBRARY_PATH の利用をやめ --disable-libgomp を利用することに。 configure スクリプトの失敗を回避するため。
- 2010-09-03
  - [bdubbs] - Perl のビルド時に -Duseshrplib を追加して共有ライブラリをビルドすることに。 Perl モジュールの中にはこれを必要とするものがあるため。 #2745 を Fix に。
  - [bdubbs] - Pkg-config における M4 スクリプトを sed コマンドを使って修正。 #2746 を Fix に。
- 2010-08-31
  - [bdubbs] - linux-2.6.35.4 へのアップグレード。 #2743 を Fix に。
- 2010-08-17
  - [matthew] - iproute2 のバグを修正。 指摘してくれた Gilles Espinasse に感謝。
- 2010-08-16
  - [bdubbs] - vim-7.3 へのアップグレード。 #2721 を Fix に。
- 2010-08-15
  - [bdubbs] - Zlib のビルドオプション -mstackrealign にて、Intel アーキテクチャ以外ではビルドエラーが発生する旨を追記。 #2733 を Fix に。
  - [matthew] - ip route get にて出力生成に失敗するバグを修正。 Thomas Trepl による報告、および Andrew Benton による修正に感謝。
  - [matthew] - Udev-161 へのアップグレード。 #2739 を Fix に。
  - [matthew] - Linux-2.6.35.2 へのアップグレード。 #2737 を Fix に。
  - [matthew] - Netfs ブートスクリプトが利用する fuser プログラムを root ファイルシステムに移動。 #2736 を Fix に。
  - [matthew] - peekfd を x86 と同様に x86_64 でもビルドすることに。 #2734 を Fix に。
- 2010-08-10
  - [bdubbs] - Sysvinit によってインストールされるプログラム一覧に fstab-decode を追加。
- 2010-08-08
  - [matthew] - Bison-2.4.3 へのアップグレード。 #2732 を Fix に。
  - [matthew] - IPRoute2-2.6.35 へのアップグレード。 #2731 を Fix に。
  - [matthew] - Glibc-2.12.1 へのアップグレード。 #2730 を Fix に。
- 2010-08-03
  - [matthew] - Autoconf-2.67 へのアップグレード。 #2729 を Fix に。
  - [matthew] - 第5章での GCC ビルドにおいて、ホストシステムの CLooG, PPL ライブラリにリンクされないようにパラメータを追加。 それらは第6章でも存在しない。 #2723 を Fix に。

- [matthew] - Linux-2.6.35 へのアップグレード。 #2728 を Fix に。
- [matthew] - GCC-4.5.1 へのアップグレード。 #2727 を Fix に。
- [matthew] - Make-3.82 へのアップグレード。 #2726 を Fix に。

- 2010-07-26
  - [matthew] - GCC のビルドにあたって、Zlib はバンドルされているものではなく、既にインストールされているものを利用するように。 #2718 を Fix に。
  - [matthew] - Psmisc-22.12 へのアップグレード。 #2717 を Fix に。
  - [matthew] - Udev-160 へのアップグレード。 #2711 を Fix に。
  - [matthew] - Linux-2.6.34.1 へのアップグレード。 #2709 を Fix に。
  - [matthew] - Autoconf-2.66 へのアップグレード。 #2705 を Fix に。
- 2010-07-18
  - [bdubbs] - パッケージを同時並行でビルドする問題点について説明修正。 #2712 を Fix に。
  - [bdubbs] - GRUB と Glibc の依存パッケージを更新。 パッチを提供してくれた splotz90 に感謝。 #2716 を Fix に。
  - [bdubbs] - 仮想ファイルシステムのマウント時におけるデバイス生成処理に関して明確化。 #2715 を Fix に。
- 2010-07-07
  - [matthew] - GRUB のブートディスク作成方法について修正。 #2706 を Fix に。 報告および修正をしてくれた Sebastian Plotz に感謝。
- 2010-07-04
  - [robert] - Util-linux tarball の URL を修正。
- 2010-07-02
  - [bdubbs] - SBU 値においてテストスイートに要する時間は、第6章では含めるが第5章では含めないものとする。パッチ提供してくれた littlebat に感謝。 #2702 を Fix に。
  - [bdubbs] - grub.cfg ファイルについての注意書きを充足させる。
  - [bdubbs] - Util-linix-ng-2.18 へのアップグレード。 #2681 を Fix に。
- 2010-07-01
  - [ken] - 記述修正。 #2701 を Fix に。
- 2010-06-27
  - [bdubbs] - udev-158 へのアップグレード。 また udev-testfiles 配布の追加と make check における手順追加。 #2692 と #2700 を Fix に。
  - [bdubbs] - sysvinit-2.88dsf へのアップグレード。 #2677 を Fix に。
  - [bdubbs] - Zlib の configure にて CFLAGS オプションを追加。 これは GCC-4.5 にてビルドする際に、セグメンテーションフォールトが発生することを回避するため。 #2691 を Fix に。
  - [bdubbs] - Glibc のタイムアウトエラーについて加筆。 また TIMEOUTFACTOR により解決可能である旨を追記。 #2683 を Fix に。
  - [bdubbs] - 各種パッケージにてインストールプログラムの一覧を更新。 パッチ提供してくれた Chris Staub に感謝。 #2678 を Fix に。
- 2010-06-22
  - [matthew] - Man-Pages-3.25 へのアップグレード。 #2695 を Fix に。
  - [matthew] - MPFR-3.0.0 へのアップグレード。 #2687 を Fix に。
- 2010-06-21
  - [matthew] - Udev-157 へのアップグレード。 #2676 を Fix に。
  - [matthew] - Gettext-0.18.1.1 へのアップグレード。 #2686 を Fix に。
  - [matthew] - Libtool-2.2.10 へのアップグレード。 #2690 を Fix に。
  - [matthew] - Expect-5.44.1.15 へのアップグレード。 #2689 を Fix に。
- 2010-06-19
  - [bdubbs] - DejaGNU に対する累積的なパッチを追加。 #2684 を Fix に。
- 2010-06-18

- [bdubbs] - Module-Init-Tools-3.12 へのアップグレード。 #2675 と #2688 を Fix に。
- [bdubbs] - Linux-2.6.34 以前に対してのバグを一時的に sed コマンドにより解消。 この変更は、後にカーネル最新版に組み入れられた際に削除予定。 #2662 を Fix に。

- 2010-06-16
  - [bdubbs] - 第5章、第6章の Glibc の構築にて指定するカーネルのバージョンを、ホスト要件に合致するように更新。
- 2010-06-14
  - [robert] - Udev のインストール時に rmdir に -v オプション追加。
- 2010-06-01
  - [bdubbs] - LFS-6.3 にて構築していたホスト要件を更新。
- 2010-05-29
  - [matthew] - Udev-156 へのアップグレード。 #2671 を Fix に。
  - [matthew] - Pkg-config-0.25 へのアップグレード。 #2670 を Fix に。
  - [matthew] - Glibc-2.11.2 へのアップグレード。 #2669 を Fix に。
  - [matthew] - Gettext-0.18 へのアップグレード。 #2660 を Fix に。
- 2010-05-26
  - [bdubbs] - ビルド手順を明確にするための記述を追加。
- 2010-05-23
  - [ken] - Gmp パッケージにおける ABI の記述（第6章）を修正。 これは 32 ビットに対するものであることを明示。 #2648 を Fix に。
  - [bdubbs] - Man-DB にて内部処理に関わる問題を修正するパッチを追加。 パッチ提供をしてくれた William Immendorf に感謝。 #2652 を Fix に。
  - [bdubbs] - 各パッケージに、インストールディレクトリの情報を追加。 パッチ提供をしてくれた Chris Staub に感謝。 #2655 を Fix に。
- 2010-05-21
  - [matthew] - Udev-154 にて生成される空のドキュメントディレクトリを削除。
  - [matthew] - IPRoute2-2.6.34 へのアップグレード。 #2668 を Fix に。
  - [matthew] - E2fsprogs-1.41.12 へのアップグレード。 #2667 を Fix に。
  - [matthew] - Perl-5.12.1 へのアップグレード。 #2666 を Fix に。
  - [matthew] - Bash に対する最新のアップストリームによるパッチを追加。 #2665 を Fix に。
  - [matthew] - MPC-0.8.2 へのアップグレード。 #2664 を Fix に。
  - [matthew] - Inetutils-1.8 へのアップグレード。 #2663 を Fix に。
  - [matthew] - Gawk-3.1.8 へのアップグレード。 #2659 を Fix に。
  - [matthew] - Man-DB のドキュメントを、バージョン番号つきのディレクトリにインストールする。 #2658 を Fix に。
  - [matthew] - Diffutils-3.0 へのアップグレード。 #2656 を Fix に。
  - [matthew] - MPFR においてドキュメントインストールの手順を修正。 パッチ提供をしてくれた Chris Staub に感謝。 #2655 を Fix に。
  - [matthew] - Coreutils-8.5 へのアップグレード。 #2643 を Fix に。
  - [matthew] - Udev-154 へのアップグレード。 #2639 を Fix に。
  - [matthew] - Zlib-1.2.5 へのアップグレード。 #2638 を Fix に。
  - [matthew] - Linux-2.6.34 へのアップグレード。 #2628 を Fix に。
  - [matthew] - Bzip2 にて適切でないバージョン番号を修正するためのパッチを追加。 パッチ提供してくれた Jeremy Huntwork（LightCube OS）に感謝。 #2624 を Fix に。
- 2010-05-06
  - [bdubbs] - 第6章の GCC 構築手順にて sed コマンド操作を削除。 GCC-4.5.0 では解消されたため。 #2653 を Fix に。
- 2010-05-03

- [bdubbs] - Tar パッケージにてバッファオーバーフローを修正するパッチを追加。 この修正は gcc-4.5 以上にてビルドする際に必要。 #2610 を Fix に。
- [bdubbs] - Diffutils における新たなテストスイートの手順を追加。 Chris Staub に感謝。 #2650 を Fix に。
- [bdubbs] - 第5章の Gawk、第6章の Patch にて若干の説明修正。 Chris Staub に感謝。 #2649 を Fix に。
- [bdubbs] - 第5章および「各パッケージを用いる理由」のページにて若干の説明修正。 パッチ提供をしてくれた Chris Staub に感謝。 #2644 を Fix に。
- [bdubbs] - 第6章における Binutils にて、警告メッセージをなくすために File パッケージを第5章に追加。 #2640 を Fix に。

- 2010-04-20
  - [matthew] GCC-4.5.0 へのアップグレード。 新たな依存パッケージ MPC の追加。 #2636 を Fix に。
  - [matthew] Perl-5.12.0 へのアップグレード。 #2635 を Fix に。
  - [matthew] Kbd-1.15.2 へのアップグレード。 #2634 を Fix に。
  - [matthew] Udev にて、使われていないパラメータ --docdir を削除。 #2633 を Fix に。
  - [matthew] Psmisc-22.11 へのアップグレード。 #2631 を Fix に。
  - [matthew] killall を /bin へ移動するように。 これは、/usr パーティションが別に分かれている場合で、かつ Sysvinit がインストールされていない状況でもブートを可能とするため。 #2622 を Fix に。
  - [matthew] Grep-2.6.3 へのアップグレード。 #2621 を Fix に。
  - [matthew] Bash にてアップストリームの修正 001 ～ 005 を追加。 #2620 を Fix に。
  - [matthew] Vim の修正パッチを削除。 このパッチは既に古くなっていることと、アップストリームの開発サイクルにて適用済と思われるため。 #2597 を Fix に。
- 2010-04-12
  - [bdubbs] - Zlib のビルド手順にて、.pc ファイルを適切に利用すること、およびすべてのライブラリを適切なディレクトリにインストールするように修正。 Chris Staub 氏に感謝する。 #2630 を Fix に。
- 2010-03-27
  - [matthew] Grep-2.6.1 へのアップグレード。 #2617 を Fix に。
  - [matthew] Util-Linux-NG-2.17.2 へのアップグレード。 #2616 を Fix に。
  - [matthew] Bison-2.4.2 へのアップグレード。 #2615 を Fix に。
- 2010-03-20
  - [bdubbs] - 最新の tar パッケージにて発生するエラーを、sed コマンドを使って修正。 これは開発元による修正。
- 2010-03-18
  - [matthew] Linux-2.6.33.1 へのアップグレード。 #2608 を Fix に。
  - [matthew] E2fsprogs-1.41.11 へのアップグレード。 #2607 を Fix に。
  - [matthew] Zlib-1.2.4 へのアップグレード。 #2606 を Fix に。
  - [matthew] Tar-1.23 へのアップグレード。 #2603 を Fix に。
  - [matthew] Grub-1.98 へのアップグレード。 #2602 を Fix に。
  - [matthew] Bash のテストスイートの実行に際して不要なコマンドを削除。 #2601 を Fix に。
  - [matthew] Binutils-2.20.1 へのアップグレード。 #2599 を Fix に。
- 2010-03-01
  - [matthew] Man-pages-3.24 へのアップグレード。 #2596 を Fix に。
  - [matthew] M4-1.4.14 へのアップグレード。 #2594 を Fix に。
  - [matthew] IPRoute2-2.6.33 へのアップグレード。 #2592 を Fix に。
  - [matthew] Linux-2.6.33 へのアップグレード。 #2587 を Fix に。
  - [matthew] Man-DB-2.5.7 へのアップグレード。 #2583 を Fix に。
  - [matthew] Util-Linux-NG-2.17.1 へのアップグレード。 #2581 を Fix に。
  - [matthew] Diffutils-2.9 へのアップグレード。 #2577 を Fix に。 アップストリームにより i18n 向けのパッチが不採用となったため削除。
  - [matthew] GMP-5.0.1 へのアップグレード。 #2572 を Fix に。

- [matthew] LFS-6.6 向けの整理。

1.4. 変更履歴（日本語版）

ここに示すのは LFS ブック 6.7 日本語版（バージョン 20100923）の変更履歴です。

日本語訳情報

本節はオリジナルの LFS ブックにはないものです。 LFS ブック日本語版の変更履歴を示すために設けています。

「r1234」という表記は、オリジナル XML ソースファイルの Subversion 管理下でのリビジョン番号を意味します。

変更履歴：

- 2010-09-23
  - [matsuand] - src/chapter06/sysvinit.ch: 訳出漏れを訂正。
  - [matsuand] - src/chapter05/generalinstructions.ch: r9381 対応。日本語訳は変わらず。
- 2010-09-19
  - [matsuand] - 6.7 リリース。

1.5. 情報源

1.5.1. FAQ

LFS システムの構築作業中にエラー発生したり、疑問を抱いたり、あるいは本書の誤記を発見した場合、まず手始めに http://www.linuxfromscratch.org/faq/ に示されている「よく尋ねられる質問」(Frequently Asked Questions; FAQ) を参照してください。

1.5.2. メーリングリスト

linuxfromscratch.org サーバーでは、LFS 開発プロジェクトのために多くのメーリングリストを立ち上げています。 このメーリングリストは主となる開発用とは別に、サポート用のものもあります。 FAQ だけでは問題解決に至らなかった場合に、次の手としてメーリングリストを検索する以下のサイトを参照してください。 http://www.linuxfromscratch.org/search.html

これ以外に、投稿の方法、アーカイブの配置場所などに関しては http://www.linuxfromscratch.org/mail.html を参照してください。

1.5.3. IRC

LFS コミュニティのメンバーの中には、インターネットリレーチャット（Internet Relay Chat; IRC）によるサポートを行っている者もいます。 ここに対して質問を挙げる場合は、FAQ やメーリングリストに同様の質問や答えがないかどうかを必ず確認してください。 IRC は irc.linuxfromscratch.org において、チャネル名 #LFS-support により提供しています。

1.5.4. ミラーサイト

LFS プロジェクトは世界中にミラーサイトがあります。 これらを使えばウェブサイト参照やパッケージのダウンロードがより便利に利用できます。 以下のサイトによりミラーサイトの情報を確認してください。 http://www.linuxfromscratch.org/mirrors.html

1.5.5. 連絡先

質問やコメントは（上に示した）メーリングリストを活用してください。

1.6. ヘルプ

本書に基づく作業の中で問題が発生したり疑問が生まれた場合は http://www.linuxfromscratch.org/faq/#generalfaq にある FAQ のページを確認してください。 質問への回答が示されているかもしれません。 そこに回答が示されていなかったなら、問題の本質部分を見極めてください。 トラブルシューティングとして以下のヒントが有用かもしれません。 http://www.linuxfromscratch.org/hints/downloads/files/errors.txt

FAQ では問題解決ができない場合、メーリングリスト http://www.linuxfromscratch.org/search.html を検索してください。

我々のサイトにはメーリングリストやチャットを通じての情報提供を行う LFS コミュニティがあります。（詳細は 1.5. 「情報源」 を参照してください。）我々は日々数多くのご質問を頂くのですが、たいていの質問は FAQ やメーリングリストを調べてみれば容易に答えが分かるものばかりです。 したがって我々が最大限の支援を提供できるよう、ある程度の問題はご自身で解決するようにしてください。 そうして頂くことで、我々はもっと特殊な状況に対するサポートを手厚く行っていくことができるからです。 いくら調べても解決に至らず、お問い合わせ頂く場合は、以下に示すように十分な情報を提示してください。

1.6.1. 特記事項

問題が発生し問い合わせをする場合には、以下に示す基本的な情報を含めてください。

- お使いの LFS ブックのバージョン。（本書の場合 6.7）
- LFS 構築に用いたホスト Linux のディストリビューションとそのバージョン。
- vii. 「ホストシステム要件」 [xv] の出力結果。
- 問題が発生したパッケージまたは本書内の該当の章または節。
- 問題となったエラーメッセージや状況に対する詳細な情報。
- 本書どおりに作業しているか、逸脱していないかの情報。

注記

本書の作業手順を逸脱していたとしても、 我々がお手伝いしないわけではありません 。 つまるところ LFS は個人的な趣味によって構築されるものです。 本書の手順とは異なるやり方を正確に説明してください。 そうすれば内容の評価、原因究明が容易になります。

1.6.2. Configure スクリプトの問題

configure スクリプトの実行時に何か問題が発生した時は config.log ファイルを確認してみてください。 configure スクリプトの実行中に、端末画面に表示されないエラーが、このファイルに出力されているかもしれません。 問合せを行う際には 該当する 行を示してください。

1.6.3. コンパイル時の問題

コンパイル時に問題が発生した場合は、端末画面への出力とともに、数々のファイルの内容も問題解決の糸口となります。 configure スクリプトと make コマンドの実行によって端末画面に出力される情報は重要です。 問い合わせの際には、出力されるすべての情報を示す必要はありませんが、関連する情報は十分に含めてください。 以下に示すのは make コマンドの実行時に出力される情報を切り出してみた例です。

```
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
```

たいていの方は、上のような場合に終わりの数行しか示してくれません。

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

問題を解決するにはあまりに不十分な情報です。 そんな情報だけでは 「何かがオカしい結果となった」 ことは分かっても 「なぜオカしい結果となった」 のかが分からないからです。 上に示したのは、十分な情報を提供して頂くべきであることを例示したものであり、実行されたコマンドや関連するエラーメッセージが十分に含んだ例となっています。

インターネット上に、問い合わせを行う方法を示した優れた文章があります。 http://catb.org/~esr/faqs/smart-questions.html　この文章に示される内容やヒントを参考にして、より確実に回答が得られるよう心がけてください。

第II部 ビルド作業のための準備

第2章 新しいパーティションの準備

2.1. はじめに

この章では LFS システムをインストールするパーティションを準備します。 パーティションを生成しファイルシステムを構築した上で、これをマウントします。

2.2. 新しいパーティションの生成

どのようなオペレーティングシステムでも同じことが言えますが、 本システムでもインストール先は専用のパーティションを用いることにします。 LFS システムを構築していくには、利用可能な空のパーティションか、 あるいはパーティション化していないものをパーティションとして生成して利用することにします。

最小限のシステムであれば 1.3 GB 程度のディスク容量があれば十分です。 これだけあればパッケージやソースの収容に十分で、そこでコンパイル作業を行っていくことができます。 しかし主要なシステムとして LFS を構築するなら、さらにソフトウェアをインストールすることになるはずなので、さらに 2~3 GBの容量が必要となります。 LFS システムそのものがそれだけの容量を要するわけではありません。 これだけの容量は十分なテンポラリ領域のために必要となるものです。 パッケージをインストールした後はテンポラリ領域は開放されますが、コンパイルの間は多くの領域を利用します。

コンパイル処理において十分なランダムアクセスメモリ（Random Access Memory; RAM）を確保できるとは限りませんので、スワップ（swap）領域をパーティションとして設けるのが普通です。 この領域へは利用頻度が低いデータを移すことで、アクティブな処理プロセスがより多くのメモリを確保できるようにカーネルが制御します。 swap パーティションは、LFS システムのものとホストシステムのものを共有することもできます。 その場合は新しいパーティションを作る必要はありません。

ディスクのパーティション生成は cfdisk コマンドや fdisk コマンドを使って行います。 コマンドラインオプションにはパーティションを生成するハードディスク名を指定します。 例えば IDE（Integrated Drive Electronics）ディスクであれば /dev/hda といったものになります。 そして Linux ネイティブパーティションと、必要なら swap パーティションを生成します。 プログラムの利用方法について不明であれば cfdisk(8) や fdisk(8) を参照してください。

新しく生成したパーティションの名前を覚えておいてください。（例えば hda5 など。）本書ではこのパーティションを LFS パーティションとして説明していきます。 また swap パーティションの名前も忘れないでください。 これらの名前は、後に生成する /etc/fstab ファイルに記述するために必要となります。

2.2.1. パーティションに関するその他の問題

LFS メーリングリストにてパーティションに関する有用情報を望む声をよく聞きます。 これは個人の趣味にもよる極めて主観的なものです。 既存ディストリビューションが採用しているデフォルトのパーティションサイズと言えば、たいていはスワップパーティションを小容量で配置した上で、そのドライブ内の残容量すべてのサイズを割り当てています。 このようなサイズ設定は LFS では最適ではありません。その理由はいくつかあります。 そのようにしてしまうと、複数のディストリビューションの導入時や LFS 構築時に、柔軟さを欠き、構築がしにくくなります。 バックアップを取る際にも無用な時間を要し、ファイルシステム上にて不適当なファイル配置を生み出すため、余計なディスク消費を発生させます。

2.2.1.1. ルートパーティション

ルートパーティション（これを /root ディレクトリと混同しないでください）は 10 GB もあれば、どんなシステムであっても妥当なところでしょう。 それだけあれば LFS 構築も、また BLFS においてもおそらく十分なはずです。 実験的に複数パーティションを設けるとしても、これだけのサイズは必要です。

2.2.1.2. スワップパーティション

既存のディストリビューションは、たいていはスワップパーティションを自動的に生成します。 一般にスワップパーティションのサイズは、物理 RAM サイズの二倍の容量とすることが推奨されています。 しかしそれだけの容量はほとんど必要ありません。 ディスク容量が限られているなら、スワップパーティションの容量を 2GB 程度に抑えておいて、ディスクスワップがどれだけ発生するかを確認してみてください。

スワップは好ましいことではありません。 一般にスワップが発生しているかどうかは、ディスクアクセスの様子やコマンド実行時にシステムがどのように反応するかを見てみれば分かります。 例えば 5GB くらいのファイルを編集するといった極端なコマンド実行を行ってみて、スワップが起きるかどうかを確認することが重要です。 スワップがごく普通に発生するようであれば、RAMを増設するのが適切です。

2.2.1.3. 有用なパーティション

この他にも、必要のないパーティションというものがいくつかあります。 しかしディスクレイアウトを取り決めるには考えておく必要があります。 以下に示すのは十分な説明ではありませんが、一つの目安として示すものです。

- /boot - 作成することが強く推奨されます。 カーネルやブート情報を収納するために利用するパーティションです。容量の大きなディスクの場合、ブート時に問題が発生することがあるので、これを回避するには、一つ目のディスクドライブの物理的に一番最初のパーティションを選びます。 パーティションサイズを 100MB とすればそれで十分です。
- /home - 作成することが強く推奨されます。 複数のディストリビューションや LFS の間で、ホームディレクトリおよびユーザー固有の設定を共有することができます。 パーティションサイズは、ある程度大きく取ることになりますが、利用可能なディスク残容量に依存します。
- /usr - /usr ディレクトリを別パーティションとして設けるのは、一般にはシンクライアント（thin client）向けサーバーやディスクレスワークステーションにおいて行われます。 普通 LFS では必要ありません。 5 GB くらいの容量があれば、たいていのアプリケーションをインストールするのに十分なものでしょう。
- /opt - このディレクトリは BLFS などにおいて、Gnome や KDE といった巨大なパッケージをいくつもインストールする際に活用されます。 /usr ディレクトリ以外にインストールする場合です。 これを別パーティションとするなら、一般的には 5 〜 10 GB 程度が適当でしょう。
- /tmp - /tmp ディレクトリを別パーティションとするのは普通は行いません。 ただしシンクライアント（thin client）では有効です。 別パーティションとする場合であっても、数GB程度あれば十分です。
- /usr/src - このパーティションは LFS のパッケージソースを収容し LFS ビルド工程にて共用するものとして有効に利用することができます。 さらに BLFS パッケージソースを収容しビルドする場所としても利用可能です。 30〜50GBくらいの容量があれば、十分なものです。

ブート時に自動的にパーティションをマウントしたい場合は /etc/fstab ファイルにて設定します。 パーティションの設定方法については 8.2. 「/etc/fstab ファイルの生成」 で説明しています。

2.3. ファイルシステムの生成

空のパーティションが準備できたのでファイルシステムを作ります。 Linux において広く用いられるファイルシステムは ext2（second extended file system）です。 より新しく大容量のハードディスクに対しては、ジャーナリングファイルシステムが一般的となりつつあります。 ext3（third extended file system）は ext2 の拡張として広く利用されるようになっています。 ext3 ではジャーナリング機能が追加され E2fsprogs ユーティリティとの互換性を持ちます。 本書では ext3 ファイルシステムを生成することにします。 他のファイルシステムの生成方法については http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html を参照してください。

LFS 用のパーティションに対して ext3 ファイルシステムを生成するために以下のコマンドを実行します。

```
mke2fs -jv /dev/<xxx>
```

<xxx> の部分は LFS パーティション名に合わせて置き換えてください。（本書の例では hda5 としています。）

注記

ホストとして利用する Linux ディストリビューションの中には、ファイルシステムを生成するツール（E2fsprogs）に特別な機能を実装しているものがあります。 第9章にて LFS システムをブートする際に、それらの機能が原因で問題が発生する場合があります。 そのような機能は LFS においてインストールする E2fsprogs ではサポートしていません。 おそらくは 「unsupported filesystem features, upgrade your e2fsprogs」（サポートされていないファイルシステムです。 e2fsprogs をアップグレードしてください）といったエラーメッセージが表示されるはずです。 ホストシステムが機能拡張しているかどうかを確認するには以下のコマンドを実行します。

```
debugfs -R feature /dev/<xxx>
```

コマンドの出力結果の中に has_journal、 ext_attr、 resize_inode、 dir_index、 filetype、 sparse_super、 large_file、 needs_recovery といったものとは異なるものが表示されていたら、あなたのホストシステムは機能拡張がなされていることを意味します。 後に問題となりますので、純粋な E2fsprogs パッケージをコンパイルし、これを用いて LFS パーティションのファイルシステムを再生成してください。

```
cd /tmp
tar -xzvf /path/to/sources/e2fsprogs-1.41.12.tar.gz
cd e2fsprogs-1.41.12
mkdir -v build
cd build
../configure
make #note that we intentionally don't 'make install' here!
./misc/mke2fs -jv /dev/<xxx>
cd /tmp
rm -rfv e2fsprogs-1.41.12
```

既に存在している swap パーティションを用いることにした場合は、初期化操作を行う必要はありません。 新しい swap パーティションを作成した場合は、以下のコマンドを実行して初期化を行う必要があります。

```
mkswap /dev/<yyy>
```

<yyy> の部分は swap パーティションの名に合わせて置き換えてください。

2.4. 新しいパーティションのマウント

ファイルシステムが生成できたら、パーティションをアクセスできるようにします。 これを行うためにはマウントポイントを定める必要があります。 本書ではファイルシステムを /mnt/lfs にマウントすることにします。このディレクトリは各自で取り決めて変えることもできます。

マウントポイントを定めたら、そのディレクトリを指し示すような環境変数 LFS を以下のようにして設定します。

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

次にマウントポイントを生成し、LFS ファイルシステムをマウントします。

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
```

<xxx> の部分は LFS パーティション名に合わせて置き換えてください。

LFS に対して複数のパーティションを用いる場合（例えば / と /usr が別パーティションである場合）は、以下を実行してそれぞれをマウントします。

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
mkdir -v $LFS/usr
mount -v -t ext3 /dev/<yyy> $LFS/usr
```

<xxx> や <yyy> の部分は、それぞれ適切なパーティション名に置き換えてください。

この新しいパーティションは特別な制限オプション（nosuid、nodev、noatime など）は設定せずにマウントします。 mount コマンドの実行時に引数を与えずに実行すれば、LFS パーティションがどのようなオプション設定によりマウントされているかが分かります。 もし nosuid、nodev、noatime といったオプションが設定されていたら、マウントし直してください。

swap パーティションを用いる場合は、swapon コマンドを使って利用可能にしてください。

```
/sbin/swapon -v /dev/<zzz>
```

<zzz> の部分は swap パーティション名に置き換えてください。

こうして動作環境が整いました。次はパッケージのダウンロードです。

第3章 パッケージとパッチ

3.1. はじめに

この章では基本的な Linux システム構築のためにダウンロードするべきパッケージの一覧を示します。 各パッケージのバージョンは動作が確認されているものを示しており、本書ではこれに基づいて説明します。 ここに示すバージョンよりも新しいものは使わないようお勧めします。 あるバージョンでビルドしたコマンドが、新しいバージョンでも動作する保証はないからです。 最新のパッケージの場合、何かの対処を要するかもしれません。 そのような対処方法は本書の開発版において開発され安定化が図られるかもしれません。

ダウンロードサイトは常にアクセス可能であるとは限りません。 本書が提供された後にダウンロードする場所が変更になっていたら Google (http://www.google.com/) を使って検索してみてください。 たいていのパッケージを見つけ出すことが出来るはずです。 それでも見つけられなかったら http://www.linuxfromscratch.org/lfs/packages.html#packages に示されている方法に従って入手してください。

ダウンロードしたパッケージやパッチは、ビルド作業を通じて常に利用可能な場所を選んで保存しておく必要があります。 またソース類を伸張してビルドを行うための作業ディレクトリも必要です。 そこで本書では $LFS/sources ディレクトリを用意し、ソースやパッチの保存場所とし、そこでビルドを行う作業ディレクトリとします。 このディレクトリにしておけば LFS パーティションに位置することから LFS ビルドを行う全工程において常に利用することが出来ます。

ダウンロードを行う前にまずはそのようなディレクトリを生成します。 root ユーザーとなって以下のコマンドを実行します。

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

このディレクトリには書き込み権限とスティッキーを与えます。 「スティッキー (Sticky)」 は複数ユーザーに対して書き込み権限が与えられても、削除については所有者しか実行出来ないようにします。 以下のコマンドによって書き込み権限とスティッキーを定めます。

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

パッケージとパッチのダウンロードを簡単に行う方法として wget-list を利用する方法があります。 これは wget の入力引数に指定し利用します。

日本語訳情報

本節にて wget-list のハイパーリンクが出てきますが、これは本来、拡張子を持たないファイル wget-list へのリンクです。 本書を Web サイト上に搭載した場合に MIME 設定 (その制約) によりアクセスが出来ないファイルとなってしまう可能性があります。 そこで本書では wget-list.txt のように拡張子 .txt をつけるようにしました。 なお別途公開している本書の tarball では wget-list と wget-list.txt を共に含めています。両者は全く同一内容です。

日本語訳情報

上の wget-list は、次節に示されているパッケージやパッチのダウンロード URL を一覧列記しています。これを wget とともに用いれば、必要なパッケージソースやパッチの一括取得ができるため大変便利です。ちなみに LFS ブック原版では wget の具体的な使い方が示されていませんが、単純には以下のようなコマンド実行になるでしょう。（wget-list ファイルを $LFS/sources ディレクトリにコピーして実行します。）

```
cd $LFS/sources
wget -N -i wget-list
```

LFS ブック原版では、それらの URL が主に米国サイトの URL となっています。 一方、日本に在住する日本の方であれば、例えば GNU のパッケージ類は国内に数多くのミラーサイトが存在するため、そちらから取得するのが適切でしょう。 これはネットワークリソースを利用する際のマナーとも言えるものです。 堅苦しい話をするつもりはありません。 国内サイトから入手することにすればダウンロード速度が断然早くなります。 メリットは大きいと思いますのでお勧めします。

国内から入手可能なものは国内から入手することを目指し、訳者は以下の手順により wget-list を書き換えて利用しています。 一例として国内には理化学研究所のサイト（ftp.riken.jp）があります。 そこでは GNU パッケージ類がミラー提供されています。 そこで wget-list にて ftp.gnu.org を指し示している URL を ftp.riken.jp に置き換えます。 また Linux カーネルの入手先（www.kernel.org）についても理化学研究所より入手可能ですので、これも置き換えます。

```
cp -pv wget-list{,.orig}
sed -e 's|http://ftp\.gnu\.org/gnu/|http://ftp.riken.jp/GNU/ftp/gnu/|g' \
    -e 's|http://www\.kernel\.org/pub/linux/|http://ftp.riken.jp/Linux/kernel.org/linux/|g' \
       wget-list.orig > wget-list
```

注意する点として各パッケージが更新されたばかりの日付では、国内ミラーサイトへの同期、反映が間に合わず、ソース類が存在しないことが考えられます。 その場合には上の方法はすんなりとは実現できません。オリジナルの URL を用いるしかありません。

上記はあくまで一例です。しかもすべてのパッケージについて、国内サイトからの入手となるわけではありません。 ただし上記を行うだけでも、大半のパッケージは国内サイトを向くことになります。

上記にて国内のミラーサイトは、ネットワーク的に ”より近い” ものを選んでください。 またミラーサイトのディレクトリ構成はサイトによって変わります。必要に応じてコマンドを書き換えてください。 さらに上記の sed による一括置換は、パッケージやソースの今後の更新状況によっては提供 URL が変わり、 wget-list のすべての URL が正しいものにはならない可能性がありますから十分注意してください。 ダウンロードできなかった場合は、上記の sed コマンドを工夫するか、手作業にて wget-list を書き換えてください。

3.2. 全パッケージ

以下に示すパッケージをダウンロードするなどしてすべて入手してください。

- Autoconf (2.67) - 1,338 KB:

ホームページ: http://www.gnu.org/software/autoconf/
ダウンロード: http://people.redhat.com/eblake/autoconf/autoconf-2.67.tar.bz2
MD5 sum: 3fbf92eb8eaca1e0d33dff9710edb5f0

- Automake (1.11.1) - 1,042 KB:

ホームページ: http://www.gnu.org/software/automake/
ダウンロード: http://ftp.gnu.org/gnu/automake/automake-1.11.1.tar.bz2
MD5 sum: c2972c4d9b3e29c03d5f2af86249876f

- Bash (4.1) - 6,444 KB:

ホームページ: http://www.gnu.org/software/bash/
ダウンロード: http://ftp.gnu.org/gnu/bash/bash-4.1.tar.gz
MD5 sum: 9800d8724815fd84994d9be65ab5e7b8

- Binutils (2.20.1) - 17,091 KB:

ホームページ: http://sources.redhat.com/binutils/
ダウンロード: http://ftp.gnu.org/gnu/binutils/binutils-2.20.1.tar.bz2
MD5 sum: 9cdfb9d6ec0578c166d3beae5e15c4e5

• Bison (2.4.3) - 1,614 KB:
ホームページ: http://www.gnu.org/software/bison/
ダウンロード: http://ftp.gnu.org/gnu/bison/bison-2.4.3.tar.bz2
MD5 sum: c1d3ea81bc370dbd43b6f0b2cd21287e

• Bzip2 (1.0.5) - 822 KB:
ホームページ: http://www.bzip.org/
ダウンロード: http://www.bzip.org/1.0.5/bzip2-1.0.5.tar.gz
MD5 sum: 3c15a0c8d1d3ee1c46a1634d00617b1a

• Coreutils (8.5) - 10,489 KB:
ホームページ: http://www.gnu.org/software/coreutils/
ダウンロード: http://ftp.gnu.org/gnu/coreutils/coreutils-8.5.tar.gz
MD5 sum: c1ffe586d001e87d66cd80c4536ee823

• DejaGNU (1.4.4) - 1,055 KB:
ホームページ: http://www.gnu.org/software/dejagnu/
ダウンロード: http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.4.4.tar.gz
MD5 sum: 053f18fd5d00873de365413cab17a666

• Diffutils (3.0) - 1,781 KB:
ホームページ: http://www.gnu.org/software/diffutils/
ダウンロード: http://ftp.gnu.org/gnu/diffutils/diffutils-3.0.tar.gz
MD5 sum: 684aaba1baab743a2a90e52162ff07da

• E2fsprogs (1.41.12) - 4,391 KB:
ホームページ: http://e2fsprogs.sourceforge.net/
ダウンロード: http://prdownloads.sourceforge.net/e2fsprogs/e2fsprogs-1.41.12.tar.gz
MD5 sum: 1b24a21fc0c2381ef420961cbfec733f

• Expect (5.44.1.15) - 534 KB:
ホームページ: http://expect.sourceforge.net/
ダウンロード: http://prdownloads.sourceforge.net/expect/expect-5.44.1.15.tar.bz2
MD5 sum: 9307bbf67e19125036ce34544a78dadf

• File (5.04) - 607 KB:
ホームページ: http://www.darwinsys.com/file/
ダウンロード: ftp://ftp.astron.com/pub/file/file-5.04.tar.gz
MD5 sum: accade81ff1cc774904b47c72c8aeea0

注記

File パッケージ (5.04) は上記の場所から入手できなくなっているかもしれません。 これはサイト管理者が、新バージョンのリリースと同時に古いバージョンを削除することがあるためです。 適切なバージョンをダウンロードするためには、以下に示す別のサイトを参照してください。 http://www.linuxfromscratch.org/lfs/download.html#ftp

• Findutils (4.4.2) - 2,100 KB:
ホームページ: http://www.gnu.org/software/findutils/
ダウンロード: http://ftp.gnu.org/gnu/findutils/findutils-4.4.2.tar.gz
MD5 sum: 351cc4adb07d54877fa15f75fb77d39f

• Flex (2.5.35) - 1,227 KB:
ホームページ: http://flex.sourceforge.net
ダウンロード: http://prdownloads.sourceforge.net/flex/flex-2.5.35.tar.bz2
MD5 sum: 10714e50cea54dc7a227e3eddcd44d57

• Gawk (3.1.8) - 1,938 KB:
ホームページ: http://www.gnu.org/software/gawk/
ダウンロード: http://ftp.gnu.org/gnu/gawk/gawk-3.1.8.tar.bz2
MD5 sum: 52b41c6c4418b3226dfb8f82076193bb

• GCC (4.5.1) - 64,572 KB:
ホームページ: http://gcc.gnu.org/
ダウンロード: http://ftp.gnu.org/gnu/gcc/gcc-4.5.1/gcc-4.5.1.tar.bz2
MD5 sum: 48231a8e33ed6e058a341c53b819de1a

• GDBM (1.8.3) - 223 KB:
ホームページ: http://www.gnu.org/software/gdbm/
ダウンロード: http://ftp.gnu.org/gnu/gdbm/gdbm-1.8.3.tar.gz
MD5 sum: 1d1b1d5c0245b1c00aff92da751e9aa1

• Gettext (0.18.1.1) - 14,785 KB:
ホームページ: http://www.gnu.org/software/gettext/
ダウンロード: http://ftp.gnu.org/gnu/gettext/gettext-0.18.1.1.tar.gz
MD5 sum: 3dd55b952826d2b32f51308f2f91aa89

• Glibc (2.12.1) - 15,300 KB:
ホームページ: http://www.gnu.org/software/libc/
ダウンロード: http://ftp.gnu.org/gnu/glibc/glibc-2.12.1.tar.bz2
MD5 sum: be0ea9e587f08c87604fe10a91f72afd

• GMP (5.0.1) - 1,959 KB:
ホームページ: http://www.gnu.org/software/gmp/
ダウンロード: http://ftp.gnu.org/gnu/gmp/gmp-5.0.1.tar.bz2
MD5 sum: 6bac6df75c192a13419dfd71d19240a7

• Grep (2.6.3) - 1,280 KB:
ホームページ: http://www.gnu.org/software/grep/
ダウンロード: http://ftp.gnu.org/gnu/grep/grep-2.6.3.tar.gz
MD5 sum: 3095b57837b312f087c0680559de7f13

• Groff (1.20.1) - 3,510 KB:
ホームページ: http://www.gnu.org/software/groff/
ダウンロード: http://ftp.gnu.org/gnu/groff/groff-1.20.1.tar.gz
MD5 sum: 48fa768dd6fdeb7968041dd5ae8e2b02

• GRUB (1.98) - 2,392 KB:
ホームページ: http://www.gnu.org/software/grub/
ダウンロード: ftp://alpha.gnu.org/gnu/grub/grub-1.98.tar.gz
MD5 sum: c0bcf60e524739bb64e3a2d4e3732a59

• Gzip (1.4) - 886 KB:
ホームページ: http://www.gzip.org/
ダウンロード: http://ftp.gnu.org/gnu/gzip/gzip-1.4.tar.gz
MD5 sum: e381b8506210c794278f5527cba0e765

• Iana-Etc (2.30) - 201 KB:
ホームページ: http://sethwklein.net/iana-etc
ダウンロード: http://sethwklein.net/iana-etc-2.30.tar.bz2
MD5 sum: 3ba3afb1d1b261383d247f46cb135ee8

• Inetutils (1.8) - 1,810 KB:
ホームページ: http://www.gnu.org/software/inetutils/
ダウンロード: http://ftp.gnu.org/gnu/inetutils/inetutils-1.8.tar.gz
MD5 sum: ad8fdcdf1797b9ca258264a6b04e48fd

• IPRoute2 (2.6.35) - 378 KB:
ホームページ: http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2
ダウンロード: http://devresources.linuxfoundation.org/dev/iproute2/download/iproute2-2.6.35.tar.bz2
MD5 sum: b0f281b3124bf04669e18f5fe16d4934

• Kbd (1.15.2) - 1,520 KB:
ダウンロード: http://ftp.altlinux.com/pub/people/legion/kbd/kbd-1.15.2.tar.gz
MD5 sum: 77d0b51454522bc6c170bbdc6e31202a

• Less (436) - 297 KB:
ホームページ: http://www.greenwoodsoftware.com/less/
ダウンロード: http://www.greenwoodsoftware.com/less/less-436.tar.gz
MD5 sum: 817bf051953ad2dea825a1cdf460caa4

• LFS-Bootscripts (20100627) - 43 KB:
ダウンロード: http://www.linuxfromscratch.org/lfs/downloads/6.7/lfs-bootscripts-20100627.tar.bz2
MD5 sum: 8d9bdd8176ccf4c26a86f76e97c1e9ca

• Libtool (2.2.10) - 2,383 KB:
ホームページ: http://www.gnu.org/software/libtool/
ダウンロード: http://ftp.gnu.org/gnu/libtool/libtool-2.2.10.tar.gz
MD5 sum: b745d220e88163fcd9eea0a90ccf21b0

• Linux (2.6.35.4) - 67,636 KB:
ホームページ: http://www.kernel.org/
ダウンロード: http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.35.4.tar.bz2
MD5 sum: 0bb2cd59c13d7412f813c8fbc0769eec

注記

Linux カーネルはわりと頻繁に更新されます。 多くの場合はセキュリティ脆弱性の発見によるものです。 特に正誤情報 (errata) のページにて説明がない限りは、入手可能な最新の 2.6.35.x カーネルを用いてください。
低速度のネットワークや高負荷の帯域幅を利用するユーザーが Linux カーネルをアップデートしようとする場合は、同一バージョンのカーネルパッケージとそのパッチを個別にダウンロードする方法もあります。 その場合、時間の節約を図ることができ、あるいはマイナーバージョンが同一であれば複数パッチを当ててアップグレードする作業時間の短縮が図れます。

• M4 (1.4.14) - 1,099 KB:
ホームページ: http://www.gnu.org/software/m4/
ダウンロード: http://ftp.gnu.org/gnu/m4/m4-1.4.14.tar.bz2
MD5 sum: e6fb7d08d50d87e796069cff12a52a93

• Make (3.82) - 1,213 KB:
ホームページ: http://www.gnu.org/software/make/
ダウンロード: http://ftp.gnu.org/gnu/make/make-3.82.tar.bz2
MD5 sum: 1a11100f3c63fcf5753818e59d63088f

• Man-DB (2.5.7) - 2,123 KB:
ホームページ: http://www.nongnu.org/man-db/
ダウンロード: http://download.savannah.gnu.org/releases/man-db/man-db-2.5.7.tar.gz
MD5 sum: eef0d8c8e54894e4e050e2176bb1d88d

• Man-pages (3.25) - 1,083 KB:
ホームページ: http://www.kernel.org/doc/man-pages/
ダウンロード: http://www.kernel.org/pub/linux/docs/manpages/Archive/man-pages-3.25.tar.bz2
MD5 sum: 3c1fbd5b8905e471827daa0ad937f6b1

• Module-Init-Tools (3.12) - 917 KB:
ホームページ: https://modules.wiki.kernel.org/index.php/Module_init_tools_3_12
ダウンロード: http://www.kernel.org/pub/linux/utils/kernel/module-init-tools/module-init-tools-3.12.tar.bz2
MD5 sum: 8b2257ce9abef74c4a44d825d23140f3

• MPC (0.8.2) - 536 KB:
ホームページ: http://www.multiprecision.org/
ダウンロード: http://www.multiprecision.org/mpc/download/mpc-0.8.2.tar.gz
MD5 sum: e98267ebd5648a39f881d66797122fb6

• MPFR (3.0.0) - 1,112 KB:
ホームページ: http://www.mpfr.org/
ダウンロード: http://www.mpfr.org/mpfr-3.0.0/mpfr-3.0.0.tar.bz2
MD5 sum: f45bac3584922c8004a10060ab1a8f9f

• Ncurses (5.7) - 2,388 KB:
ホームページ: http://www.gnu.org/software/ncurses/
ダウンロード: ftp://ftp.gnu.org/gnu/ncurses/ncurses-5.7.tar.gz
MD5 sum: cce05daf61a64501ef6cd8da1f727ec6

• Patch (2.6.1) - 248 KB:
ホームページ: http://directory.fsf.org/project/patch/
ダウンロード: http://ftp.gnu.org/gnu/patch/patch-2.6.1.tar.bz2
MD5 sum: 0818d1763ae0c4281bcdc63cdac0b2c0

• Perl (5.12.1) - 12,008 KB:
ホームページ: http://cpan.org/
ダウンロード: http://cpan.org/src/5.0/perl-5.12.1.tar.bz2
MD5 sum: f7f2d7f5aaac15a75028381b159a560f

• Pkg-config (0.25) - 966 KB:
ホームページ: http://pkg-config.freedesktop.org/
ダウンロード: http://pkgconfig.freedesktop.org/releases/pkg-config-0.25.tar.gz
MD5 sum: a3270bab3f4b69b7dc6dbdacbcae9745

• Procps (3.2.8) - 279 KB:
ホームページ: http://procps.sourceforge.net/
ダウンロード: http://procps.sourceforge.net/procps-3.2.8.tar.gz
MD5 sum: 9532714b6846013ca9898984ba4cd7e0

• Psmisc (22.12) - 366 KB:
ホームページ: http://psmisc.sourceforge.net/
ダウンロード: http://prdownloads.sourceforge.net/psmisc/psmisc-22.12.tar.gz
MD5 sum: 16c83a351c292cfc845b27d6395e05fb

• Readline (6.1) - 2,209 KB:
ホームページ: http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html
ダウンロード: http://ftp.gnu.org/gnu/readline/readline-6.1.tar.gz
MD5 sum: fc2f7e714fe792db1ce6ddc4c9fb4ef3

• Sed (4.2.1) - 878 KB:
ホームページ: http://www.gnu.org/software/sed/
ダウンロード: http://ftp.gnu.org/gnu/sed/sed-4.2.1.tar.bz2
MD5 sum: 7d310fbd76e01a01115075c1fd3f455a

• Shadow (4.1.4.2) - 1,748 KB:
ホームページ: http://pkg-shadow.alioth.debian.org/
ダウンロード: ftp://pkg-shadow.alioth.debian.org/pub/pkg-shadow/shadow-4.1.4.2.tar.bz2
MD5 sum: d593a9cab93c48ee0a6ba056db8c1997

• Sysklogd (1.5) - 85 KB:
ホームページ: http://www.infodrom.org/projects/sysklogd/
ダウンロード: http://www.infodrom.org/projects/sysklogd/download/sysklogd-1.5.tar.gz
MD5 sum: e053094e8103165f98ddafe828f6ae4b

• Sysvinit (2.88dsf) - 108 KB:
ホームページ: http://savannah.nongnu.org/projects/sysvinit
ダウンロード: http://ftp.twaren.net/Unix/NonGNU/sysvinit/sysvinit-2.88dsf.tar.bz2
MD5 sum: 6eda8a97b86e0a6f59dabbf25202aa6f

• Tar (1.23) - 2,138 KB:
ホームページ: http://www.gnu.org/software/tar/
ダウンロード: http://ftp.gnu.org/gnu/tar/tar-1.23.tar.bz2
MD5 sum: 41e2ca4b924ec7860e51b43ad06cdb7e

• Tcl (8.5.8) - 4,348 KB:
ホームページ: http://tcl.sourceforge.net/
ダウンロード: http://prdownloads.sourceforge.net/tcl/tcl8.5.8-src.tar.gz
MD5 sum: 7f123e53b3daaaba2478d3af5a0752e3

• Texinfo (4.13a) - 2,687 KB:
ホームページ: http://www.gnu.org/software/texinfo/
ダウンロード: http://ftp.gnu.org/gnu/texinfo/texinfo-4.13a.tar.gz
MD5 sum: 71ba711519209b5fb583fed2b3d86fcb

• Udev (161) - 532 KB:
ホームページ: http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html
ダウンロード: http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-161.tar.bz2
MD5 sum: 95b6a0ffc9913f3e8021c65f51eb4f88

• Udev Test Tarball (161) - 147 KB:
ダウンロード: http://anduin.linuxfromscratch.org/sources/other/udev-161-testfiles.tar.bz2
MD5 sum: 8dc84014d4425784313fbbe3d0930e68

• Udev Configuration Tarball - 7 KB:
ダウンロード: http://www.linuxfromscratch.org/lfs/downloads/6.7/udev-config-20100128.tar.bz2
MD5 sum: f417a8c9f43e50fbd43797c0f0a1a7da

• Util-linux-ng (2.18) - 7,490 KB:
ホームページ: http://userweb.kernel.org/~kzak/util-linux-ng/
ダウンロード: http://www.kernel.org/pub/linux/utils/util-linux-ng/v2.18/util-linux-ng-2.18.tar.bz2
MD5 sum: 2f5f71e6af969d041d73ab778c141a77

• Vim (7.3) - 8,675 KB:
ホームページ: http://www.vim.org
ダウンロード: ftp://ftp.vim.org/pub/vim/unix/vim-7.3.tar.bz2
MD5 sum: 5b9510a17074e2b37d8bb38ae09edbf2

• Zlib (1.2.5) - 532 KB:
ホームページ: http://www.zlib.net/
ダウンロード: http://www.zlib.net/zlib-1.2.5.tar.bz2
MD5 sum: be1e89810e66150f5b0327984d8625a0

全パッケージのサイズ合計: 約 281 MB

3.3. 必要なパッチ

パッケージに加えて、いくつかのパッチも必要となります。 それらのパッチはパッケージの不備をただすもので、本来なら開発者が修正すべきものです。 パッチは不備修正だけでなく、ちょっとした修正を施して扱いやすいものにする目的のものもあります。 以下に示すものが LFS システム構築に必要となるパッチすべてです。

日本語訳情報

各パッチには簡略な名称がつけられていますが、 これを日本語に訳してしまうと、パッチの特定ができなくなることが考えられるため、 訳出せずそのまま表記することにします。

• Bash Upstream Fixes Patch - 5.1 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/bash-4.1-fixes-2.patch
MD5 sum: 7813f0e42d41dc4443dc3d161ad24987

• Bzip2 Documentation Patch - 1.6 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/bzip2-1.0.5-install_docs-1.patch
MD5 sum: 6a5ac7e89b791aae556de0f745916f7f

• Bzip2 Version Fixes Patch - 5.3 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/bzip2-1.0.5-version_fixes-1.patch
MD5 sum: 5ef00b9da463b399f3d67ecfa276e7ea

• Coreutils Internationalization Fixes Patch - 121 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/coreutils-8.5-i18n-1.patch
MD5 sum: e806ba5734411d1384f1e56169f31b22

• Coreutils Uname Patch - 1.6 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/coreutils-8.5-uname-2.patch
MD5 sum: 500481b75892e5c07e19e9953a690e54

• Dejagnu Consolidated Patch - 6 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/dejagnu-1.4.4-consolidated-1.patch
MD5 sum: b9949a8abcc210d1dc9cdda06821c199

- Expect Tk Configure Patch - 4.4 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/expect-5.44.1.15-no_tk-1.patch
MD5 sum: ba1b2c5841eea6c62b7522dfde412e65
- Flex GCC-4.4.x Patch - 1 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/flex-2.5.35-gcc44-1.patch
MD5 sum: ad9109820534278c6dd0898178c0788f
- GCC Startfiles Fix Patch - 1.5 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/gcc-4.5.1-startfiles_fix-1.patch
MD5 sum: 799ef1971350d2e3c794f2123f247cc6
- Glibc GCC Build Fix Patch - 2.5 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/glibc-2.12.1-gcc_fix-1.patch
MD5 sum: d1f28cb98acb9417fe52596908bbb9fd
- Glibc Makefile Fix Patch - 1 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/glibc-2.12.1-makefile_fix-1.patch
MD5 sum: 0ef634ac78e582f45d0e7643bfda7505
- Kbd Backspace/Delete Fix Patch - 12 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/kbd-1.15.2-backspace-1.patch
MD5 sum: f75cca16a38da6caa7d52151f7136895
- Man-DB Upstream Fix Assertion Patch - 3.9 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/man-db-2.5.7-fix_man_assertion-1.patch
MD5 sum: a2d7e211160564c13296476cb5f05574
- Patch Testsuite Fix Patch - 1 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/patch-2.6.1-test_fix-1.patch
MD5 sum: c51e1a95bfc5310635d05081472c3534
- Perl Libc Patch - 1 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/perl-5.12.1-libc-1.patch
MD5 sum: 800dfd3c9618731ee5cf57f77a7942b4
- Procps Watch Patch - 3.5 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/procps-3.2.8-watch_unicode-1.patch
MD5 sum: cd1a757e532d93662a7ed71da80e6b58
- Tar Overflow Patch - 1.7 KB:
ダウンロード: http://www.linuxfromscratch.org/patches/lfs/6.7/tar-1.23-overflow_fix-1.patch
MD5 sum: 1912ec36d2a0e2b96678651cf583ce6f

全パッチの合計サイズ: 約 174.1 KB

上に挙げた必須のパッチに加えて LFS コミュニティが提供する任意のパッチが数多くあります。 それらは微小な不備改修や、デフォルトでは利用できない機能を有効にするなどを行います。 http://www.linuxfromscratch.org/patches/downloads/ にて提供しているパッチ類を確認してください。 そして自分のシステムにとって必要なものは自由に適用してください。

第4章 準備作業の仕上げ

4.1. $LFSについて

本書の中では環境変数 LFS を利用していきます。 この変数は常に定義しておくことが必要です。 これはLFSパーティションとして選んだマウントポイントを定義します。 変数 LFS が適切に定義できているかどうかは、以下を実行すれば確認できます。

```
echo $LFS
```

上の出力結果が、LFSパーティションのマウントポイントであることを確認してください。 本書に示す例に従っている場合は /mnt/lfs が表示されるはずです。出力が正しくない場合は、以下のようにして変数をセットします。

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

上のように変数を定義しておくと、 例えば mkdir $LFS/tools といったコマンドを、この通りに入力することで実行できるので便利です。 これが実行されると、シェルが 「$LFS」 を 「/mnt/lfs」 に（あるいは変数にセットされている別のディレクトリに）置換して処理してくれます。

$LFS が常にセットされていることを忘れずに確認してください。 特に、別ユーザーでログインし直した場合（su コマンドによって root ユーザーや別のユーザーでログインした場合）には、忘れずに確認してください。

4.2. $LFS/tools ディレクトリの生成

第5章 にてビルドしていくプログラムは、すべて $LFS/tools ディレクトリ配下にインストールされます。 これらは第6章 にてコンパイル生成されるプログラムとは区別されます。 ここでコンパイルするプログラムは一時的なものであり、最終的な LFS システムを構成するものではありません。 これらのプログラムを分離したディレクトリに置いておけば、後に必要がなくなった時には簡単に削除できます。 またホストシステムの実行環境に入り混じってしまうことを避ける意味もあります。（第5章 の作業でついうっかり、といった失敗がなくなります。）

$LFS/tools ディレクトリは root ユーザーになって以下のコマンドを実行して生成します。

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

次にホストシステム上に /tools のシンボリックリンクを作成します。 これは LFS パーティションに生成されたディレクトリを指し示すものです。 root ユーザーのままで以下を実行します。

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

注記

上のコマンドに間違いはありません。 ln コマンドにはいくつか文法の異なるバージョンがあります。 間違いがあると思った場合には info coreutils ln や ln(1) をよく確認してください。

シンボリックリンクを作成することで、ツールチェーンをコンパイルする準備が整いました。 これにより常に /tools ディレクトリを参照したツールチェーンが生成できます。 コンパイラ、アセンブラ、リンカは本章において動作し（いくつかのツール類は依然ホストシステムのものを利用しますが）、次章においても同様に動作します。（次章では「chroot」 によって LFS パーティションに移動して利用します。）

4.3. LFS ユーザーの追加

root ユーザーでログインしていると、ちょっとした誤操作がもとで、システムを破壊する重大な事態につながることがあります。 そこでパッケージのビルドにあたっては通常のユーザー権限にて作業することにします。 あなた自身のユーザーを利用するのでも構いませんが、全く新しいユーザー環境として lfs というユーザーを作成するのが分かりやすいでしょう。 所属するグループも lfs という名で作成します。 ビルド作業においてはこのユーザーを利用していきます。 そこで root ユーザーになって、新たなユーザーを追加する以下のコマンドを実行します。

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

コマンドラインオプションの意味:

-s /bin/bash
: lfs ユーザーが利用するデフォルトのシェルを bash にします。

-g lfs
lfs ユーザーのグループを lfs とします。

-m
lfs ユーザーのホームディレクトリを生成します。

-k /dev/null
このパラメータは、ディレクトリ名をヌルデバイス（null device）に指定しています。 こうすることでスケルトンディレクトリ（デフォルトは /etc/skel）からのファイル群のコピーを無効とします。

lfs
生成するグループおよびユーザーの名称を与えます。

lfs ユーザーとしてログインするために lfs に対するパスワードを設定します。（root ユーザーでログインしている時に lfs へのユーザー切り替えを行なう場合には lfs ユーザーのパスワードは設定しておく必要はありません。）

```
passwd lfs
```

$LFS/tools ディレクトリの所有者を lfs ユーザーとすることで、このディレクトリへのフルアクセス権を設定します。

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

前述したような作業ディレクトリを作成している場合は、そのディレクトリに対しても lfs ユーザーを所有者とします。

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

lfs でログインします。 これはディスプレイマネージャを通じて仮想端末を用いることができます。 また以下のコマンドを実行するのでも構いません。

```
su - lfs
```

パラメータ 「-」 は su コマンドの実行において、非ログイン（non-login）シェルではなく、ログインシェルを起動することを指示します。 ログインシェルとそうでないシェルの違いについては bash(1) や info bash を参照してください。

4.4. 環境設定

作業しやすい動作環境とするために bash シェルに対するスタートアップファイルを二つ作成します。 lfs ユーザーでログインして、以下のコマンドによって .bash_profile ファイルを生成します。

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

lfs ユーザーとしてログインした時、起動されるシェルは普通は ログイン シェルとなります。 この時、ホストシステムの /etc/profile ファイル（おそらく環境変数がいくつか定義されている）と .bash_profile が読み込まれます。 .bash_profile ファイル内の exec env -i.../bin/bash というコマンドが、起動しているシェルを全くの空の環境として起動し直し HOME、 TERM、 PS1 という環境変数だけを設定します。 これはホストシステム内の不要な設定や危険をはらんだ設定を、ビルド環境に持ち込まないようにするためです。 このようにすることできれいな環境作りを実現できます。

新しく起動するシェルはログインシェルではなくなります。 したがってこのシェルは /etc/profile ファイルや .bash_profile ファイルは読み込まず、代わりに .bashrc ファイルを読み込みます。 そこで以下のようにして .bashrc ファイルを生成します。

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL LFS_TGT PATH
EOF
```

set +h コマンドは bash のハッシュ機能を無効にします。 通常このハッシュ機能は有用なものです。 実行ファイルのフルパスをハッシュテーブルに記憶しておき、再度そのパスを探し出す際に PATH 変数の探査を省略します。 しかしこれより作り出すツール類はインストール直後にすぐ利用していきます。 ハッシュ機能を無効にすることで、プログラム実行が行われる際に、シェルは必ず PATH を探しにいきます。 つまり $LFS/tools ディレクトリ以下に新たに構築したツール類は必ず実行されるようになるわけです。 そのツールの古いバージョンがどこか別のディレクトリにあったとしても、その場所を覚えていて実行されるということがなくなります。

ユーザーのファイル生成マスク（file-creation mask; umask）を 022 にセットするのは、新たなファイルやディレクトリの生成はその所有者にのみ許可し、他者は読み取りと実行を可能とするためです。（システムコール open(2) にてデフォルトモードが適用される場合、新規生成ファイルのパーミッションモードは 644、同じくディレクトリは 755 となります。）

環境変数 LFS は常に指定したマウントポイントを指し示すように設定します。

LC_ALL 変数は特定のプログラムが扱う国情報を制御します。 そのプログラムが出力するメッセージを、指定された国情報に基づいて構成します。 ホストシステムの Glibc が 2.2.4 よりも古いものであって、この LC_ALL を（本章の作業中に）「POSIX」 でもなく 「C」 でもない値にセットしていた場合、chroot 環境からの exit と再度の環境移行を行う際に問題が発生します。 LC_ALL 変数は 「POSIX」 か 「C」 にセットしてください。（両者は同じです。）そのようにセットしておけば、chroot 環境下での作業が問題なく進められます。

LFS_TGT 変数は標準にないマシン名称を設定します。 しかしこれはこの先、クロスコンパイラやクロスリンカの構築、これを用いたツールチェーンの構築の際に、うまく動作させるための設定です。 詳しくは 5.2. 「ツールチェーンの技術的情報」 にて説明しているので参照してください。

/tools/bin ディレクトリを PATH 変数の先頭に設定します。 第5章 にてインストールするプログラムは、インストールした直後からシェルによって実行指示が下されます。 この設定は、ハッシュ機能をオフとしたことと連携して、古いプログラムが実行されないようにします。 たとえホストシステムとの間で同一の実行プログラムがあったとしても、第5章の作業環境下では適切なプログラム実行が実現されます。

一時的なツールを構築する準備の最後として、今作り出したユーザープロファイルを source によって取り込みます。

```
source ~/.bash_profile
```

4.5. SBU 値について

各パッケージをコンパイルしインストールするのにどれほどの時間を要するか、誰しも知りたくなるところです。 しかし Linux From Scratch は数多くのシステム上にて構築可能であるため、正確な処理時間を見積ることは困難です。 最も大きなパッケージ（Glibc）の場合、処理性能の高いシステムでも 20 分はかかります。 それが性能の低いシステムとなると 3日はかかるかもしれません! 本書では処理時間を正確に示すのでなく、 標準ビルド単位（Standard Build Unit; SBU）を用いることにします。

SBU の測定は以下のようにします。 本書で最初にコンパイルするのは 第5章 における Binutils です。 このパッケージのコンパイルに要する時間を標準ビルド時間とし、他のコンパイル時間はその時間からの相対時間として表現します。

例えばあるパッケージのコンパイル時間が 4.5 SBU であったとします。 そして Binutils の1回目のコンパイルが 10 分であったとすると、そのパッケージは およそ 45分かかることを意味しています。 幸いにも、たいていのパッケージは Binutils よりもコンパイル時間は短いものです。

一般にコンパイル時間は、例えばホストシステムの GCC のバージョンの違いなど、多くの要因に左右されるため SBU 値は正確なものになりません。 SBU 値は、インストールに要する時間の目安を示すものに過ぎず、場合によっては十数分の誤差が出ることもあります。

特定マシンにおける実際の処理時間については、以下の LinuxFromScratch SBU ホームページに示していますので参照してください。 http://www.linuxfromscratch.org/~sbu/

注記

最新のシステムは複数プロセッサ（デュアルコアとも言います）であることが多く、パッケージのビルドにあたっては「同時並行のビルド」によりビルド時間を削減できます。 その場合プロセッサ数がいくつなのかを環境変数に指定するか、あるいは make プログラムの実行時に指定する方法があります。 例えばコア2デュオであれば、以下のようにして同時並行の二つのプロセスを実行することができます。

```
export MAKEFLAGS='-j 2'
```

あるいはビルド時の指定として以下のようにすることもできます。

```
make -j2
```

上のようにして複数プロセッサが利用されると、 本書に示している SBU 単位は、通常の場合に比べて大きく変化します。 したがってビルド結果を検証するにしても話が複雑になります。 複数のプロセスラインがインターリーブにより多重化されるためです。 ビルド時に何らかの問題が発生したら、 単一プロセッサ処理を行ってエラーメッセージを分析してください。

4.6. テストスイートについて

各パッケージにはたいていテストスイートがあります。 新たに構築したパッケージに対しては、テストスイートを実行しておくのがよいでしょう。 テストスイートは 「健全性検査（sanity check)」 を行い、パッケージのコンパイルが正しく行われたことを確認します。 テストスイートの実行によりいくつかのチェックが行われ、開発者の意図したとおりにパッケージが正しく動作することを確認していきます。 ただこれは、パッケージにバグがないことを保証するものではありません。

テストスイートの中には他のものにも増して重要なものがあります。 例えば、ツールチェーンの要である GCC、Binutils、Glibc に対してのテストスイートです。 これらのパッケージはシステム機能を確実なものとする重要な役割を担うものであるためです。 GCC と Glibc におけるテストスイートはかなりの時間を要します。 それが低い性能のマシンであればなおさらです。 でもそれらを実行しておくことを強く推奨します。

注記

作業を進めてみれば分かることですが 第5章 の作業においてテストスイートを実行することはあまり意味がありません。 というのも、この章において実施するテストに対しては、ホストシステムによるある程度の影響があるためです。 時には不可解なエラーが発生することもあります。 第5章 にて生成するツール類は一時的なものであり、その後には利用しなくなります。 したがって普通のユーザーであれば 第5章 においてはテストスイートを実行しないことをお勧めします。 テストスイートを実行する手順を説明してはいますが、それはテスターの方、開発者の方のために説明しているものであって、それらは全くのオプションです。

Binutils と GCC におけるテストスイートの実行では、擬似端末（pseudo terminals; PTY）を使い尽くす問題が発生します。 これにより相当数のテストが失敗します。 これが発生する理由はいくつかありますが、もっともありがちな理由としてはホストシステムの **devpts** ファイルシステムが正しく構成されていないことがあげられます。 この点については http://www.linuxfromscratch.org//lfs/faq.html#no-ptys においてかなり詳しく説明しています。

パッケージの中にはテストスイートに失敗するものがあります。 しかしこれらは開発元が認識しているもので致命的なものではありません。 以下の http://www.linuxfromscratch.org/lfs/build-logs/6.7/ に示すログを参照して、失敗したテストが実は予期されているものであるかどうかを確認してください。 このサイトは、本書におけるすべてのテストスイートの正常な処理結果を示すものです。

第5章 一時的環境の構築

5.1. はじめに

この章では最小限の Linux システムを構築していく方法を示します。 このシステムは、最終的に 第6章 にて LFS システムを構築するためのもので、そのために必要なツール類をすべて含んでいます。 最小限とは言いつつも、取り扱いやすい実行環境を提供します。

最小限のシステムを構築するために、以下の二段階の手順を踏みます。 初めにホストシステムに依存しない新しいツールチェーン（コンパイラ、アセンブラ、リンカ、ライブラリ、その他の有用なユーティリティ）を構築します。 次にこのツールチェーンを使って、他の重要なツール類を構築していきます。

この章にて生成されるファイル群は $LFS/tools ディレクトリ配下にインストールされます。 これらのファイルは、次章にてインストールされるファイル群や、ホスト環境にあるファイル群とは区分けされます。 ここで構築されるパッケージ類は、あくまで一時的なものであるため、この後に構築する LFS システムを汚したくないためにこのようにします。

5.2. ツールチェーンの技術的情報

本節ではシステムをビルドする原理や技術的な詳細について説明します。 この節のすべてをすぐに理解する必要はありません。 この先、実際の作業を行っていけば、いろいろな情報が明らかになってくるはずです。 各作業を進めながら、いつでもこの節に戻って読み直してみてください。

第5章 の最終目標は一時的なシステム環境を構築することです。 この一時的なシステムには、システム構築のための十分なツール類を有し、ホストシステムとは切り離されたものです。 この環境へは chroot によって移行します。この環境は 第6章 において、クリーンでトラブルのない LFS システムの構築を行う土台となるものです。 構築手順の説明においては、初心者の方であっても失敗を最小限にとどめ、同時に最大限の学習材料となるように心がけています。

重要項目

これより先に進む前に、作業するプラットフォームの 「三つの組（target triplet)」 で表される名称を確認してください。「三つの組」 は config.guess スクリプトを実行することで簡単に確認できます。 そのスクリプトは多くのパッケージのソースに含まれています。 Binutils パッケージのソースを伸張（解凍）し ./config.guess スクリプトを実行してその出力を確認してみてください。 例えば最近の 32 ビット Intel プロセッサでは i686-pc-linux-gnu のような出力が得られます。

利用しているプラットフォームに応じたダイナミックリンカ（dynamic linker）の名前についても確認してください。 ダイナミックローダ（dynamic loader）とも表現されるものです。（Binutils が提供する標準的なリンカ ld とは異なりますので注意してください。）Glibc が提供するこのダイナミックリンカは、プログラムが必要としている共有ライブラリを見つけ出してロードし、実行のための準備を行った上で実際に実行します。 32 ビットマシンのダイナミックリンカの名前は ld-linux.so.2 といったものになります。 確実にその名前を調べるなら、ホストシステム内のどれでも良いので実行モジュールを選んで readelf -l <実行モジュール名> | grep interpreter と入力します。出力される結果を確認してください。 あらゆるプラットフォームの情報を知りたいなら Glibc のソースディレクトリのルートにある shlib-versions ファイルに記されています。

第5章 におけるビルド手順がどのように機能するのか、その技術的な情報を以下に示します。

- 動作させているプラットフォームの名前を微妙に変えます。 三つの組の ”ベンダー ” フィールドを変更するもので、 LFS_TGT 変数に定め利用します。 こうしておいて Binutils と GCC の初回の構築を行なえば、互換性のあるクロスコンパイラ、クロスリンカを確実に構築できるようになります。 もう一つ別のアーキテクチャに対する実行モジュールを作らなくても、そのクロスコンパイラとクロスリンカを使えば、生成される実行モジュールは現在のハードウェアに適合したものとなります。
- 一時的に構築するライブラリはクロスコンパイルにより生成します。 クロスコンパイラというものは元来、ホストシステムへ依存するものではないためです。 こうすることで、ホストシステムのヘッダやライブラリが、一時的なツール類を壊してしまうような危険を減らすことができ、同時に 64 ビットマシンにて 32 ビットあるいは 64 ビットの双方のライブラリを構築することができるようになります。
- gcc のスペック（specs）ファイルを適切に調整することで、どのダイナミックリンカを用いるのかをコンパイラに指示します。

Binutils をまず初めにインストールします。 この後の GCC や Glibc の configure スクリプトの実行ではアセンブラやリンカに対する様々な機能テストが行われるためで、そこではどの機能が利用可能または利用不能であるかが確認されます。 ただ重要なのは Binutils を一番初めにビルドするという点だけではありません。 Gcc や Glibc の

configure が正しく処理されなかったとすると、ツールチェーンがわずかながらも不完全な状態で生成されてしまいます。 この状態は、すべてのビルド作業を終えた最後になって、大きな不具合となって現れてくることになります。 テストスイートを実行することが欠かせません。 これを実行しておけば、この先に行う多くの作業に入る前に不備があることが分かるからです。

Binutils はアセンブラとリンカを二箇所にインストールします。 /tools/bin と /tools/$LFS_TGT/bin です。 これらは一方が他方のハードリンクとなっています。 リンカの重要なところはライブラリを検索する順番です。 ld コマンドに --verbose オプションをつけて実行すれば詳しい情報が得られます。 例えば ld --verbose | grep SEARCH を実行すると、検索するライブラリのパスとその検索順を示してくれます。 ダミープログラムをコンパイルして ld に --verbose オプションをつけてリンクを行うと、どのファイルがリンクされたが分かります。 例えば gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded と実行すれば、リンカの処理中にオープンに成功したファイルがすべて表示されます。

次にインストールするのは GCC です。 configure の実行時には以下のような出力が行われます。

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

これを示すのには重要な意味があります。 GCC の configure スクリプトは、利用するツール類を探し出す際に PATH ディレクトリを参照していないということです。 しかし gcc の実際の処理にあたっては、その検索パスが必ず使われるわけでもありません。 gcc が利用する標準的なリンカを確認するには gcc -print-prog-name=ld を実行します。

さらに詳細な情報を知りたいときは、ダミープログラムをコンパイルする際に -v オプションをつけて実行します。 例えば gcc -v dummy.c と入力すると、プリプロセッサ、コンパイル、アセンブルの各処理工程が示されますが、さらに gcc がインクルードした検索パスとその読み込み順も示されます。

次のパッケージは Glibc です。 Glibc 構築の際に気にかけるべき重要なものは、コンパイラ、バイナリツール、カーネルヘッダです。 コンパイラについては、一般にはあまり問題にはなりません。 Glibc は常に configure スクリプトにて指定される --host パラメータに関連づけしたコンパイラを用いるからです。 我々の作業では i686-lfs-linux-gnu-gcc になります。 バイナリツールとカーネルヘッダは多少複雑です。 従って無理なことはせずに有効な configure オプションを選択することが必要です。 configure 実行の後は glibc-build ディレクトリにある config.make ファイルに重要な情報が示されているので確認してみてください。 なお CC="i686-lfs-gnu-gcc" とすれば、どこにある実行モジュールを利用するかを制御でき -nostdinc と -isystem を指定すれば、コンパイラに対してインクルードファイルの検索パスを制御できます。 これらの指定は Glibc パッケージの重要な面を示しています。 Glibc がビルドされるメカニズムは自己完結したビルドが行われるものであり、ツールチェーンのデフォルト設定には基本的に依存しないことを示しています。

Glibc をインストールした後は、 gcc のスペックファイルにて /tools/lib ディレクトリにある新しいダイナミックリンカを用いるような修正を行います。 この修正により /tools 内での検索とリンクが行われるようにします。 ダイナミックリンカに対する固定的な検索パスの設定は、ここから生成されるすべての ELF (Executable and Link Format) 形式の実行モジュールにも埋め込まれていきます。 その結果は readelf -l <実行モジュール名> | grep interpreter を実行すれば確認できます。 gcc のスペック・ファイルを修正するのは、これ以降、本章の最後に至るまで、すべてのプログラムのコンパイル時に /tools/lib にあるダイナミックリンカが利用されるよう仕向けるものです。

GCC の第2回目のビルドにおいても、スペックファイルを修正して新しいダイナミックリンカが用いられるようにします。 これをもし誤ってしまうと、ホストシステムの /lib ディレクトリが埋め込まれたダイナミックリンカを用いるものとして GCC が生成されてしまいます。 こうしてしまうと、ホストシステムに依存しない形を目指すという目的が達成できません。

Binutils の2回めのビルドにおいては ld コマンドのライブラリ検索パスを設定するために configure の --with-lib-path オプションを指定します。 それ以降ツールチェーンの核となるツール類は、自分自身から作り出された (self-contained) 自分だけで処理できる (self-hosted) 形となります。 第5章 において構築する残りのパッケージは /tools ディレクトリの新しい Glibc を用いてビルドされます。

第6章 での chroot による環境下では、実質的なパッケージとして Glibc を初めにビルドします。 これは上に述べているように自己完結した性質を目指すためです。 /usr に Glibc をインストールしたら、ツールチェーンのデフォルトディレクトリの変更を行い LFS システムを構築する残りのパッケージをビルドしていきます。

5.3. 全般的なコンパイル手順

パッケージをビルドしていく際には、以下に示す内容を前提とします:

- パッケージの中には、コンパイルする前にパッチを当てるものがあります。 パッチを当てるのは、そのパッケージが抱える問題を回避するためです。 本章と次章の双方でパッチを当てるものがあり、 あるいは本章と次章のいずれか一方でパッチを当てるものもあります。 したがってパッチをダウンロードする説明が書かれていないなら、 何も気にせず先に進んでください。 パッチを当てた際に offset や fuzz といった警告メッセージが出る場合がありますが、これらは気にしないでください。 このような時でもパッチは問題なく適用されています。

- コンパイルの最中に、警告メッセージが画面上に出力されることがよくあります。 これは問題はないため無視して構いません。 警告メッセージは、メッセージ内に説明されているように、C や C++ の文法が誤りではないものの推奨されていないものであることを示しています。 C 言語の標準はよく変更されますが、パッケージの中には古い基準に従っているものもあります。 問題はないのですが、警告として画面表示されることになるわけです。

重要項目

各パッケージをインストールした後は、特に具体的な指示がない限りは、そのソースディレクトリやビルドディレクトリは削除してください。 ソースディレクトリを削除するのは、後にもう一度そのパッケージをインストールする際に、構築のミスを防ぐためです。

- もう一度、環境変数 LFS が正しく設定されているかを確認します。

```
echo $LFS
```

上の出力結果が LFS パーティションのマウントポイントのディレクトリであることを確認してください。 本書では /mnt/lfs ディレクトリとして説明しています。

- 最後に以下の二つの点にも注意してください。

重要項目

ビルド作業においては bash シェルの利用を想定しています。

重要項目

パッケージのビルド操作を進めるために、まずは lfs ユーザーによってパッケージファイルの伸張（解凍）を行い、cd コマンドによりそのパッケージディレクトリに移動します。

ビルド作業では以下の点が重要です。

1. ソースやパッチファイルを配置するディレクトリは /mnt/lfs/sources/ などのように
   chroot 環境でもアクセスが出来るディレクトリとしてください。
   /mnt/lfs/tools/ ディレクトリにソースを置くことは やめて ください。
2. ソースディレクトリに入ります。
3. 各パッケージにおいては、
   a. tar コマンドを使ってパッケージの tarball を伸張（解凍）します。
   b. パッケージの伸張（解凍）後に生成されたディレクトリに入ります。
   c. 本書の手順に従ってビルド作業を行っていきます。
   d. ソースディレクトリに戻ります。
   e. ビルド作業を通じて生成されたパッケージディレクトリを削除します。さらに
      <package>-build なるディレクトリを生成していた場合は、それも削除します。

5.4. Binutils-2.20.1 - 1回め

Binutils パッケージは、リンカやアセンブラなどのようにオブジェクトファイルを取り扱うツール類を提供します。

概算ビルド時間: 1 SBU
必要ディスク容量: 248 MB

5.4.1. クロスコンパイル版 Binutils のインストール

注記

前の節に戻って再度説明をよく読み、重要事項として説明している内容をよく理解しておいてください。 そうすればこの後の無用なトラブルを減らすことができるはずです。

Binutils は一番最初にビルドするパッケージです。 ここでビルドされるリンカやアセンブラを使って、Glibc や GCC の様々な機能が利用できるかどうかを判別することになります。

Binutils のドキュメントでは Binutils をビルドする際に、ソースディレクトリではなく、ビルド専用のディレクトリを使ってビルドすることを推奨しています。

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

注記

本節以降で SBU値を示していきます。 これを活用していくなら、本パッケージの configure から初めのインストールまでの処理時間を計測しましょう。 具体的には処理コマンドを time で囲んで time { ./configure ... && make && make install; } と入力すれば実現できます。

注記

概算ビルド時間と必要ディスク容量は、この第5章ではテストスイートに関わる時間や容量は含めないことにします。

Binutils をコンパイルするための準備をします。

```
../binutils-2.20.1/configure \
    --target=$LFS_TGT --prefix=/tools \
    --disable-nls --disable-werror
```

configure オプションの意味：

--target=$LFS_TGT
変数 LFS_TGT に設定しているマシン名は config.guess スクリプトが返すものとは微妙に異なります。 そこでこのオプションは、Binutils のビルドにあたってクロスリンカをビルドするように configure スクリプトに指示するものです。

--prefix=/tools
configure スクリプトに対して Binutils プログラムを /tools ディレクトリ以下にインストールすることを指示します。

--disable-nls
一時的なツール構築にあたっては i18n 国際化は行わないことを指示します。

--disable-werror
ホストのコンパイラが警告を発した場合に、ビルドが中断することがないようにします。

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。通常ならここでテストスイートを実行します。 しかしシステム構築初期のこの段階ではテストスイートのフレームワーク（Tcl, Expect, DejaGNU）が準備できていません。 さらにこの時点で生成されるプログラムは、すぐに次の生成作業によって置き換えられますから、 この時点でテストを実行することはあまり意味がありません。

x86_64 にて作業をしている場合は、ツールチェーンの切り分けを適切に行うためにシンボリックリンクを作成します。

```
case $(uname -m) in
  x86_64) mkdir -v /tools/lib && ln -sv lib /tools/lib64 ;;
esac
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.12.2. 「Binutils の構成」 を参照してください。

5.5. GCC-4.5.1 - 1回め

GCC パッケージは C コンパイラや C++ コンパイラなどの GNU コンパイラコレクションを提供します。

概算ビルド時間: 5.0 SBU
必要ディスク容量: 809 MB

5.5.1. クロスコンパイル版 GCC のインストール

最近の GCC は GMP、MPFR、MPC の各パッケージを必要とします。 これらのパッケージはホストシステムに含まれていないかもしれないため、以下を実行してビルドの準備をします。

```
tar -jxf ../mpfr-3.0.0.tar.bz2
mv -v mpfr-3.0.0 mpfr
tar -jxf ../gmp-5.0.1.tar.bz2
mv -v gmp-5.0.1 gmp
tar -zxf ../mpc-0.8.2.tar.gz
mv -v mpc-0.8.2 mpc
```

GCC のドキュメントでは、ソースディレクトリ以外の専用のビルドディレクトリを作成することが推奨されています。

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

GCC をコンパイルするための準備を行います。

```
../gcc-4.5.1/configure \
    --target=$LFS_TGT --prefix=/tools \
    --disable-nls --disable-shared --disable-multilib \
    --disable-decimal-float --disable-threads \
    --disable-libmudflap --disable-libssp \
    --disable-libgomp --enable-languages=c \
    --with-gmp-include=$(pwd)/gmp --with-gmp-lib=$(pwd)/gmp/.libs \
    --without-ppl --without-cloog
```

configure オプションの意味：

--disable-shared
このオプションは内部ライブラリをスタティックライブラリとしてリンクすることを指示します。 ホストシステムに関係しそうな問題を回避するためです。

--disable-decimal-float, --disable-threads, --disable-libmudflap, --disable-libssp, --disable-libgomp
これらのオプションは順に、十進浮動小数点制御、スレッド処理、libmudflap、libssp、libgomp のサポートをいずれも無効にすることを指示します。 これらの機能を含めていると、クロスコンパイラをビルドする際にはコンパイルに失敗します。 またクロスコンパイルによって一時的な libc ライブラリを構築する際には不要なものです。

--disable-multilib
x86_64 に対して LFS は まだ multilib のサポートをしていません。 このオプション指定は x86 には無関係です。

--enable-languages=c
このオプションは C コンパイラのみビルドすることを指示します。 この時点で必要なのはこの言語だけだからです。

--with-gmp-include=...
このオプションは GCC に対して GMP のヘッダファイルの場所を指定するものです。

--with-gmp-lib=...
このオプションは GCC に対して GMP のライブラリファイルの場所を指定するものです。

--without-ppl, --without-cloog
このオプションは、PPL および CLooG ライブラリがホストシステムに存在していたとしても、chroot 環境ではそれらを利用することが出来ないため、リンクしないようにします。

GCC をコンパイルします。

```
make
```

コンパイルが終了しました。 この時点でもテストスイートを実行することはできます。 ただ前にも述べているように、テストスイートのフレームワークがまだ準備できていません。 さらにこの時点で生成されるプログラムは、すぐに次の生成作業によって置き換えられますから、 この時点でテストを実行することはあまり意味がありません。

パッケージをインストールします。

```
make install
```

--disable-shared オプションを指定すると libgcc_eh.a を生成せずインストールしません。 Glibc パッケージはこのライブラリに依存しており、ビルドの際に -lgcc_eh を指定することで利用されます。依存している点は libgcc.a へのシンボリックリンクを生成しておけば問題はありません。 libgcc_eh.a に含まれるオブジェクトが、 最終的には libgcc.a の中にも含まれることになるからです。

```
ln -vs libgcc.a `$LFS_TGT-gcc -print-libgcc-file-name | \
    sed 's/libgcc/&_eh/'`
```

本パッケージの詳細は 6.16.2. 「GCC の構成」 を参照してください。

5.6. Linux-2.6.35.4 API ヘッダ

Linux API ヘッダは Glibc が利用するカーネル API を提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 466 MB

5.6.1. Linux API ヘッダのインストール

Linux カーネルはアプリケーションプログラミングインターフェース（Application Programming Interface）を、システムの C ライブラリ（LFS の場合 Glibc）に対して提供する必要があります。 これを行うには Linux カーネルのソースに含まれる、さまざまな C ヘッダファイルを「健全化（sanitizing)」して利用します。

これより前に一度処理を行っていたとしても、不適切なファイルや誤った依存関係を残さないように、以下を処理します。

```
make mrproper
```

そしてユーザーが利用するカーネルヘッダファイルをテストし、ソースから抽出します。 それらはいったん中間的なローカルディレクトリに置かれ、必要な場所にコピーされます。 ターゲットディレクトリに既にあるファイルは削除されてからソースからの抽出処理が行われます。

```
make headers_check
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /tools/include
```

本パッケージの詳細は 6.7.2. 「Linux API ヘッダの構成」 を参照してください。

5.7. Glibc-2.12.1

Glibc パッケージは主要な C ライブラリを提供します。 このライブラリは基本的な処理ルーチンを含むもので、メモリ割り当て、ディレクトリ走査、ファイルのオープン・クローズや入出力、文字列操作、パターンマッチング、算術処理、等々があります。

概算ビルド時間: 6.9 SBU
必要ディスク容量: 371 MB

5.7.1. Glibc のインストール

Glibc が GCC-4.5.1 に対してビルドできなくなるバグを修正します。

```
patch -Np1 -i ../glibc-2.12.1-gcc_fix-1.patch
```

Make の バージョンが 3.81 より最新では Glibc がビルドできないバグを修正します。

```
patch -Np1 -i ../glibc-2.12.1-makefile_fix-1.patch
```

Glibc のドキュメントでは、ソースディレクトリ以外の専用のビルドディレクトリを作成することが推奨されています。

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

Glibc は i386 をサポートしなくなりました。開発者によると x86 マシンに対してはコンパイラフラグとして **-march=i486** を指定する必要があります。 これを実際に指定する方法はいくつかあります。 もっとも試してみれば分かりますが、ビルド時の変数 「CFLAGS」 にて設定しておくのが一番です。 Glibc の内部的なビルドシステムが利用する CFLAGS を上書き設定するのとは別に **configparms** ファイルという特別なファイルを使って CFLAGS に対して追加設定を行うこともできます。 -mtune=native というフラグも必要で -march を設定した際に変更される -mtune の値を適切にリセットします。

```
case 'uname -m' in
  i?86) echo "CFLAGS += -march=i486 -mtune=native" > configparms ;;
esac
```

次に Glibc をコンパイルするための準備をします。

```
../glibc-2.12.1/configure --prefix=/tools \
    --host=$LFS_TGT --build=$(../glibc-2.12.1/scripts/config.guess) \
    --disable-profile --enable-add-ons \
    --enable-kernel=2.6.22.5 --with-headers=/tools/include \
    libc_cv_forced_unwind=yes libc_cv_c_cleanup=yes
```

configure オプションの意味：

--host=$LFS_TGT, --build=$(../glibc-2.12.1/scripts/config.guess)
このようなオプションを組み合わせることで /tools ディレクトリにあるクロスコンパイラ、クロスリンカを使って Glibc がクロスコンパイルされるようになります。

--disable-profile
プロファイル情報を含めずにライブラリをビルドすることを指示します。 一時的なツールにてプロファイル情報が必要な場合は、このオプションを取り除いてください。

--enable-add-ons
スレッド処理のライブラリとして NPTL アドオンを利用することを指示します。

--enable-kernel=2.6.22.5
Linux カーネル 2.6.22.5 以上のサポートを行うよう指示します。 これ以前のカーネルは利用することができません。

--with-headers=/tools/include
これまでに tools ディレクトリにインストールしたヘッダファイルを用いて Glibc をビルドすることを指示します。 こうすればカーネルにどのような機能があるか、どのようにして処理効率化を図れるかなどの情報を Glibc が得られることになります。

libc_cv_forced_unwind=yes
5.4. 「Binutils-2.20.1 - 1回め」 においてインストールしたリンカは、クロスコンパイルにより生成したものです。 これは Glibc をインストールするまでは使えません。 これはつまり force-unwind サポートに対するテスト

は失敗することを意味します。 正しく動作するリンカに依存するためです。 libc_cv_forced_unwind=yes の変数設定は、 configure スクリプトに対して テストを実行しなくても force-unwind サポート機能を利用可能とすることを指示します。

libc_cv_c_cleanup=yes
上と同様に configure スクリプトに対して libc_cv_c_cleanup=yes を指示します。 これによりテストが省略され、C のクリーンアップハンドリング（cleanup handling）のサポートを指定します。

ビルド中には以下のようなメッセージが出力されるかもしれません。

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

msgfmt プログラムがない場合（missing）や互換性がない場合（incompatible）でも特に問題はありません。 msgfmt プログラムは Gettext パッケージが提供するもので、ホストシステムに含まれているかもしれません。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートは存在しますが、ここで実行することはできません。 この時点ではまだ C++ コンパイラを構築していないためです。

注記

テストスイートを正しく実行するためには、さらにロケールデータも必要になります。 ロケールデータは、システム内の各種ユーティリティが、日付、時刻、通貨などの情報を利用したり出力したりするために用いられるものです。 テストスイートの実行は不要と説明していることから、これに従って実行しない場合はロケールデータをここでインストールする必要はありません。 適切なロケールデータは次章にてインストールします。 それでもここでインストールするなら 6.9. 「Glibc-2.12.1」 に示される手順に従ってください。

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.9.4. 「Glibc の構成」 を参照してください。

5.8. ツールチェーンの調整

一時的な C ライブラリをインストールしました。 これ以降の章でコンパイルしていくツール類は、このライブラリをリンクしていきます。 リンクを行うにはクロスコンパイラのスペックファイルを修正して /tools ディレクトリにあるダイナミックリンカを指し示すようにします。

具体的にはコンパイラの 「スペック（specs）」 ファイルをダンプして、これが参照されるディレクトリに置きます。以下では単純な sed コマンドによる置換によって GCC が利用するダイナミックリンカを変更します。 ここで為すことは /lib ディレクトリ内（ホストが 64 ビットなら /lib64 ディレクトリ内）のダイナミックリンカファイルへの参照を探し出し、これを新しい /tools への参照へと調整することです。

作業を正確に行うために、以下のコマンド実行にあたってはコピー・ペーストによりコマンド入力を行うことをお勧めします。 そしてスペックファイルを開いて、ダイナミックリンカの配置場所を示す記述がすべて適切に調整されていることを確認してください。 必要に応じて 5.2. 「ツールチェーンの技術的情報」 を読み直し、ダイナミックリンカのデフォルト名を確認してください。

```
SPECS='dirname $($LFS_TGT-gcc -print-libgcc-file-name)'/specs
$LFS_TGT-gcc -dumpspecs | sed \
  -e 's@/lib\(64\)\?/ld@/tools&@g' \
  -e "/^\*cpp:$/{n;s,$, -isystem /tools/include,}" > $SPECS
echo "New specs file is: $SPECS"
unset SPECS
```

注意

この時点において新しく構築したツールチェーンの基本的な（コンパイルやリンクなどの）機能が正しく動作していることを確認する必要があります。 健全性検査（sanity check）を行うために以下を実行してください。

```
echo 'main(){}' > dummy.c
$LFS_TGT-gcc -B/tools/lib dummy.c
readelf -l a.out | grep ': /tools'
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドから出力される結果は以下のようになるはずです。

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

ダイナミックリンカのディレクトリは /tools/lib に、あるいは 64 ビットマシンであれば /tools/lib64 になります。

コマンドの出力結果が上と異なっていたり、あるいは何も出力されなかった場合は、何かがおかしいことを意味します。 どこに問題があるのか調査・再試行を行って解消してください。 解決せずにこの先に進まないでください。 おかしいとすれば、上で行ったスペックファイルの修正に何か問題があったのかもしれません。もしそうであったなら、スペックファイルの修正を、コマンドのコピー・ペースト作業に十分注意して再度行ってください。

すべてが終了したらテストファイルを削除します。

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

注記

次節にてビルドする Binutils では、ツールチェーンが正しくビルドできているかどうかを改めてチェックします。 もし Binutils のビルドが失敗したなら、それはここまでに行ってきた Binutils、GCC、Glibc のビルドに失敗していることを意味します。

5.9. Binutils-2.20.1 - 2回め

Binutils パッケージは、リンカやアセンブラなどのようにオブジェクトファイルを取り扱うツール類を提供します。

概算ビルド時間: 1.3 SBU
必要ディスク容量: 259 MB

5.9.1. Binutils のインストール

ビルドのためのディレクトリを再び生成します。

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

Binutils をコンパイルするための準備をします。

```
CC="$LFS_TGT-gcc -B/tools/lib/" \
   AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib \
   ../binutils-2.20.1/configure --prefix=/tools \
   --disable-nls --with-lib-path=/tools/lib
```

configure オプションの意味：

CC="$LFS_TGT-gcc -B/tools/lib/" AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib
Binutils をネイティブにビルドすることが目的なので、ホストシステムに存在しているクロスコンパイラや関連ツールは使わず、ビルドしているシステム内のものを用いるように指定します。

--with-lib-path=/tools/lib
configure スクリプトに対して Binutils のコンパイル中でのライブラリパスを指定します。 リンカに対して /tools/lib ディレクトリを指定するものです。 こうすることでリンカがホスト上のライブラリを検索しないようにします。

パッケージをコンパイルします。

```
make
```

パッケージをインストールします。

```
make install
```

次章で行う「再調整」の作業に向けてリンカを準備します。

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
cp -v ld/ld-new /tools/bin
```

make パラメータの意味：

-C ld clean
サブディレクトリ ld にコンパイル生成されたプログラムをすべて削除します。

-C ld LIB_PATH=/usr/lib:/lib
サブディレクトリ ld の中に生成されるべきプログラムを再生成します。 Makefile ファイル内の変数 LIB_PATH をコマンドラインから与えることで、一時的なツール類の設定を上書き指定し、適切なパスを指示します。 この変数の設定はリンカに対するデフォルトの検索パスを指定するものであり、次章に向けた準備となります。

本パッケージの詳細は 6.12.2. 「Binutils の構成」 を参照してください。

5.10. GCC-4.5.1 - 2回め

GCC パッケージは C コンパイラや C++ コンパイラなどの GNU コンパイラコレクションを提供します。

概算ビルド時間: 9.0 SBU
必要ディスク容量: 1003 MB

5.10.1. GCC のインストール

バージョン 4.3 以降の GCC を用いてここでのビルド作業を行うと、出来上がるのは再配置可能なコンパイラ（relocated compiler）であり、--prefix パラメータによって指定されたディレクトリからの起動ファイル（startfiles）の探索が行われないものになります。 しかしここで作り出すのは再配置可能なコンパイラではなく、 /tools ディレクトリにある起動ファイルは /tools ディレクトリ内のライブラリにリンクされたコンパイラを作り出すことが必要であるため、以下のパッチを適用します。 このパッチは、部分的に GCC の古い機能を復活させるものです。

```
patch -Np1 -i ../gcc-4.5.1-startfiles_fix-1.patch
```

通常の利用環境において GCC が提供する fixincludes スクリプトは、根本的に不備のあるヘッダファイルを修正する目的で利用されます。 しかしこの時点で GCC-4.5.1 と Glibc-2.12.1 を既にインストールしており、それぞれのヘッダファイルは修正する必要がないことが分かっています。 つまり fixincludes スクリプトを利用する必要がありません。 もし実行してしまうと、ホストシステムに既に存在していたヘッダファイルが修正され、それが GCC のプライベートなディレクトリへとインストールされることになり、ビルド環境を壊してしまうことになります。 そこで fixincludes スクリプトの実行を無効とするために以下を実行します。

```
cp -v gcc/Makefile.in{,.orig}
sed 's@\./fixinc\.sh@-c true@' gcc/Makefile.in.orig > gcc/Makefile.in
```

x86 マシンにおいてブートストラップビルドを行うと、コンパイラフラグ -fomit-frame-pointer が設定されます。 しかしブートストラップではないビルドの場合はデフォルトではこのフラグが無効化されてしまいます。 ここで実現したいのは、ブートストラップビルドを行った場合とまったく同じコンパイラをビルドすることです。 そこで以下の sed コマンドにより、強制的に上のフラグを利用するようにします。

```
cp -v gcc/Makefile.in{,.tmp}
sed 's/^T_CFLAGS =$/& -fomit-frame-pointer/' gcc/Makefile.in.tmp \
  > gcc/Makefile.in
```

以下のコマンドは GCC が利用するダイナミックリンカの場所を変更して /tools ディレクトリにインストールしたものを用いるようにします。 同時に GCC が探索するインクルードファイルのパスから /usr/include を取り除きます。 インストールの後にスペックファイルを調整する方法もありますが、今ここでこのようにするのは GCC の実際のビルドにおいて新しいダイナミックリンカを用いるようにするためです。 つまりここでのビルドを通じてすべての実行モジュール類を新しい Glibc に対してリンクするものです。 以下のコマンドによりそれを実現します。

```
for file in \
 $(find gcc/config -name linux64.h -o -name linux.h -o -name sysv4.h)
do
  cp -uv $file{,.orig}
  sed -e 's@/lib\(64\)\?\(32\)\?/ld@/tools&@g' \
  -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_INCLUDE_DIR
#define STANDARD_INCLUDE_DIR 0
#define STANDARD_STARTFILE_PREFIX_1 ""
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
  touch $file.orig
done
```

上のコマンドがよく分からない場合は一つ一つ読み下していってください。 まず gcc/config ディレクトリには linux.h、 linux64.h、 sysv4.h といったファイルのいずれかがあるはずです。 それらが存在したら、ファイル名称の末尾に 「.orig」 をつけたファイルとしてコピーします。 そして一つめの sed コマンドでは、そのファイル内にある「/lib/ld」、 「/lib64/ld」、 「/lib32/ld」 という記述部分の頭に 「/tools」 を付与します。 また二つめの sed コマンドによってハードコーディングされている 「/usr」 という部分を書き換えます。 そしてここで加えるべき定義文をファイルの末尾に追加し、検索パスと startfile プリフィックスを変更します。 最後に touch によってコピーしたファイルのタイムスタンプを更新します。 cp -u を用いるのは、誤ってコマンドを二度起動したとしてもオリジナルファイルを壊さないようにするためです。

x86_64 では GCC の multilib スペックを無効化します。 これはホスト上のライブラリにリンクされないようにするためです。

```
case $(uname -m) in
  x86_64)
    for file in $(find gcc/config -name t-linux64) ; do \
      cp -v $file{,.orig}
      sed '/MULTILIB_OSDIRNAMES/d' $file.orig > $file
    done
  ;;
esac
```

GCC を初めてビルドする際には GMP、MPFR、MPC の各パッケージを必要とします。 tarball を解凍して、所定のディレクトリ名に移動させます。

```
tar -jxf ../mpfr-3.0.0.tar.bz2
mv -v mpfr-3.0.0 mpfr
tar -jxf ../gmp-5.0.1.tar.bz2
mv -v gmp-5.0.1 gmp
tar -zxf ../mpc-0.8.2.tar.gz
mv -v mpc-0.8.2 mpc
```

専用のディレクトリを再度生成します。

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

GCC のビルドに入る前に、デフォルトの最適化フラグを上書きするような環境変数の設定がないことを確認してください。

GCC をコンパイルするための準備をします。

```
CC="$LFS_TGT-gcc -B/tools/lib/" \
    AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib \
    ../gcc-4.5.1/configure --prefix=/tools \
    --with-local-prefix=/tools --enable-clocale=gnu \
    --enable-shared --enable-threads=posix \
    --enable-__cxa_atexit --enable-languages=c,c++ \
    --disable-libstdcxx-pch --disable-multilib \
    --disable-bootstrap --disable-libgomp \
    --with-gmp-include=$(pwd)/gmp --with-gmp-lib=$(pwd)/gmp/.libs \
    --without-ppl --without-cloog
```

configure オプションの意味：

--enable-clocale=gnu
このオプションはあらゆる状況において C++ ライブラリに対するロケールモデルが正しく設定されるようにします。 configure スクリプト実行時に de_DE ロケールがインストール済みであることが分かれば、正しい GNU ロケールモデルが設定されます。 しかし de_DE ロケールがインストールされていなかったら、誤った汎用ロケールモデルが設定されてしまうため、アプリケーションバイナリインターフェース（Application Binary Interface; ABI）とは非互換の C++ ライブラリが生成されてしまう可能性があります。

--enable-threads=posix
マルチスレッドコードを扱う C++ の例外処理を有効にします。

--enable-__cxa_atexit
このオプションは atexit を使用せず __cxa_atexit の使用を有効にします。 これによりローカルなスタティックオブジェクトおよびグローバルオブジェクトに対する C++ デストラクタを登録します。 このオプションは、標準に完全準拠したデストラクタ実装のために必要です。 またこれは C++ ABI に影響するものであり C++ 共有ライブラリ、C++ プログラムを作り出し、他の Linux ディストリビューションとの互換性を実現します。

--enable-languages=c,c++
C と C++ の両コンパイラを生成することを指示します。

--disable-libstdcxx-pch
libstdc++ に対してプリコンパイルヘッダ（pre-compiled header; PCH）をビルドしないように指示します。 これを含めてしまうとサイズが増えることになり、そもそも利用する必要がありません。

--disable-bootstrap
GCC のネイティブビルドを行うには、デフォルトでは "ブートストラップ" ビルドを行ないます。 これは単に GCC をコンパイルするのではなく、数回のコンパイルを繰り返します。 つまり一回めにビルドされたプログラムを使って二回め、三回めのコンパイルを行うものです。 二回め、三回めとコンパイルを繰り返すのは、これによって自分自身を再生成して完璧なものを作り出すためです。 このことによってコンパイルが正確に行われたことを暗に示すことにもなります。 しかし LFS のビルドでは、何度もブートストラップを行う必要のない、手堅い(solid) コンパイラを作り出します。

パッケージをコンパイルします。

```
make
```

パッケージをインストールします。

```
make install
```

最後にシンボリックリンクを作成します。 プログラムやスクリプトの中には gcc ではなく cc を用いるものが結構あります。 シンボリックリンクを作ることで各種のプログラムを汎用的にすることができ、通常 GNU C コンパイラがインストールされていない多くの UNIX システムでも利用できるものになります。 cc を利用することにすれば、 システム管理者がどの C コンパイラをインストールすべきかを判断する必要がなくなります。

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

注意

この時点で、構築したツールチェーンの基本的な（コンパイルやリンクなどの）機能が正しく動作していることを確認する必要があります。 健全性検査（sanity check）を行うために以下を実行してください。

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

問題なく動作した場合はエラーがなかったということで、最後のコマンドから出力される結果は以下のようになるはずです。

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

ここでダイナミックリンカのディレクトリが /tools/lib であることを確認してください。 あるいは 64 ビットマシンであれば /tools/lib64 であることを確認してください。

コマンドの出力結果が上と異なっていたり、あるいは何も出力されなかった場合は、何かがおかしいことを意味します。 どこに問題があるのか調査・再試行を行って解消してください。 解決せずにこの先に進まないでください。 cc ではなく gcc を使って再度健全性検査を行ってみてください。 これで解決したなら /tools/bin/cc のシンボリックリンクが正しくないということです。 正しく生成し直してください。 また環境変数 PATH が正しいかどうかも確認してください。 echo $PATH を実行して、実行パスリストの先頭が /tools/bin であるかどうか確認します。 PATH が間違っていたなら、実はあなたは lfs ユーザーでログインしていないのかもしれませんし 4.4. 「環境設定」 での作業に間違いがあったのかもしれません。

すべてが終了したらテストファイルを削除します。

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

本パッケージの詳細は 6.16.2. 「GCC の構成」 を参照してください。

5.11. Tcl-8.5.8

Tcl パッケージはツールコマンド言語（Tool Command Language）を提供します。

概算ビルド時間:　　0.5 SBU
必要ディスク容量:　　32 MB

5.11.1. Tcl のインストール

本パッケージとこれに続く二つのパッケージ（Expect と DejaGNU）は、GCC および Binutils におけるテストスイートを実行するのに必要となるためインストールするものです。 テスト目的のためにこれら三つのパッケージをインストールするというのは、少々大げさなことかもしれません。 ただ本質的ではないことであっても、重要なツール類が正常に動作するという確認が得られれば安心できます。 本章ではテストスイートを実行することは必須ではないため、実行しないものとしていますが、それらの三つのパッケージは 第6章 で行うテストのために必要となるものです。

Tcl をコンパイルするための準備をします。

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

パッケージをビルドします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
TZ=UTC make test
```

Tcl のテストスイートは、特定のホスト環境において失敗することがありますが、その原因はよく分かっていません。したがってテストスイートの失敗は驚くことではなく、さして重大なことではありません。 TZ=UTC はタイムゾーンを協定世界時間（Coordinated Universal Time; UTC）あるいはグリニッジ標準時間としても知られる時間に設定します。ただしこれはテストスイートを実行する時だけの設定です。 こうしておけば時刻に関するテストが正しく処理されます。 環境変数 TZ については 第7章 にて詳しく説明しています。

パッケージをインストールします。

```
make install
```

インストールされたライブラリを書き込み可能にします。 こうすることで後にデバッグシンボルを削除できるようにします。

```
chmod -v u+w /tools/lib/libtcl8.5.so
```

Tcl のヘッダファイルをインストールします。 これらは次にビルドする Expect が必要とするファイルです。

```
make install-private-headers
```

必要となるシンボリックリンクを生成します。

```
ln -sv tclsh8.5 /tools/bin/tclsh
```

5.11.2. Tcl の構成

インストールプログラム:　　tclsh（tclsh8.5 へのリンク）, tclsh8.5
インストールライブラリ:　　libtcl8.5.so, libtclstub8.5.a

概略説明

tclsh8.5	Tcl コマンドシェル
tclsh	tclsh8.5 へのリンク
libtcl8.5.so	Tcl ライブラリ
libtclstub8.5.a	Tcl スタブライブラリ

5.12. Expect-5.44.1.15

Expect パッケージは、他のプログラムと対話的に処理を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 4.1 MB

5.12.1. Expect のインストール

Tk への依存性をなくすためのパッチを適用します。 Tk はホストシステム上に存在するかもしれませんが、chroot 環境に入った後には利用できなくなります。

```
patch -Np1 -i ../expect-5.44.1.15-no_tk-1.patch
```

Expect の configure スクリプトは、ホストシステムの /usr/local/bin/stty を利用しようとしますが、/bin/stty を利用するように修正します。 これを行うのは、ここで構築しているテストスイートのツール類を、ツールチェーンの最終構築まで正常動作してもらうために必要となるからです。

```
cp -v configure{,.orig}
sed 's:/usr/local/bin:/bin:' configure.orig > configure
```

Expect をコンパイルするための準備をします。

```
./configure --prefix=/tools --with-tcl=/tools/lib \
  --with-tclinclude=/tools/include --with-tk=no
```

configure オプションの意味：

--with-tcl=/tools/lib
Tcl のインストールモジュールを、ホストシステムに存在しているツール類の場所からではなく、一時的ツールを配置したディレクトリから探し出すことを指示します。

--with-tclinclude=/tools/include
Tcl の内部ヘッダファイルを探し出す場所を指定します。 configure は自動的には Tcl ヘッダファイルの場所を探し出さないため、これを明示します。

--with-tk=no
Tk（Tcl の GUI コンポーネント）や X ウィンドウシステムライブラリを検索しないことを指示します。 いずれもホストシステムに存在するかもしれませんが、 今作り出す一時システムには存在しません。

パッケージをビルドします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make test
```

Expect のテストスイートは、特定のホスト環境において失敗することがありますが、その原因はよく分かっていません。 したがってテストスイートの失敗は驚くことではなく、さして重大なことではありません。

パッケージをインストールします。

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

make パラメータの意味：

SCRIPTS=""
Expect の補助的なスクリプトはインストールしないことを指示します。 これらは必要ありません。

5.12.2. Expect の構成

インストールプログラム: expect
インストールライブラリ: libexpect-5.44.a

概略説明

expect	スクリプトを通じて他の対話的なプログラムとの処理を行います。
libexpect-5.44.a	Tcl 拡張機能を通じて、あるいは（Tcl がない場合に）C や C++ から直接、Expect とのやりとりを行う関数を提供します。

5.13. DejaGNU-1.4.4

DejaGNU パッケージは、他のプログラムをテストするフレームワークを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 6.1 MB

5.13.1. DejaGNUのインストール

本パッケージの最新バージョンは 2004 年にリリースされています。 それ以降に発生した修正を適用します。

```
patch -Np1 -i ../dejagnu-1.4.4-consolidated-1.patch
```

DejaGNU をコンパイルするための準備をします。

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

パッケージをビルドしてインストールします。

```
make install
```

コンパイル結果をテストするなら以下を実行します。

```
make check
```

5.13.2. DejaGNUの構成

インストールプログラム: runtest

概略説明

runtest expect シェルの適正な場所を特定し DejaGNU を実行するためのラッパースクリプト。

5.14. Ncurses-5.7

Ncurses パッケージは、端末に依存しない、文字ベースのスクリーン制御を行うライブラリを提供します。

概算ビルド時間: 0.7 SBU
必要ディスク容量: 30 MB

5.14.1. Ncurses のインストール

Ncurses をコンパイルするための準備をします。

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

configure オプションの意味：

--without-ada
このオプションは Ncurses に対して Ada コンパイラのサポート機能をビルドしないよう指示します。 この機能はホストシステムでは提供されているかもしれませんが、 chroot 環境に入ってしまうと利用できなくなります。

--enable-overwrite
このオプションは Ncurses のヘッダファイルを /tools/include/ncurses ではなく /tools/include にインストールすることを指示します。 これは他のパッケージが Ncurses のヘッダファイルを正しく見つけ出せるようにするためです。

パッケージをコンパイルします。

```
make
```

このパッケージにはテストスイートがありますが、インストールした後に実行しなければなりません。 テストスイートのためのファイル群はサブディレクトリ test/ 以下に残っています。 詳しいことはそのディレクトリ内にある README ファイルを参照してください。

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.19.2. 「Ncurses の構成」 を参照してください。

5.15. Bash-4.1

Bash は Bourne-Again SHell を提供します。

概算ビルド時間: 0.5 SBU
必要ディスク容量: 35 MB

5.15.1. Bash のインストール

Bash のアップストリームにより報告あるいは修正された数種のバグフィックスを適用します。

```
patch -Np1 -i ../bash-4.1-fixes-2.patch
```

Bash をコンパイルするための準備をします。

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

configure オプションの意味：

--without-bash-malloc
このオプションは Bash のメモリ割り当て関数（malloc）を利用しないことを指示します。 この関数はセグメンテーションフォールトが発生する可能性があるものとして知られています。 このオプションをオフにすることで、Bash は Glibc が提供する malloc 関数を用いるものとなり、そちらの方が安定しています。

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make tests
```

パッケージをインストールします。

```
make install
```

他のプログラム類がシェルとして sh を用いるものがあるためリンクを作ります。

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

本パッケージの詳細は 6.29.2. 「Bash の構成」 を参照してください。

5.16. Bzip2-1.0.5

Bzip2 パッケージはファイル圧縮、伸長（解凍）を行うプログラムを提供します。 テキストファイルであれば、これまでよく用いられてきた gzip に比べて bzip2 の方が圧縮率の高いファイルを生成できます。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 4.8 MB

5.16.1. Bzip2 のインストール

Bzip2 パッケージには configure がありません。 コンパイルおよびテストを行うには以下を実行します。

```
make
```

パッケージをインストールします。

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

本パッケージの詳細は 6.36.2. 「Bzip2 の構成」 を参照してください。

5.17. Coreutils-8.5

Coreutils パッケージはシステムの基本的な特性を表示したり設定したりするためのユーティリティを提供します。

概算ビルド時間: 0.7 SBU
必要ディスク容量: 88 MB

5.17.1. Coreutils のインストール

Coreutils をコンパイルするための準備をします。

```
./configure --prefix=/tools --enable-install-program=hostname
```

configure オプションの意味：

--enable-install-program=hostname
このオプションは hostname プログラムを生成しインストールすることを指示します。 このプログラムはデフォルトでは生成されません。そしてこれは Perl のテストスイートを実行するのに必要となります。

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make RUN_EXPENSIVE_TESTS=yes check
```

パラメータ RUN_EXPENSIVE_TESTS=yes は、テストスイートの実行にあたって（CPU パワーとメモリ使用量の観点で）比較的負荷の高いテストを追加で実行することを指示します。 特定のプラットフォームに対してのテスト確認となりますが、一般的に Linux 上において支障はありません。

パッケージをインストールします。

```
make install
```

上のコマンド実行では su がインストールされません。 一般ユーザーではこのプログラムを root 権限でインストールできないためです。 別名ファイルを作り出して手動でインストールすることで、最終的に構築するシステムでもテストの実行を一般ユーザーにより行います。 またホストシステムにある su コマンドは PATH 変数上に保持しておき可能な限り利用します。 上を行うために以下を実行します。

```
cp -v src/su /tools/bin/su-tools
```

本パッケージの詳細は 6.22.2. 「Coreutils の構成」 を参照してください。

5.18. Diffutils-3.0

Diffutils パッケージはファイルやディレクトリの差分を表示するプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 6.1 MB

5.18.1. Diffutils のインストール

Diffutils をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.37.2. 「Diffutils の構成」 を参照してください。

5.19. File-5.04

File パッケージは、指定されたファイルの種類を決定するユーティリティを提供します。

概算ビルド時間:　　0.2 SBU
必要ディスク容量:　　9.5 MB

5.19.1. File のインストール

File をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.39.2. 「File の構成」 を参照してください。

5.20. Findutils-4.4.2

Findutils パッケージはファイル検索を行うプログラムを提供します。 このプログラムはディレクトリツリーを再帰的に検索したり、データベースの生成・保守・検索を行います。（データベースによる検索は再帰的検索に比べて処理速度は速いものですが、データベースが最新のものに更新されていない場合は信頼できない結果となります。）

概算ビルド時間: 0.3 SBU
必要ディスク容量: 20 MB

5.20.1. Findutils のインストール

Findutils をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.40.2. 「Findutils の構成」 を参照してください。

5.21. Gawk-3.1.8

Gawk パッケージはテキストファイルを操作するプログラムを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 19 MB

5.21.1. Gawk のインストール

Gawk をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.38.2. 「Gawk の構成」 を参照してください。

5.22. Gettext-0.18.1.1

Gettext パッケージは国際化を行うユーティリティを提供します。 各種プログラムに対して NLS (Native Language Support) を含めてコンパイルすることができます。 つまり各言語による出力メッセージが得られることになります。

概算ビルド時間: 0.8 SBU
必要ディスク容量: 82 MB

5.22.1. Gettext のインストール

ここで構築している一時的なツールに際して、Gettext パッケージからは1つのバイナリをビルドしてインストールするだけで十分です。

Gettext をコンパイルするための準備をします。

```
cd gettext-tools
./configure --prefix=/tools --disable-shared
```

configure オプションの意味：

--disable-shared
Gettext の共有ライブラリはこの時点では必要でないため、それらをビルドしないようにします。

パッケージをコンパイルします。

```
make -C gnulib-lib
make -C src msgfmt
```

1つのバイナリしかコンパイルしなかったため、その他のライブラリをコンパイルしない限り、テストスイートを成功させることはできません。 したがってテストスイートをこの段階で実行することはお勧めしません。

msgfmt プログラムをインストールします。

```
cp -v src/msgfmt /tools/bin
```

本パッケージの詳細は 6.42.2. 「Gettext の構成」 を参照してください。

5.23. Grep-2.6.3

Grep パッケージはファイル内の検索を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 6.7 MB

5.23.1. Grep のインストール

Grep をコンパイルするための準備をします。

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

configure オプションの意味：

--disable-perl-regexp
このオプションは grep プログラムに対して Perl 互換正規表現 (Perl Compatible Regular Expression; PCRE) ライブラリをリンクしないように指示します。 このライブラリはホストシステムに存在するかもしれませんが chroot 環境に入ってしまうと利用できなくなります。

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.27.2. 「Grep の構成」 を参照してください。

5.24. Gzip-1.4

Gzip パッケージはファイルの圧縮、伸長（解凍）を行うプログラムを提供します。

概算ビルド時間:　　0.1 SBU 以下
必要ディスク容量:　　3.3 MB

5.24.1. Gzip のインストール

Gzip をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.45.2. 「Gzip の構成」 を参照してください。

5.25. M4-1.4.14

M4 パッケージはマクロプロセッサを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 11.6 MB

5.25.1. M4 のインストール

不足しているインクルードディレクティブを追加します。 これがないと、M4 が Glibc-2.12.1 においてビルドできません。

```
sed -i -e '/"m4.h"/a\
#include <sys/stat.h>' src/path.c
```

M4 をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.24.2. 「M4 の構成」 を参照してください。

5.26. Make-3.82

Make パッケージは、 パッケージ類をコンパイルするためのプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 9.6 MB

5.26.1. Make のインストール

Make をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.49.2. 「Make の構成」 を参照してください。

5.27. Patch-2.6.1

Patch パッケージは「パッチ」ファイルを適用することにより、ファイルの修正・生成を行うプログラムを提供します。 「パッチ」ファイルは diff プログラムにより生成されます。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 1.9 MB

5.27.1. Patch のインストール

Patch をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.52.2. 「Patch の構成」 を参照してください。

5.28. Perl-5.12.1

Perl パッケージは Perl 言語 (Practical Extraction and Report Language) を提供します。

概算ビルド時間: 0.8 SBU
必要ディスク容量: 106 MB

5.28.1. Perl のインストール

以下のパッチを適用します。 これは C ライブラリに対する固定的なパスを適用します。

```
patch -Np1 -i ../perl-5.12.1-libc-1.patch
```

Perl をコンパイルするための準備をします。(以下のコマンドにて 'Data/Dumper Fcntl IO' の部分は間違いなく入力してください。 それらはすべて英字です。)

```
sh Configure -des -Dprefix=/tools \
                  -Dstatic_ext='Data/Dumper Fcntl IO'
```

configure オプションの意味:

-Dstatic_ext='Data/Dumper Fcntl IO POSIX'
次章での Coreutils と Glibc のインストールとテストのためには、最低限の静的拡張モジュール (static extensions) さえあれば十分です。 そこで静的拡張モジュールのビルドを指示します。

本パッケージにてビルドに必要となるのは、数個のユーティリティとライブラリだけです。

```
make perl utilities ext/Errno/pm_to_blib
```

Perl にはテストスイートがありますが、この時点での実行はお勧めしません。 Perl を部分的にしかビルドしていない状態で make test を実行すると、他の実行ファイルなどもビルドすることになってしまいます。 それらは今の時点では必要ありません。 テストスイートを実行したい場合は次章にて行ってください。

これらのツールとライブラリをインストールします。

```
cp -v perl pod/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.12.1
cp -Rv lib/* /tools/lib/perl5/5.12.1
```

本パッケージの詳細は 6.33.2. 「Perl の構成」 を参照してください。

5.29. Sed-4.2.1

Sed パッケージはストリームエディタを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 8.0 MB

5.29.1. Sed のインストール

Sed をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.17.2. 「Sed の構成」 を参照してください。

5.30. Tar-1.23

Tar パッケージはアーカイブプログラムを提供します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 20.9 MB

5.30.1. Tar のインストール

最新ソースにて発生するバグを修正します。

```
sed -i /SIGPIPE/d src/tar.c
```

Tar をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.57.2. 「Tar の構成」 を参照してください。

5.31. Texinfo-4.13a

Texinfo パッケージは info ページへの読み書き・変換を行うプログラムを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 20 MB

5.31.1. Texinfo のインストール

Texinfo をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイルが終了しました。 前にも述べたように、この章にて一時的ツールのテストスイートを実行することは必須ではありません。 しかしテストスイートを実行するなら、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

本パッケージの詳細は 6.58.2. 「Texinfo の構成」 を参照してください。

5.32. ストリップ

本節に示す作業は必須ではありません。 ただ LFS パーティションの容量が比較的少ない場合には、不要なものは削除することを覚えておきましょう。 ここまでにビルドしてきた実行ファイルやライブラリには、合計で 70 MB ほどの不要なデバッグシンボル情報が含まれています。 それらを取り除くには以下を実行します。

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

上のコマンド実行ではいくつものファイルがフォーマット不明となって処理がスキップされます。 それらはたいてい、バイナリではなくスクリプトであることを示しています。

--strip-unneeded パラメータは 絶対に ライブラリに対して用いないでください。 もし用いるとスタティックライブラリが破壊され、ツールチェーンを構成するパッケージをすべて作り直さなければならなくなります。

さらに 25 MB ほどを節約するためにドキュメント類を削除します。

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

この時点において環境変数 $LFS の配下には最低でも 850 MB の空き容量が必要になります。 これは次のフェーズにて Glibc をビルドしインストールするためです。 Glibc のビルドとインストールができさえすれば、残りのものもすべてビルド、インストールができます。

5.33. 所有者の変更

注記

本書のこれ以降で実行するコマンドはすべて root ユーザーでログインして実行します。 もう lfs ユーザーは不要です。 root ユーザーの環境にて環境変数 $LFS がセットされていることを今一度確認してください。

$LFS/tools ディレクトリの所有者は今は lfs ユーザーであり、これはホストシステム上に存在するユーザーです。 この $LFS/tools ディレクトリをこのままにしておくということは、そこにあるファイル群が、存在しないアカウントに対するユーザーIDによって所有される形を生み出すことになります。 これは危険なことです。 後にユーザーアカウントが生成され同一のユーザーIDを持ったとすると $LFS/tools の所有者となってしまい、そこにあるファイルすべてを所有することになって、悪意のある操作に利用されてしまいます。

この問題を解消するためには、新しく作り出される LFS システムに lfs ユーザーを作成することが考えられます。 その場合には同一のユーザーID、グループIDとなるように作ります。 もっと良い方法があります。 $LFS/tools ディレクトリの所有者を root ユーザーにすることです。以下のコマンドによりこれを実現します。

```
chown -R root:root $LFS/tools
```

$LFS/tools ディレクトリは LFS システムの構築作業を終えれば削除することができます。 一方これを残しておいて本書と同一バージョンの LFS システムを新たに構築する際に利用することもできます。 $LFS/tools ディレクトリをどのように残すかは読者の皆さんの好みに応じて取り決めてください。

注意

この先の LFS システム構築に向けて一時的なツール類を残しておきたい場合は この時点で バックアップを取っておくのが良いでしょう。 第6章で実施する作業を通じて、今存在している一時的ツールは変更が加えられますので、将来、別のビルド作業を行う際には使えないものとなります。

第III部 LFSシステムの構築

第6章 基本的なソフトウェアのインストール

6.1. はじめに

この章ではビルド環境に入って正式な LFS システムの構築作業を始めます。 chroot によって一時的なミニ Linux システムへ移行し、準備作業を行った上でパッケージ類のインストールを行っていきます。

パッケージ類のインストール作業は簡単なものです。 インストール手順の説明は、たいていは手短に一般的なものだけで済ますこともできます。 ただ誤りの可能性を極力減らすために、個々のインストール手順の説明は十分に行うことにします。 Linux システムがどのようにして動作しているかを学ぶには、個々のパッケージが何のために用いられていて、なぜユーザー（あるいはシステム）がそれを必要としているのかを知ることが重要になります。

コンパイラには最適化オプションがありますが、これを利用することはお勧めしません。 コンパイラの最適化を用いればプログラムが若干速くなる場合もありますが、そもそもコンパイルが出来なかったり、プログラムの実行時に問題が発生したりする場合があります。 もしコンパイラの最適化によってパッケージビルドが出来なかったら、最適化をなしにしてもう一度コンパイルすることで解決するかどうかを確認してください。 最適化を行ってパッケージがコンパイル出来たとしても、コードとビルドツールの複雑な関連に起因してコンパイルが適切に行われないリスクをはらんでいます。 また -march オプションや -mtune オプションにて指定する値は、本書には明示しておらずテストも行っていませんので注意してください。 これらはツールチェーンパッケージ（Binutils、GCC、Glibc）に影響を及ぼすことがあります。 最適化オプションを用いることによって得られるものがあったとしても、それ以上にリスクを伴うことがしばしばです。 初めて LFS 構築を手がける方は、最適化オプションをなしにすることをお勧めします。 これ以降にビルドしていくツール類は、それでも十分に速く安定して動作するはずです。

本章にてインストールしていくパッケージ類のビルド順は、必ず本書どおりに行ってください。 プログラムはすべて /tools ディレクトリを直接参照するような形でビルドしてはなりません。 また同じ理由でパッケージ類を同時並行でビルドしないでください。 特にデュアル CPU マシンにおいて同時にビルドしていくと時間の節約を図ることができますが /tools ディレクトリを直接参照するプログラムが出来上がってしまい、このディレクトリが存在しなくなった時にはプログラムが動作しないことになります。

各ページではインストール手順の説明よりも前に、パッケージの内容やそこに何が含まれているかを簡単に説明し、ビルドにどれくらいの時間を要するか、ビルド時に必要となるディスク容量はどれくらいかを示しています。 またインストール手順の最後には、パッケージがインストールするプログラムやライブラリの一覧を示し、それらがどのようなものかを簡単に説明しています。

注記

本章にて導入するパッケージにおいて SBU 値と必要ディスク容量には、テストスイート実施による時間や容量をすべて含んでいます。

6.2. 仮想カーネルファイルシステムの準備

カーネルが取り扱う様々なファイルシステムは、カーネルとの間でやり取りが行われます。 これらのファイルシステムは仮想的なものであり、ディスクを消費するものではありません。 ファイルシステムの内容はメモリ上に保持されます。

ファイルシステムをマウントするディレクトリを以下のようにして生成します。

```
mkdir -v $LFS/{dev,proc,sys}
```

6.2.1. 初期デバイスノードの生成

カーネルがシステムを起動する際には、いくつかのデバイスノードの存在が必要です。 特に console と null です。デバイスノードはハードディスク上に生成されます。 そして udevd が起動し、また Linux が起動パラメータ init=/bin/bash によって起動されれば利用可能となります。 以下のコマンドによりデバイスノードを生成します。

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

6.2.2. /dev のマウントと有効化

各デバイスを /dev に設定する方法としては、/dev ディレクトリに対して tmpfs のような仮想ファイルシステムをマウントすることが推奨されます。 こうすることで各デバイスが検出されアクセスされる際に、その仮想ファイルシステム上にて動的にデバイスを生成する形を取ることができます。 このデバイス生成処理は一般的にはシステム起動時

に Udev によって行われます。 今構築中のシステムにはまだ Udev を導入していませんし、再起動も行っていませんので /dev のマウントと有効化は手動で行ないます。 これはホストシステムの /dev ディレクトリに対して、バインドマウントを行うことで実現します。 バインドマウント（bind mount）は特殊なマウント方法の一つで、ディレクトリのミラーを生成したり、他のディレクトリへのマウントポイントを生成したりします。 以下のコマンドにより実現します。

```
mount -v --bind /dev $LFS/dev
```

6.2.3. 仮想カーネルファイルシステムのマウント

残りの仮想カーネルファイルシステムを以下のようにしてマウントします。

```
mount -vt devpts devpts $LFS/dev/pts
mount -vt tmpfs shm $LFS/dev/shm
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
```

6.3. パッケージ管理

パッケージ管理についての説明を LFS ブックに加えて欲しいとの要望をよく頂きます。 パッケージ管理ツールがあれば、インストールされるファイル類を管理し、パッケージの削除やアップグレードを容易に実現できます。 パッケージ管理ツールでは、バイナリファイルやライブラリファイルだけでなく、設定ファイル類のインストールも取り扱います。 パッケージ管理ツールをどうしたら・・・ いえいえ本節は特定のパッケージ管理ツールを説明するわけでなく、その利用を勧めるものでもありません。 もっと広い意味で、管理手法にはどういったものがあり、どのように動作するかを説明します。 あなたにとって最適なパッケージ管理がこの中にあるかもしれません。 あるいはそれらをいくつか組み合わせて実施することになるかもしれません。 本節ではパッケージのアップグレードを行う際に発生する問題についても触れます。

LFS や BLFS において、パッケージ管理ツールについて触れていない理由には以下のものがあります。

- 本書の目的は Linux システムがいかに構築されているかを学ぶことです。 パッケージ管理はその目的からはずれてしまいます。
- パッケージ管理についてはいくつもの方法があり、それらには一長一短があります。 ユーザーに対して満足のいくものを選び出すのは困難です。

ヒントプロジェクト（Hints Project）ページに、パッケージ管理についての情報が示されています。 それらが望むものかどうか確認してみてください。

6.3.1. アップグレードに関する問題

パッケージ管理ツールがあれば、各種ソフトウェアの最新版がリリースされた際に容易にアップグレードができます。全般に LFS ブックや BLFS ブックに示されている作業手順に従えば、新しいバージョンへのアップグレードを行っていくことはできます。 以下ではパッケージをアップグレードする際に注意すべき点、特に稼動中のシステムに対して実施するポイントについて説明します。

- ツールチェーン（Glibc、GCC、Binutils）のいずれかについて、マイナーバージョンをアップグレードする必要がある場合は、LFS を再構築するのが無難です。 この場合、すべてのパッケージの依存関係を考慮して順番に作り直せば実現できるはず ですが、これはあまりお勧めしません。 例えば glibc-2.2.x を glibc-2.3.x にアップグレードする必要がある場合は、再構築するのが無難です。 マイクロバージョンをアップグレードする場合は、もっと単純にそのパッケージをインストールし直すだけで動作すると思いますが、保証はありません。 例えば glibc-2.3.4 を glibc-2.3.5 にアップグレードする場合、普通は何も問題ないでしょう。
- 共有ライブラリを提供しているパッケージをアップデートする場合で、そのライブラリの名前が変更になった場合は、そのライブラリを動的にリンクしているすべてのパッケージは、新しいライブラリにリンクされるように再コンパイルを行う必要があります。（パッケージのバージョンとライブラリ名との間には相関関係はありません。）例えば foo-1.2.3 というパッケージが共有ライブラリ libfoo.so.1 をインストールするものであるとします。 そして今、新しいバージョン foo-1.2.4 にアップグレードし、共有ライブラリ libfoo.so.2 をインストールするとします。 この例では libfoo.so.1 を動的にリンクいるパッケージがあったとすると、それらはすべて libfoo.so.2 に対してリンクするよう再コンパイルしなければなりません。 古いライブラリに依存しているパッケージすべてを再コンパイルするまでは、そのライブラリを削除するべきではありません。

6.3.2. パッケージ管理手法

以下に一般的なパッケージ管理手法について示します。 パッケージ管理マネージャを用いる前に、様々な方法を検討し、特にそれぞれの欠点も確認してください。

6.3.2.1. すべては頭の中で

そうです。 これもパッケージ管理のやり方の一つです。 いろいろなパッケージに精通していて、どんなファイルがインストールされるか分かっている人もいます。 そんな人はパッケージ管理ツールを必要としません。 あるいはパッケージが更新された際に、システム全体を再構築しようと考えている人なら、やはりパッケージ管理ツールを必要としません。

6.3.2.2. 異なるディレクトリへのインストール

これは最も単純なパッケージ管理のやり方であり、パッケージ管理のためのツールを用いる必要はありません。 個々のパッケージを個別のディレクトリにインストールする方法です。 例えば foo-1.1 というパッケージを /usr/pkg/foo-1.1 ディレクトリにインストールし、この /usr/pkg/foo-1.1 に対するシンボリックリンク /usr/pkg/foo を作成します。 このパッケージの新しいバージョン foo-1.2 をインストールする際には /usr/pkg/foo-1.2 ディレクトリにインストールした上で、先ほどのシンボリックリンクをこのディレクトリを指し示すように置き換えます。

PATH、 LD_LIBRARY_PATH、 MANPATH、 INFOPATH、 CPPFLAGS といった環境変数に対しては /usr/pkg/foo ディレクトリを加える必要があるかもしれません。 もっともパッケージによっては、このやり方では管理できないものもあります。

6.3.2.3. シンボリックリンク方式による管理

これは一つ前に示したパッケージ管理テクニックの応用です。 各パッケージは同様にインストールします。 ただし先ほどのようなシンボリックリンクを生成するのではなく /usr ディレクトリ階層の中に各ファイルのシンボリックリンクを生成します。 この方法であれば環境変数を追加設定する必要がなくなります。 シンボリック・リンクを自動生成することもできますが、パッケージ管理ツールの中にはこの手法を使って構築されているものもあります。 よく知られているものとして Stow、Epkg、Graft、Depot があります。

インストール時には意図的な指示が必要です。 パッケージにとっては /usr にインストールすることが指定されたものとなりますが、実際には /usr/pkg 配下にインストールされるわけです。 このインストール方法は単純なものではありません。 例えば今 libfoo-1.1 というパッケージをインストールするものとします。 以下のようなコマンドでは、このパッケージを正しくインストールできません。

```
./configure --prefix=/usr/pkg/libfoo/1.1
make
make install
```

インストール自体は動作しますが、このパッケージに依存している他のパッケージは、期待どおりには libfoo を正しくリンクしません。 例えば libfoo をリンクするパッケージをコンパイルする際には /usr/lib/libfoo.so.1 がリンクされると思うかもしれませんが、実際には /usr/pkg/libfoo/1.1/lib/libfoo.so.1 がリンクされることになります。 正しくリンクするためには DESTDIR 変数を使って、パッケージのインストールをうまく仕組む必要があります。 この方法は以下のようにして行います。

```
./configure --prefix=/usr
make
make DESTDIR=/usr/pkg/libfoo/1.1 install
```

多くのパッケージは、たいていはこの手法をサポートしていますが、そうでないものもあります。 この手法を取り入れていないパッケージに対しては、手作業にてインストールすることが必要になります。 またはそういった問題を抱えるパッケージであれば /opt ディレクトリにインストールする方が容易なことかもしれません。

6.3.2.4. タイムスタンプによる管理方法

この方法ではパッケージをインストールするにあたって、あるファイルにタイムスタンプが記されます。 インストールの直後に find コマンドを適当なオプション指定により用いることで、インストールされるすべてのファイルのログが生成されます。 これはタイムスタンプファイルの生成の後に行われます。 この方法を用いたパッケージ管理ツールとして install-log があります。

この方法はシンプルである利点がありますが、以下の二つの欠点があります。 インストールの際に、いずれかのファイルのタイムスタンプが現在時刻でなかった場合、そういったファイルはパッケージ管理ツールが正しく制御できません。 またこの方法は一つのパッケージだけが、その時にインストールされることを前提とします。 例えば二つのパッケージが二つの異なる端末から同時にインストールされるような場合は、ログファイルが適切に生成されません。

6.3.2.5. インストールスクリプトの追跡管理

この方法はインストールスクリプトが実行するコマンドを記録するものです。 これには以下の二種類の手法があります。

環境変数 LD_PRELOAD を使えば、インストール前にあらかじめロードされるライブラリを定めることができます。 パッケージのインストール中には cp、 install、 mv など様々な実行モジュールにそのライブラリをリンクさせ、ファイルシステムを変更するようなシステムコールを監視することで、そのライブラリがパッケージを追跡管理できるようになります。 この方法を実現するためには、動的リンクする実行モジュールはすべて suid ビット、sgid ビットがオフでなければなりません。 事前にライブラリをロードしておくと、インストール中に予期しない副作用が発生するかもしれません。 したがって、ある程度のテスト確認を行って、パッケージ管理ツールが不具合を引き起こさないこと、しかるべきファイルの記録を取っておくことが必要とされます。

二つめの方法は strace を用いるものです。 これはインストールスクリプトの実行中に発生するシステムコールを記録するものです。

6.3.2.6. パッケージのアーカイブを生成する方法

この方法では、シンボリックリンク方式によるパッケージ管理にて説明したのと同じように、パッケージが個別のディレクトリにインストールされます。 インストールされた後には、インストールファイルを使ってアーカイブが生成されます。 このアーカイブはこの後に、ローカルPCへのインストールに用いられ、他のPCのインストールに利用することもできます。

商用ディストリビューションが採用しているパッケージ管理ツールは、ほとんどがこの方法によるものです。 この方法に従ったパッケージ管理ツールの例に RPM があります。（これは Linux Standard Base Specification が規定しています。）また pkg-utils、Debian の apt、Gentoo の Portage システムがあります。 このパッケージ管理手法を LFS システムに適用するヒント情報が http://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt にあります。

パッケージファイルにその依存パッケージ情報まで含めてアーカイブ生成することは、非常に複雑となり LFS の範疇を超えるものです。

Slackware は、パッケージアーカイブに対して tar ベースのシステムを利用しています。 他のパッケージ管理ツールはパッケージの依存性を取り扱いますが、このシステムは意図的にこれを行っていません。 Slackware のパッケージ管理に関する詳細は http://www.slackbook.org/html/package-management.html を参照してください。

6.3.2.7. ユーザー情報をベースとする管理方法

この手法は LFS に固有のものであり Matthias Benkmann により考案されました。 ヒントプロジェクト（Hints Project）から入手することが出来ます。 考え方としては、各パッケージを個々のユーザーが共有ディレクトリにインストールします。 パッケージに属するファイル類は、ユーザーIDを確認することで容易に特定出来るようになります。 この手法の特徴や短所については、複雑な話となるため本節では説明しません。 詳しくは http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt に示されているヒントを参照してください。

6.3.3. 他システムへの LFS の配置

LFS システムの利点の一つとして、どのファイルもディスク上のどこに位置していても構わないことです。 他のコンピュータに対してビルドした LFS の複製を作ろうとするなら、それが同等のアーキテクチャであれば容易に実現できます。 つまり tar コマンドを使って LFS のルートディレクトリを含むパーティション（LFS の基本的なビルドの場合、非圧縮で 250MB 程度）をまとめ、これをネットワーク転送か、あるいは CD-ROM を通じて新しいシステムにコピーし、伸張（解凍）するだけです。 この場合でも、設定ファイルはいくらか変更することが必要です。 変更が必要となる設定ファイルは以下のとおりです。 /etc/hosts、 /etc/fstab、 /etc/passwd、 /etc/group、 /etc/shadow、 /etc/ld.so.conf、 /etc/scsi_id.config、 /etc/sysconfig/network、 /etc/sysconfig/network-devices/ifconfig.eth0/ipv4

新しいシステムのハードウェアと元のカーネルに差異があるかもしれないため、カーネルを再ビルドする必要があるでしょう。

最後に新システムを起動可能とするために 8.4. 「GRUB を用いたブートプロセスの設定」 を設定する必要があります。

6.4. Chroot 環境への移行

chroot 環境に入って最終的な LFS システムの構築、インストールを行っていきます。 root ユーザーになって以下のコマンドを実行します。 chroot 環境内は、この時点では一時的なツール類のみが利用可能な状態です。

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

env コマンドの -i パラメータは、chroot 環境での変数定義をすべてクリアするものです。 そして HOME, TERM, PS1, PATH という変数だけここで定義し直します。 TERM=$TERM は chroot 環境に入る前と同じ値を TERM 変数に与えます。 この設定は vim や less のようなプログラムの処理が適切に行われるために必要となります。 これ以外の変数として CFLAGS や CXXFLAGS などが必要であれば、ここで定義しておくと良いでしょう。

ここから先は LFS 変数は不要となります。 すべての作業は LFS ファイル・システム内で行っていくことになるからです。 起動される Bash シェルは $LFS ディレクトリがルート（/ ディレクトリ）となって動作します。

/tools/bin が PATH 変数内の最後に加わっています。 一時的なツール類は、それぞれの正式版がインストールされていくに従って使われなくなります。 これがうまく動作するのは bash の +h オプションを用いることによってハッシュ機能をオフにしているからであり、実行モジュールの場所を覚えておく機能を無効にしているからです。

bash のプロンプトに I have no name! と表示されますがこれは正常です。 この時点ではまだ /etc/passwd を生成していないからです。

注記

本章のこれ以降と次章では、すべてのコマンドを chroot 環境内にて実行することが必要です。 例えばシステムを再起動する場合のように chroot 環境からいったん抜け出した場合には、 6.2.2. 「/dev のマウントと有効化」 と 6.2.3. 「仮想カーネルファイルシステムのマウント」 にて説明した仮想カーネル・ファイル・システムがマウントされていることを確認してください。 そして chroot 環境に入り直してからインストール作業を再開してください。

6.5. ディレクトリの生成

LFS ファイルシステムにおけるディレクトリ構成を作り出していきます。 以下のコマンドを実行して標準的なディレクトリを生成します。

```
mkdir -pv /{bin,boot,etc/opt,home,lib,mnt,opt}
mkdir -pv /{media/{floppy,cdrom},sbin,srv,var}
install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv /usr/{,local/}share/{doc,info,locale,man}
mkdir -v  /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
for dir in /usr /usr/local; do
  ln -sv share/{man,doc,info} $dir
done
case $(uname -m) in
 x86_64) ln -sv lib /lib64 && ln -sv lib /usr/lib64 ;;
esac
mkdir -v /var/{lock,log,mail,run,spool}
mkdir -pv /var/{opt,cache,lib/{misc,locate},local}
```

ディレクトリは標準ではパーミッションモード 755 で生成されますが、すべてのディレクトリをこのままとするのは適当ではありません。 上のコマンド実行ではパーミッションを変更している箇所が二つあります。 一つは root ユーザーのホームディレクトリに対してであり、もう一つはテンポラリディレクトリに対してです。

パーミッションモードを変更している一つめは /root ディレクトリに対して、他のユーザーによるアクセスを制限するためです。 通常のユーザーが持つ、自分自身のホームディレクトリへのアクセス権設定と同じことを行ないます。 二つめのモード変更は /tmp ディレクトリや /var/tmp ディレクトリに対して、どのユーザーも書き込み可能とし、ただし他のユーザーが作成したファイルは削除できないようにします。 ビットマスク 1777 の最上位ビット、いわゆる「スティッキービット（sticky bit)」 を用いて実現します。

6.5.1. FHS コンプライアンス情報

本書のディレクトリ構成は標準ファイルシステム構成（Filesystem Hierarchy Standard; FHS）に基づいています。（その情報は http://www.pathname.com/fhs/ に示されています。）FHS に加えて man、 doc、 info の各ディレクトリに対するシンボリックリンクも作成します。 これは多くのパッケージがドキュメントファイルをインストールする先として /usr/share/<ディレクトリ> や /usr/local/share/<ディレクトリ> ではなく、いまだに /usr/<ディレクトリ> や /usr/local/<ディレクトリ> としているためです。 また FHS では /usr/local/games や /usr/share/games を規定していますが、一方で /usr/local/share については明確なものがありません。 したがって本書では必要なディレクトリのみを作成していくことにします。 もっとも FHS に準拠した構成を望むなら、どうぞ自由に作成してください。

6.6. 基本的なファイルとリンクの生成

プログラムの中には固定的に他のプログラムへのパスを保持しているものがあります。 そのパスは今の時点ではまだ存在していません。 このようなプログラムを正しく動作させるため、シンボリックリンクをいくつか作成します。 このリンクは本章の作業を通じて各種ソフトウェアをインストールしていくことで、 その実体であるファイルに置き換えられていきます。

```
ln -sv /tools/bin/{bash,cat,echo,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv /tools/lib/libstdc++.so{,.6} /usr/lib
ln -sv bash /bin/sh
```

Linux システムが適切に動作しているなら、マウントしているファイルシステムの情報を /etc/mtab ファイルに保持しています。 このファイルは普通は、新しいファイルシステムをマウントした際に生成されます。 しかし今の我々の chroot 環境では、ファイルシステムを一つもマウントしていません。 そこで、このファイルの存在を前提としているプログラムを正しく動作させるため、空の /etc/mtab を作成しておきます。

```
touch /etc/mtab
```

root ユーザーがログインできるように、またその 「root」 という名称を認識できるように /etc/passwd ファイルと /etc/group ファイルには該当する情報が登録されている必要があります。

以下のコマンドを実行して /etc/passwd ファイルを生成します。

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/dev/null:/bin/false
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
EOF
```

root ユーザーに対する本当のパスワードは後に定めます。 (「x」 は単に場所を設けるために設定しているものです。)

以下のコマンドを実行して /etc/group ファイルを生成します。

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
mail:x:34:
nogroup:x:99:
EOF
```

作成するグループは何かの標準に基づいたものではありません。 一部は本章の Udev の設定に必要となるものですし、一部は既存の Linux ディストリビューションが採用している慣用的なものです。 Linux Standard Base (http://www.linuxbase.org 参照) では root グループのグループID (GID) は 0、 bin グループの GID は 1 を定めているにすぎません。 他のグループとその GID はシステム管理者が自由に取り決めることができます。 というのも通常のプログラムであれば GID の値に依存することはなく、あくまでグループ名を用いてプログラミングされているからです。

プロンプトに表示される 「I have no name!」 を正しくするため、新たなシェルを起動します。 第5章 にて完全に Glibc をインストールし、 /etc/passwd ファイルと /etc/group ファイルを作ったので、ユーザー名とグループ名の名前解決が適切に動作します。

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

ディレクティブ +h について触れておきます。 これは bash に対して実行パスの内部ハッシュ機能を利用しないよう指示するものです。 もしこのディレクティブを指定しなかった場合 bash は一度実行したファイルのパスを記憶します。コンパイルしてインストールした実行ファイルはすぐに利用していくために、本章での作業では +h ディレクティブを常に使っていくことにします。

login、 agetty、 init といったプログラム（あるいは他のプログラム）は、システムに誰がいつログインしたかといった情報を多くのログファイルに記録します。 しかしログファイルがあらかじめ存在していない場合は、ログファイルの出力が行われません。 そこでそのようなログファイルを作成し、適切なパーミッションを与えます。

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

/var/run/utmp ファイルは現在ログインしているユーザーの情報を保持します。 /var/log/wtmp ファイルはすべてのログイン・ログアウトの情報を保持します。 /var/log/lastlog ファイルは各ユーザーが最後にログインした情報を保持します。 /var/log/btmp ファイルは不正なログイン情報を保持します。

6.7. Linux-2.6.35.4 API ヘッダ

Linux API ヘッダは Glibc が利用するカーネル API を提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 466 MB

6.7.1. Linux API ヘッダのインストール

Linux カーネルはアプリケーションプログラミングインターフェース（Application Programming Interface）を、システムの C ライブラリ（LFS の場合 Glibc）に対して提供する必要があります。 これを行うには Linux カーネルのソースに含まれる、さまざまな C ヘッダファイルを「健全化（sanitizing)」して利用します。

これより前に一度処理を行っていたとしても、不適切なファイルや誤った依存関係を残さないように、以下を処理します。

```
make mrproper
```

そしてユーザーが利用するカーネルヘッダファイルをテストし、ソースから抽出します。 それらはいったん中間的なローカルディレクトリに置かれ、必要な場所にコピーされます。 ターゲットディレクトリに既にあるファイルは削除されてからソースからの抽出処理が行われます。 なおファイルの中にはカーネル開発者が隠しファイルとしているものがあります。 それらは LFS では必要ないため、中間ディレクトリから削除します。

```
make headers_check
make INSTALL_HDR_PATH=dest headers_install
find dest/include \( -name .install -o -name ..install.cmd \) -delete
cp -rv dest/include/* /usr/include
```

6.7.2. Linux API ヘッダの構成

インストールヘッダ: /usr/include/asm/*.h, /usr/include/asm-generic/*.h, /usr/include/drm/*.h, /usr/include/linux/*.h, /usr/include/mtd/*.h, /usr/include/rdma/*.h, /usr/include/scsi/*.h, /usr/include/sound/*.h, /usr/include/video/*.h, /usr/include/xen/*.h

インストールディレクトリ: /usr/include/asm, /usr/include/asm-generic, /usr/include/drm, /usr/include/linux, /usr/include/mtd, /usr/include/rdma, /usr/include/scsi, /usr/include/sound, /usr/include/video, /usr/include/xen

概略説明

/usr/include/asm/*.h	The Linux API ASM ヘッダファイル
/usr/include/asm-generic/*.h	The Linux API ASM の汎用的なヘッダファイル
/usr/include/drm/*.h	The Linux API DRM ヘッダファイル
/usr/include/linux/*.h	The Linux API Linux ヘッダファイル
/usr/include/mtd/*.h	The Linux API MTD ヘッダファイル
/usr/include/rdma/*.h	The Linux API RDMA ヘッダファイル
/usr/include/scsi/*.h	The Linux API SCSI ヘッダファイル
/usr/include/sound/*.h	The Linux API Sound ヘッダファイル
/usr/include/video/*.h	The Linux API Video ヘッダファイル
/usr/include/xen/*.h	The Linux API Xen ヘッダファイル

6.8. Man-pages-3.25

Man-pages パッケージは 1,900 以上のマニュアルページを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 21 MB

6.8.1. Man-pages のインストール

Man-pages をインストールするために以下を実行します。

```
make install
```

6.8.2. Man-pages の構成

インストールファイル: 様々な man ページ

概略説明

man ページ C 言語の関数、重要なデバイスファイル、重要な設定ファイルなどを説明します。

6.9. Glibc-2.12.1

Glibc パッケージは主要な C ライブラリを提供します。 このライブラリは基本的な処理ルーチンを含むもので、メモリ割り当て、ディレクトリ走査、ファイルのオープン・クローズや入出力、文字列操作、パターンマッチング、算術処理、等々があります。

概算ビルド時間: 16.9 SBU
必要ディスク容量: 637 MB

6.9.1. Glibc のインストール

注記

LFS が取り扱っていないパッケージの中には GNU libiconv の導入を推奨しているものがあります。 これは文字データのエンコーディングを変換する機能を持ちます。 プロジェクトのホームページ（http://www.gnu.org/software/libiconv/ ）には以下のような説明があります。 「このライブラリは iconv() 関数を提供します。 この関数を持たないシステムや、Unicode を取り扱うことができないシステムにて、この関数を利用することができます。 」 Glibc が iconv() 関数を用意しており Unicode の変換を実現しているため LFS では libiconv は用いないことにします。

Glibc は自らによってビルドされるものであり、そうして完全な形でインストールされます。 ただしコンパイラのスペックファイルやリンカは、まだ /tools ディレクトリを示したままです。 スペックファイルやリンカを再調整するのは Glibc をインストールした後になります。 これは Glibc の autoconf テストが失敗するためであり、最終的にきれいなビルド結果を得るという目的が達成できないためです。

make install を実行すると test-installation.pl というスクリプトが実行され、新たに作り出された Glibc に対しての簡単な健全性テストが実施されます。 しかしこの時点ではツールチェーンが /tools ディレクトリを指し示しているため、誤った Glibc を対象としてテストが実施されてしまいます。 このスクリプトのテスト対象が、これから作り出す Glibc となるように以下を実行します。

```
DL=$(readelf -l /bin/sh | sed -n 's@.*interpret.*/tools\(.*\)]$@\1@p')
sed -i "s|libs -o|libs -L/usr/lib -Wl,-dynamic-linker=$DL -o|" \
        scripts/test-installation.pl
unset DL
```

ldd シェルスクリプトは Bash が定める文法書式により構成されています。 デフォルトで記述されているインタープリタを /bin/bash に変更します。 BLFS ブックの シェル（Shells）で説明しているように、別の /bin/sh がインストールされている場合もあるからです。

```
sed -i 's|@BASH@|/bin/bash|' elf/ldd.bash.in
```

Glibc が GCC-4.5.1 に対してビルドできなくなるバグを修正します。

```
patch -Np1 -i ../glibc-2.12.1-gcc_fix-1.patch
```

Make の バージョンが 3.81 より最新では Glibc がビルドできないバグを修正します。

```
patch -Np1 -i ../glibc-2.12.1-makefile_fix-1.patch
```

Glibc のドキュメントではソースディレクトリ以外の専用のビルドディレクトリを作成することが推奨されています。

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

第5章と同じように x86 マシンにおいては CFLAGS に対してコンパイラフラグの追加が必要です。 ライブラリ構築においても gcc コンパイラに対して最適化フラグをセットすることで、コンパイル時間を向上（-pipe）させ、パッケージのパフォーマンスも向上（-O3）させます。

```
case 'uname -m' in
  i?86) echo "CFLAGS += -march=i486 -mtune=native -O3 -pipe" > configparms ;;
esac
```

Glibc をコンパイルするための準備をします。

```
../glibc-2.12.1/configure --prefix=/usr \
    --disable-profile --enable-add-ons \
    --enable-kernel=2.6.22.5 --libexecdir=/usr/lib/glibc
```

configure オプションの意味：

--libexecdir=/usr/lib/glibc
このオプションは pt_chown プログラムのインストール先を、デフォルトの /usr/libexec から /usr/lib/glibc に変更します。

パッケージをコンパイルします。

```
make
```

重要項目

本節における Glibc のテストスイートは極めて重要なものです。 したがってどのような場合であっても必ず実行してください。

テストを実施する前に、ソースディレクトリからビルドディレクトリにファイルを一つコピーします。 いくつかのテストが失敗してしまうことを回避するためです。 こうしておいてコンパイル結果をテストします。

```
cp -v ../glibc-2.12.1/iconvdata/gconv-modules iconvdata
make -k check 2>&1 | tee glibc-check-log
grep Error glibc-check-log
```

posix/annexc のテストはおそらく失敗します。 これは想定されていることであり無視することができます。 そもそも Glibc のテストスイートはホストシステムにある程度依存します。 発生しがちな問題を以下に示します。

- nptl/tst-clock2, nptl/tst-clock2, tst-attr3 の各テストは失敗することがあります。 失敗の理由は明確ではありません。 ただ処理速度が原因してそれらが発生すると思われます。
- math テストは、純正 Intel プロセッサや AMD プロセッサが最新のものではない場合に失敗することがあります。
- LFS パーティションを noatime オプションを用いてマウントしている場合 atime テストが失敗します。 2.4. 「新しいパーティションのマウント」 で説明しているように、LFS のビルド中は noatime オプションを使わないようにしてください。
- 旧式のハードウェアや性能の低いハードウェア、あるいは負荷の高いシステムにおいてテストを行うと、処理時間をオーバーしてタイムアウトが発生しテストが失敗します。 make check コマンドにて TIMEOUTFACTOR をセットするものに修正すれば、それらのエラーは回避できると報告されています。（例： TIMEOUTFACTOR=16 make -k check）

支障が出る話ではありませんが Glibc のインストール時には /etc/ld.so.conf ファイルが存在していないとして警告メッセージが出力されます。 これをなくすために以下を実行します。

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

パッケージをインストールします。

```
make install
```

システムを各種の言語に対応させるためのロケールは、今までのコマンドではインストールされませんが、テストスイートにおいてロケールは必要ではありません。 ただ将来的にはロケールがないことによって、重要なテストを逃してしまうかもしれません。

各ロケールは localedef プログラムを使ってインストールします。 例えば以下に示す一つめの localedef では、キャラクタセットには依存しないロケール定義 /usr/share/i18n/locales/cs_CZ とキャラクタマップ定義 /usr/share/i18n/charmaps/UTF-8.gz とを結合させて /usr/lib/locale/locale-archive ファイルにその情報を付け加えます。 以下のコマンドは、テストを成功させるために必要となる最低限のロケールをインストールするものです。

```
mkdir -pv /usr/lib/locale
localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i de_DE -f UTF-8 de_DE.UTF-8
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 en_US -f UTF-8 en_US.UTF-8
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
```

上に加えて、あなたの国、言語、キャラクタセットを定めるためのロケールをインストールしてください。

必要に応じて glibc-2.12.1/localedata/SUPPORTED に示されるすべてのロケールを同時にインストールしてください。(そこには上のロケールも含め、すべてのロケールが列記されています。) 以下のコマンドによりそれを実現します。ただしこれには相当な処理時間を要します。

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

さらに必要なら glibc-2.12.1/localedata/SUPPORTED ファイルに示されていない特殊なロケールは localedef コマンドを使って生成・インストールを行ってください。

6.9.2. Glibc の設定

/etc/nsswitch.conf ファイルを作成しておく必要があります。 Glibc はこのファイルが無い場合や誤っている場合でもデフォルト設定を用いて動作しますが、ネットワーク環境下ではデフォルト設定であっても正しく動作しません。 またタイムゾーンの設定も必要になります。

以下のコマンドを実行して /etc/nsswitch.conf ファイルを生成します。

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

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

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

ローカルなタイムゾーンの設定を行うために、ここでは以下のスクリプトを実行します。

```
tzselect
```

地域情報を設定するためにいくつか尋ねられるのでそれに答えます。 このスクリプトはタイムゾーン名を表示します。(例えば America/Edmonton などです。) /usr/share/zoneinfo ディレクトリにはさらに Canada/Eastern や EST5EDT のようなタイムゾーンもあります。 これらはこのスクリプトでは認識されませんが、利用することは可能です。

以下のコマンドにより /etc/localtime ファイルを生成します。

```
cp -v --remove-destination /usr/share/zoneinfo/<xxx> \
    /etc/localtime
```

<xxx> の部分は設定するタイムゾーンの名前（例えば Canada/Eastern など）に置き換えてください。

cp オプションの意味：

--remove-destination
このオプションは既に存在するシンボリックリンクを削除します。 ここではシンボリックリンクを再生成するのではなく、ファイルのコピーを行います。 これは別パーティション内に /usr ディレクトリが存在するケースに対応するためです。 シングルユーザーモードでシステムを起動する際にはこのことが必要となります。

6.9.3. ダイナミックローダの設定

デフォルトにおいてダイナミックリンカ（/lib/ld-linux.so.2）は /lib ディレクトリと /usr/lib ディレクトリを検索しにいきます。 これに従って、他のプログラムが実行される際に必要となるダイナミックライブラリがリンクされます。 もし /lib や /usr/lib 以外のディレクトリにライブラリファイルがあるなら /etc/ld.so.conf ファイルに記述を追加して、ダイナミックローダがそれらを探し出せるようにしておくことが必要です。 追加のライブラリが配置されるディレクトリとしては /usr/local/lib ディレクトリと /opt/lib ディレクトリという二つがよく利用されます。 ダイナミックローダの検索パスとして、それらのディレクトリを追加します。

以下のコマンドを実行して /etc/ld.so.conf ファイルを新たに生成します。

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

/usr/local/lib
/opt/lib

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

6.9.4. Glibc の構成

インストールプログラム:	catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump, zic
インストールライブラリ:	ld.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libbsd-compat.a, libc.{a,so}, libc_nonshared.a, libcidn.so, libcrypt.{a,so}, libdl.{a,so}, libg.a, libieee.a, libm.{a,so}, libmcheck.a, libmemusage.so, libnsl.{a,so}, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.{a,so}, libpthread_nonshared.a, libresolv.{a,so}, librpcsvc.a, librt.{a,so}, libthread_db.so, libutil.{a,so}
インストールディレクトリ:	/usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/rpcsvc, /usr/include/sys, /usr/lib/gconv, /usr/lib/glibc, /usr/lib/locale, /usr/share/i18n, /usr/share/zoneinfo

概略説明

catchsegv	プログラムがセグメンテーションフォールトにより停止した時に、スタックトレースを生成するために利用します。
gencat	メッセージカタログを生成します。
getconf	ファイルシステムに固有の変数に設定された値を表示します。

getent	管理データベースから設定項目を取得します。
iconv	キャラクターセットを変換します。
iconvconfig	高速ロードができる iconv モジュール設定ファイルを生成します。
ldconfig	プログラム実行時におけるダイナミックリンカのリンクを設定します。
ldd	指定したプログラムまたは共有ライブラリが必要としている共有ライブラリを表示します。
lddlibc4	オブジェクトファイルを使って ldd コマンドを補助します。[訳註：意味不明]
locale	現在のロケールに対する様々な情報を表示します。
localedef	ロケールの設定をコンパイルします。
mtrace	メモリトレースファイル（memory trace file）を読み込んで解釈します。 そして可読可能な書式で出力します。
nscd	一般的なネームサービスへの変更要求のキャッシュを提供するデーモン。
pcprofiledump	PC プロファイリングによって生成される情報をダンプします。
pt_chown	grantpt コマンドのヘルパープログラム。 所有者、グループ、スレーブ擬似端末（slave pseudo terminal）へのアクセスパーミッションをそれぞれ設定します。
rpcgen	リモートプロシージャコール（Remote Procedure Call; RPC）を実装するための C 言語コードを生成します。
rpcinfo	RPC サーバーに対しての RPC コールを行います。
sln	スタティックなリンクを行う ln プログラム。
sprof	共有オブジェクトのプロファイリングデータを読み込んで表示します。
tzselect	ユーザーに対してシステムの設置地域を問合せ、対応するタイムゾーンの記述を表示します。
xtrace	プログラム内にて現在実行されている関数を表示することで、そのプログラムの実行状況を追跡します。
zdump	タイムゾーンをダンプします。
zic	タイムゾーンコンパイラ。
ld.so	共有ライブラリのためのヘルパープログラム。
libBrokenLocale	Glibc が内部で利用するもので、異常が発生しているプログラムを見つけ出します。（例えば Motif アプリケーションなど）詳しくは glibc-2.12.1/locale/broken_cur_max.c に書かれたコメントを参照してください。
libSegFault	セグメンテーションフォールトのシグナルハンドラ。 catchsegv が利用します。
libanl	非同期の名前解決（asynchronous name lookup）ライブラリ。
libbsd-compat	特定の BSD（Berkeley Software Distribution）プログラムを Linux 上で動作させるために必要な可搬ライブラリを提供します。
libc	主要な C ライブラリ。
libcidn	Glibc が内部的に利用するもので getaddrinfo() 関数によって国際化ドメイン名（internationalized domain names）を取り扱います。
libcrypt	暗号化ライブラリ。
libdl	ダイナミックリンクのインターフェースライブラリ。
libg	関数を全く含まないダミーのライブラリ。 かつては g++ のランタイムライブラリであったものです。
libieee	このモジュールをリンクすると、数学関数におけるエラー制御方法を IEEE（the Institute of Electrical and Electronic Engineers）が定義するものに従うようになります。 デフォルトは POSIX.1 エラー制御方法です。
libm	数学ライブラリ。
libmcheck	このライブラリにリンクした場合、メモリ割り当てのチェック機能を有効にします。
libmemusage	memusage コマンドが利用するもので、プログラムのメモリ使用に関する情報を収集します。
libnsl	ネットワークサービスライブラリ。
libnss	NSS（Name Service Switch）ライブラリ。 ホスト、ユーザー名、エイリアス、サービス、プロトコルなどの名前解決を行う関数を提供します。
libpcprofile	プロファイリングを行う関数を提供するもので、特定のソース行に費やされる CPU 時間を追跡するために利用します。

libpthread	POSIX スレッドライブラリ。
libresolv	インターネットドメインネームサーバーに対しての、パケットの生成・送信・解析を行う関数を提供します。
librpcsvc	様々な RPC サービスを実現する関数を提供します。
librt	POSIX.1b リアルタイム拡張（Realtime Extension）にて既定されている、インターフェースをほぼ網羅した関数を提供します。
libthread_db	マルチスレッドプログラム用のデバッガを構築するための有用な関数を提供します。
libutil	数多くの Unix ユーティリティにて利用される 「標準」 関数を提供します。

6.10. ツールチェーンの再調整

最終的な C ライブラリがこれまでに構築できました。 ここでツールチェーンの調整を再度行います。 これを行うことで、新たに生成したプログラムが新たに生成したライブラリにリンクされます。 この作業は 第5章 の冒頭にて行った「調整」作業と同様のことです。 ただし調整される方向が逆になります。 第5章 では、ホストシステムの /{,usr/}lib ディレクトリを新しく作った /tools/lib ディレクトリに仕向けていました。 今度は同じ /tools/lib ディレクトリを LFS の /{,usr/}lib ディレクトリに向けます。

まず /tools ディレクトリにあるリンカのバックアップをとっておき、第5章にて作成した調整済みリンカに置き換えます。 /tools/$(gcc-dumpmachine)/bin ディレクトリにあるリンカに対してのシンボリックリンクも正しく生成しておきます。

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

次に GCC スペックファイルを修正し、新しいダイナミックリンカを指し示すようにします。 単純に 「/tools」 という記述を取り除けば、ダイナミックリンカへの正しい参照となります。 またスペックファイルを修正することで GCC がヘッダファイル、および Glibc の起動ファイルを適切に探し出せるようになります。 以下の sed によりこれを実現します。

```
gcc -dumpspecs | sed -e 's@/tools@@g' \
    -e '/\*startfile_prefix_spec:/{n;s@.*@/usr/lib/ @}' \
    -e '/\*cpp:/{n;s@$@ -isystem /usr/include@}' > \
    'dirname $(gcc --print-libgcc-file-name)'/specs
```

スペックファイルの内容を実際に確認して、今変更した内容が正しく反映されていることを確認しておいてください。

この時点において、調整したツールチェーンの基本的な（コンパイルやリンクなどの）機能が正しく動作していることを確認する必要があります。 これを行うために以下の健全性検査を実行します。

```
echo 'main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドから出力される結果は以下のようになるはずです。（ダイナミックリンカの名前はプラットフォームによって違っているかもしれません。）

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

ダイナミックリンカのディレクトリは、今度は /lib となっているはずです。

ここで起動ファイルが正しく用いられていることを確認します。

```
grep -o '/usr/lib.*/crt[1in].*succeeded' dummy.log
```

問題なく動作した場合はエラーがなかったということで、上のコマンドの出力は以下のようになるはずです。

```
/usr/lib/crt1.o succeeded
/usr/lib/crti.o succeeded
/usr/lib/crtn.o succeeded
```

コンパイラが正しいヘッダファイルを読み取っているかどうかを検査します。

```
grep -B1 '^ /usr/include' dummy.log
```

上のコマンドは正常に終了すると、以下の出力を返します。

```
#include <...> search starts here:
 /usr/include
```

次に、新たなリンカが正しいパスを検索して用いられているかどうかを検査します。

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドの出力は以下のようになるはずです。（作業するプラットフォームに応じて「三つの組（target triplet)」の表記は異なります。）

```
SEARCH_DIR("/tools/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib");
```

次に libc が正しく用いられていることを確認します。

```
grep "/lib.*/libc.so.6 " dummy.log
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドの出力は以下のようになるはずです。（64 ビットマシンであれば lib64 ディレクトリとなるはずです。）

```
attempt to open /lib/libc.so.6 succeeded
```

最後に GCC が正しくダイナミックリンカを用いているかを確認します。

```
grep found dummy.log
```

問題なく動作した場合はエラーがなかったということで、上のコマンドの出力は以下のようになるはずです。（ダイナミックリンカの名前はプラットフォームによって違っているかもしれません。 また 64 ビットマシンであれば lib64 ディレクトリとなるはずです。）

```
found ld-linux.so.2 at /lib/ld-linux.so.2
```

出力結果が上と異なっていたり、出力が全く得られなかったりした場合は、何かが根本的に間違っているということです。 どこに問題があるのか調査・再試行を行って解消してください。 最もありがちな理由は、スペックファイルの修正を誤っていることです。 問題を残したままこの先には進まないでください。

すべてが正しく動作したら、テストに用いたファイルを削除します。

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

6.11. Zlib-1.2.5

Zlib パッケージは、各種プログラムから呼び出される、圧縮、伸張（解凍）を行う関数を提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.8 MB

6.11.1. Zlib のインストール

まずパッケージ内のヘッダーファイルのタイポを修正します。

```
sed -i 's/ifdef _LARGEFILE64_SOURCE/ifndef _LARGEFILE64_SOURCE/' zlib.h
```

Zlib を生成する準備をします。

```
CFLAGS='-mstackrealign -fPIC -O3' ./configure --prefix=/usr
```

configure における環境変数の意味：

CFLAGS='-mstackrealign -fPIC -O3'
CFLAGS を設定することで、デフォルトの最適化オプション指定を上書きします。 実行時エラーが発生する場合があるため、それを修正するものです。 -mstackrealign オプションは Intel アーキテクチャではないシステムではビルドに失敗するかもしれません。

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

共有ライブラリは /lib に移す必要があります。 またそれに合わせて /usr/lib にある .so ファイルを再生成する必要があります。

```
mv -v /usr/lib/libz.so.* /lib
ln -sfv ../../lib/libz.so.1.2.5 /usr/lib/libz.so
```

6.11.2. Zlib の構成

インストールライブラリ: libz.{a,so}

概略説明

libz 各種プログラムから呼び出される、圧縮、伸張（解凍）を行う関数を提供します。

6.12. Binutils-2.20.1

Binutils パッケージは、リンカやアセンブラなどのようにオブジェクトファイルを取り扱うツール類を提供します。

概算ビルド時間: 2.1 SBU
必要ディスク容量: 222 MB

6.12.1. Binutils のインストール

PTY が chroot 環境内にて正しく作動しているかどうかを確認するために、以下の簡単なテストを実行します。

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

上のコマンドは以下を出力するはずです。

```
spawn ls
```

上のような出力ではなく、以下のような出力メッセージが含まれていたら、PTY の動作が適切に構築できていないことを示しています。 Binutils や GCC のテストスイートを実行する前に、この症状は解消しておく必要があります。

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

standards.info ファイルの日付が古いため、インストールしないことにします。 より新しいものが Autoconf の作業を通じてインストールされます。

```
rm -fv etc/standards.info
sed -i.bak '/^INFO/s/standards.info //' etc/Makefile.in
```

Binutils のドキュメントによると Binutils のビルドにあたっては、ソースディレクトリ以外の専用のビルドディレクトリを作成することが推奨されています。

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

Binutils をコンパイルするための準備をします。

```
../binutils-2.20.1/configure --prefix=/usr \
    --enable-shared
```

パッケージをコンパイルします。

```
make tooldir=/usr
```

make パラメータの意味：

tooldir=/usr
通常 tooldir（実行ファイルが最終的に配置されるディレクトリ）は $(exec_prefix)/$(target_alias) に設定されています。 x86_64 マシンでは /usr/x86_64-unknown-linux-gnu となります。 LFS は自分で設定を定めていくシステムですから /usr ディレクトリ配下に CPU ターゲットを特定するディレクトリを設ける必要がありません。 $(exec_prefix)/$(target_alias) というディレクトリ構成は、クロスコンパイル環境において必要となるものです。（例えばパッケージをコンパイルするマシンが Intel であり、そこから PowerPC マシン用の実行コードを生成するような場合です。）

重要項目

本節における Binutils のテストスイートは極めて重要なものです。 したがってどのような場合であっても必ず実行してください。

コンパイル結果をテストします。

```
make check
```

パッケージをインストールします。

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

libiberty ヘッダファイルをインストールします。 他のパッケージがこれを必要としている場合があるためです。

```
cp -v ../binutils-2.20.1/include/libiberty.h /usr/include
```

6.12.2. Binutils の構成

インストールプログラム:	addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings, strip
インストールライブラリ:	libiberty.a, libbfd.{a,so}, libopcodes.{a,so}
インストールディレクトリ:	/usr/lib/ldscripts

概略説明

addr2line	指定された実行モジュール名とアドレスに基づいて、プログラム内のアドレスをファイル名と行番号に変換します。 これは実行モジュール内のデバッグ情報を利用します。 特定のアドレスがどのソースファイルと行番号に該当するかを確認するものです。
ar	アーカイブの生成、修正、抽出を行います。
as	gcc の出力結果をアセンブルして、オブジェクトファイルとして生成するアセンブラ。
c++filt	リンカから呼び出されるもので C++ と Java のシンボルを複合（demangle）し、オーバーロード関数が破壊されることを回避します。
gprof	コールグラフ（call graph）のプロファイルデータを表示します。
ld	複数のオブジェクトファイルやアーカイブファイルから、一つのファイルを生成するリンカ。 データの再配置やシンボル参照情報の結合を行います。
nm	指定されたオブジェクトファイル内のシンボル情報を一覧表示します。
objcopy	オブジェクトファイルの変換を行います。
objdump	指定されたオブジェクトファイルの各種情報を表示します。 様々なオプションを用いることで特定の情報表示が可能です。 表示される情報は、コンパイル関連ツールを開発する際に有用なものです。
ranlib	アーカイブの内容を索引として生成し、それをアーカイブに保存します。 索引は、アーカイブのメンバによって定義されるすべてのシンボルの一覧により構成されます。 アーカイブのメンバとは再配置可能なオブジェクトファイルのことです。
readelf	ELF フォーマットのバイナリファイルの情報を表示します。
size	指定されたオブジェクトファイルのセクションサイズと合計サイズを一覧表示します。
strings	指定されたファイルに対して、印字可能な文字の並びを出力します。 文字は所定の長さ（デフォルトでは 4文字）以上のものが対象となります。 オブジェクトファイルの場合デフォルトでは、初期化セクションとロードされるセクションからのみ文字列を抽出し出力します。 これ以外の種類のファイルの場合は、ファイル全体が走査されます。
strip	オブジェクトファイルからデバッグシンボルを取り除きます。
libiberty	以下に示すような数多くの GNU プログラムが利用する処理ルーチンを提供します。 getopt、obstack、 strerror、 strtol、 strtoul
libbfd	バイナリファイルディスクリプタ（Binary File Descriptor）ライブラリ。
libopcodes	opcodes（オペレーションコード; プロセッサ命令を 「認識可能なテキスト」 として表現したもの）を取り扱うライブラリ。 このライブラリは objdump などのように、ビルド作業にて利用するユーティリティプログラムが利用しています。

6.13. GMP-5.0.1

GMP パッケージは数値演算ライブラリを提供します。 このライブラリには任意精度演算（arbitrary precision arithmetic）を行う有用な関数が含まれます。

概算ビルド時間: 1.7 SBU
必要ディスク容量: 39 MB

6.13.1. GMP のインストール

注記

32 ビット x86 CPU にて環境構築する際に、 64 ビットコードを扱う CPU 環境であって かつ CFLAGS を指定していると、 本パッケージの configure スクリプトは 64 ビット用の処理を行い失敗します。 これを回避するには、以下のように処理してください。

```
ABI=32 ./configure ...
```

GMP をコンパイルするための準備をします。

```
./configure --prefix=/usr --enable-cxx --enable-mpbsd
```

configure オプションの意味：

--enable-cxx
C++ サポートを有効にします。

--enable-mpbsd
Berkeley MP に対する互換ライブラリをビルドします。

パッケージをコンパイルします。

```
make
```

重要項目

本節における GMP のテストスイートは極めて重要なものです。 したがってどのような場合であっても必ず実行してください。

テストを実行します。

```
make check 2>&1 | tee gmp-check-log
```

162個のテストが完了することを確認してください。 テスト結果は以下のコマンドにより確認することができます。

```
awk '/tests passed/{total+=$2} ; END{print total}' gmp-check-log
```

パッケージをインストールします。

```
make install
```

必要ならドキュメントをインストールします。

```
mkdir -v /usr/share/doc/gmp-5.0.1
cp    -v doc/{isa_abi_headache,configuration} doc/*.html \
         /usr/share/doc/gmp-5.0.1
```

6.13.2. GMP の構成

インストールライブラリ: libgmp.{a,so}, libgmpxx.{a,so}, libmp.{a,so}
インストールディレクトリ: /usr/share/doc/gmp-5.0.1

概略説明

libgmp 精度演算関数（precision math functions）を提供します。

libgmpxx　C++ 用の精度演算関数を提供します。
libmp　Berkley MP 演算関数を提供します。

6.14. MPFR-3.0.0

MPFR パッケージは倍精度演算（multiple precision）の関数を提供します。

概算ビルド時間:　　1.1 SBU
必要ディスク容量:　　27.1 MB

6.14.1. MPFR のインストール

MPFR をコンパイルするための準備をします。

```
./configure --prefix=/usr --enable-thread-safe \
  --docdir=/usr/share/doc/mpfr-3.0.0
```

パッケージをコンパイルします。

```
make
```

重要項目

本節における MPFR のテストスイートは極めて重要なものです。 したがってどのような場合であっても必ず実行してください。

すべてのテストが正常に完了していることを確認してください。

```
make check
```

パッケージをインストールします。

```
make install
```

ドキュメントをインストールします。

```
make html
make install-html
```

6.14.2. MPFR の構成

インストールライブラリ:　　libmpfr.{a,so}
インストールディレクトリ:　　/usr/share/doc/mpfr-3.0.0

概略説明

libmpfr　　倍精度演算の関数を提供します。

6.15. MPC-0.8.2

MPC パッケージは複素数演算を可能とするライブラリを提供するものです。 高い精度と適切な丸め（rounding）を実現します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 10.5 MB

6.15.1. MPC のインストール

MPC をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.15.2. MPC の構成

インストールライブラリ: libmpc.{a,so}

概略説明

libmpc 複素数による演算関数を提供します。

6.16. GCC-4.5.1

GCC パッケージは C コンパイラや C++ コンパイラなどの GNU コンパイラコレクションを提供します。

概算ビルド時間: 44 SBU
必要ディスク容量: 1.1 GB

6.16.1. GCC のインストール

sed による置換を行って libiberty.a をインストールしないようにします。 libiberty.a は Binutils が提供するものを利用することにします。

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

5.10. 「GCC-4.5.1 - 2回め」 にて行ったように sed を使って以下のようにコンパイラフラグ -fomit-frame-pointer を強制的に指定し、一貫したコンパイルを実現します。

```
case 'uname -m' in
  i?86) sed -i 's/^T_CFLAGS =$/& -fomit-frame-pointer/' \
        gcc/Makefile.in ;;
esac
```

fixincludes スクリプトは、たまにエラーを引き起こし、それまでにインストールされているシステムヘッダーファイルを修正してしまうことがあります。 ここまでにインストールしてきたヘッダファイルは修正する必要がないので、以下のコマンドを実行して fixincludes スクリプトを実行しないようにします。

```
sed -i 's@\./fixinc\.sh@-c true@' gcc/Makefile.in
```

GCC のドキュメントによると GCC のビルドにあたっては、ソースディレクトリ以外の専用のビルドディレクトリを作成することが推奨されています。

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

GCC をコンパイルするための準備をします。

```
../gcc-4.5.1/configure --prefix=/usr \
    --libexecdir=/usr/lib --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-clocale=gnu --enable-languages=c,c++ \
    --disable-multilib --disable-bootstrap --with-system-zlib
```

他のプログラミング言語は、また別の依存パッケージなどを要しますが、現時点では準備できていません。 GCC がサポートする他のプログラム言語の構築方法については BLFS ブックの説明を参照してください。

Configure オプションの意味:

--with-system-zlib
: このオプションはシステムに既にインストールされている Zlib ライブラリをリンクすることを指示するものであり、内部にて作成されるライブラリを用いないようにします。

パッケージをコンパイルします。

```
make
```

重要項目

本節における GCC のテストスイートは極めて重要なものです。 したがってどのような場合であっても必ず実行してください。

GCC テストスイートの中で、スタックを使い果たすものがあります。 そこでテスト実施にあたり、スタックサイズを増やします。

```
ulimit -s 16384
```

コンパイル結果をテストします。 エラーが発生しても停止しないようにします。

```
make -k check
```

テスト結果を確認するために以下を実行します。

```
../gcc-4.5.1/contrib/test_summary
```

テスト結果の概略のみ確認したい場合は、 出力結果をパイプ出力して grep -A7 Summ を実行してください。

テスト結果については http://www.linuxfromscratch.org/lfs/build-logs/6.7/ と http://gcc.gnu.org/ml/gcc-testresults/ にある情報と比較することができます。

テストに失敗することがありますが、これを回避することはできません。 GCC の開発者はこの問題を認識していますが、まだ解決していない状況です。 特に libmudflap のテストは大いに問題があり GCC のバグとして知られています。（http://gcc.gnu.org/bugzilla/show_bug.cgi?id=20003 ）この URL に示されている結果と大きく異なっていなかったら、 問題はありませんので先に進んでください。

パッケージをインストールします。

```
make install
```

パッケージの中には C プリプロセッサが /lib ディレクトリにあることを前提にしているものがあります。 そのようなものに対応するため、以下のシンボリックリンクを作成します。

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

パッケージの多くは C コンパイラとして cc を呼び出しています。 これに対応するため、以下のシンボリックリンクを作成します。

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

最終的なツールチェーンが出来上がりました。 ここで再びコンパイルとリンクが正しく動作することを確認することが必要です。 そこで本節の初めの方で実施した健全性テストをここでも実施します。

```
echo 'main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドから出力される結果は以下のようになるはずです。（ダイナミックリンカの名前はプラットフォームによって違っているかもしれません。）

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

ここで起動ファイルが正しく用いられていることを確認します。

```
grep -o '/usr/lib.*/crt[1in].*succeeded' dummy.log
```

問題なく動作した場合はエラーがなかったということで、上のコマンドの出力は以下のようになるはずです。

```
/usr/lib/gcc/i686-pc-linux-gnu/4.5.1/../../../crt1.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.5.1/../../../crti.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.5.1/../../../crtn.o succeeded
```

作業しているマシンアーキテクチャによっては、上の結果が微妙に異なるかもしれません。 その違いは、たいていは /usr/lib/gcc の次のディレクトリ名にあります。 作業マシンが 64 ビット機である場合、ディレクトリ名の後ろの方に lib64 という名が出てくることになります。 ここで確認すべき重要なポイントは gcc が /usr/lib ディレクトリ配下に三つのファイル crt*.o を見つけ出しているかどうかです。

コンパイラが正しいヘッダファイルを読み取っているかどうかを検査します。

```
grep -B4 '^ /usr/include' dummy.log
```

上のコマンドは正常に終了すると、以下の出力を返します。

```
#include <...> search starts here:
 /usr/local/include
 /usr/lib/gcc/i686-pc-linux-gnu/4.5.1/include
 /usr/lib/gcc/i686-pc-linux-gnu/4.5.1/include-fixed
 /usr/include
```

もう一度触れておきますが、プラットフォームの 「三つの組（target triplet)」 の次にくるディレクトリ名は CPU アーキテクチャにより異なる点に注意してください。

注記

GCC のバージョン 4.3.0 では limits.h ファイルを無条件に include-fixed ディレクトリにインストールします。 したがってそのディレクトリは存在していなければなりません。

次に、新たなリンカが正しいパスを検索して用いられているかどうかを検査します。

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドの出力は以下のようになるはずです。（作業するプラットフォームに応じて「三つの組（target triplet)」の表記は異なります。）

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

64 ビットシステムではさらにいくつかのディレクトリが出力されます。 例えば x86_64 マシンであれば、その出力は以下のようになります。

```
SEARCH_DIR("/usr/x86_64-unknown-linux-gnu/lib64")
SEARCH_DIR("/usr/local/lib64")
SEARCH_DIR("/lib64")
SEARCH_DIR("/usr/lib64")
SEARCH_DIR("/usr/x86_64-unknown-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

次に libc が正しく用いられていることを確認します。

```
grep "/lib.*/libc.so.6 " dummy.log
```

問題なく動作した場合はエラーがなかったということで、最後のコマンドの出力は以下のようになるはずです。（64 ビットマシンであれば lib64 ディレクトリとなるはずです。）

```
attempt to open /lib/libc.so.6 succeeded
```

最後に GCC が正しくダイナミックリンカを用いているかを確認します。

```
grep found dummy.log
```

問題なく動作した場合はエラーがなかったということで、上のコマンドの出力は以下のようになるはずです。（ダイナミックリンカの名前はプラットフォームによって違っているかもしれません。 また 64 ビットマシンであれば lib64 ディレクトリとなるはずです。）

```
found ld-linux.so.2 at /lib/ld-linux.so.2
```

出力結果が上と異なっていたり、出力が全く得られなかったりした場合は、何かが根本的に間違っているということです。 どこに問題があるのか調査・再試行を行って解消してください。 最もありがちな理由は、スペックファイルの修正を誤っていることです。 問題を残したままこの先には進まないでください。

すべてが正しく動作したら、テストに用いたファイルを削除します。

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

6.16.2. GCC の構成

インストールプログラム:	c++, cc（gcc へのリンク), cpp, g++, gcc, gccbug, gcov
インストールライブラリ:	libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libmudflap.{a,so}, libmudflapth.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so} libsupc++.a
インストールディレクトリ:	/usr/include/c++, /usr/lib/gcc, /usr/share/gcc-4.5.1

概略説明

c++	C++ コンパイラ
cc	C コンパイラ
cpp	C プリプロセッサ。 コンパイラがこれを利用して、ソース内に記述された #include、#define や同じようなステートメントを展開します。
g++	C++ コンパイラ
gcc	C コンパイラ
gccbug	有用なバグ報告の生成を手助けするスクリプト。
gcov	カバレッジテストツール。 プログラムを解析して、最適化が最も効果的となるのはどこかを特定します。
libgcc	gcc のランタイムサポートを提供します。
libgcov	GCC のプロファイリングを有効にした場合にこのライブラリがリンクされます。
libgomp	C/C++ や Fortran において、マルチプラットフォームでの共有メモリ並行プログラミング（multi-platform shared-memory parallel programming）を行うための、GNU による OpenMP API インプリメンテーションです。
libmudflap	GCC の配列境界チェック（bounds checking）機能をサポートするルーチンを提供します。
libssp	GCC のスタック破壊を防止する（stack-smashing protection）機能をサポートするルーチンを提供します。
libstdc++	標準 C++ ライブラリ
libsupc++	C++ プログラミング言語のためのサポートルーチンを提供します。

6.17. Sed-4.2.1

Sed パッケージはストリームエディタを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 8.3 MB

6.17.1. Sed のインストール

Sed をコンパイルするための準備をします。

```
./configure --prefix=/usr --bindir=/bin --htmldir=/usr/share/doc/sed-4.2.1
```

configure オプションの意味：

--htmldir
HTML ドキュメントをインストールするディレクトリを指定します。

パッケージをコンパイルします。

```
make
```

HTML ドキュメントを生成します。

```
make html
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

HTML ドキュメントをインストールします。

```
make -C doc install-html
```

6.17.2. Sed の構成

インストールプログラム: sed
インストールディレクトリ: /usr/share/doc/sed-4.2.1

概略説明

sed テキストファイルを一度の処理でフィルタリングし変換します。

6.18. Pkg-config-0.25

pkg-config パッケージは、他のパッケージ類の configure やメイクを行う際に、インクルードパスやライブラリパスの情報を伝えるためのツールを提供します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 11.5 MB

6.18.1. Pkg-config のインストール

最新の autoconf にて発生する問題を修正します。

```
sed -i -e 's/XT])dnl/XT])[]dnl/' \
       -e 's/\.])dnl/\.])[]dnl/' pkg.m4
```

Pkg-config をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.18.2. Pkg-config の構成

インストールプログラム: pkg-config

概略説明

pkg-config 指定されたライブラリやパッケージの全般的な情報を返します。

6.19. Ncurses-5.7

Ncurses パッケージは、端末に依存しない、文字ベースのスクリーン制御を行うライブラリを提供します。

概算ビルド時間: 0.8 SBU
必要ディスク容量: 35 MB

6.19.1. Ncurses のインストール

Ncurses をコンパイルするための準備をします。

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

configure オプションの意味：

--enable-widec
このオプションは通常のライブラリ（libncurses.so.5.7）ではなくワイド文字対応のライブラリ（libncursesw.so.5.7）をビルドすることを指示します。 ワイド文字対応のライブラリは、マルチバイトロケールと従来の 8ビットロケールの双方に対して利用可能です。 通常のライブラリでは 8ビットロケールに対してしか動作しません。 ワイド文字対応と通常のものとでは、ソース互換があるもののバイナリ互換がありません。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありますが、パッケージをインストールした後でないと実行できません。 テストスイートのためのファイル群はサブディレクトリ test/ 以下に残っています。 詳しいことはそのディレクトリ内にある README ファイルを参照してください。

パッケージをインストールします。

```
make install
```

共有ライブラリを /lib ディレクトリに移動します。 これらはここにあるべきものです。

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

ライブラリを移動させたので、シンボリックリンク先が存在しないことになります。 そこでリンクを再生成します。

```
ln -sfv ../../lib/libncursesw.so.5 /usr/lib/libncursesw.so
```

アプリケーションによっては、ワイド文字対応ではないライブラリをリンカが探し出すよう求めるものが多くあります。 そのようなアプリケーションに対しては、以下のようなシンボリックリンクやリンカスクリプトを作り出して、ワイド文字対応のライブラリにリンクさせるよう仕向けます。

```
for lib in ncurses form panel menu ; do \
    rm -vf /usr/lib/lib${lib}.so ; \
    echo "INPUT(-l${lib}w)" >/usr/lib/lib${lib}.so ; \
    ln -sfv lib${lib}w.a /usr/lib/lib${lib}.a ; \
done
ln -sfv libncurses++w.a /usr/lib/libncurses++.a
```

最後に古いアプリケーションにおいて、ビルド時に -lcurses を指定するものがあるため、これもビルド可能なものにします。

```
rm -vf /usr/lib/libcursesw.so
echo "INPUT(-lncursesw)" >/usr/lib/libcursesw.so
ln -sfv libncurses.so /usr/lib/libcurses.so
ln -sfv libncursesw.a /usr/lib/libcursesw.a
ln -sfv libncurses.a /usr/lib/libcurses.a
```

必要なら Ncurses のドキュメントをインストールします。

```
mkdir -v       /usr/share/doc/ncurses-5.7
cp -v -R doc/* /usr/share/doc/ncurses-5.7
```

注記

ここまでの作業手順では、ワイド文字対応ではない Ncurses ライブラリは生成しませんでした。 ソースからコンパイルして構築するパッケージなら、実行時にそのようなライブラリにリンクするものはないからです。バイナリコードしかないアプリケーションを取り扱う場合、あるいは LSB 対応を要する場合で、それがワイド文字対応ではないライブラリを必要とするなら、以下のコマンドによりそのようなライブラリを生成してください。

```
make distclean
./configure --prefix=/usr --with-shared --without-normal \
  --without-debug --without-cxx-binding
make sources libs
cp -av lib/lib*.so.5* /usr/lib
```

6.19.2. Ncurses の構成

インストールプログラム:	captoinfo (tic へのリンク), clear, infocmp, infotocap (tic へのリンク), ncursesw5-config, reset (tset へのリンク), tic, toe, tput, tset
インストールライブラリ:	libcursesw.{a,so} (libncursesw.{a,so} へのシンボリックリンクおよびリンカスクリプト), libformw.{a,so}, libmenuw.{a,so}, libncurses++w.a, libncursesw.{a,so}, libpanelw.{a,so} これらに加えてワイド文字対応ではない通常のライブラリで、その名称から "w" を取り除いたもの。
インストールディレクトリ:	/usr/share/tabset, /usr/share/terminfo

概略説明

captoinfo	termcap の記述を terminfo の記述に変換します。
clear	画面消去が可能ならこれを行います。
infocmp	terminfo の記述どうしを比較したり出力したりします。
infotocap	terminfo の記述を termcap の記述に変換します。
ncursesw5-config	ncurses の設定情報を提供します。
reset	端末をデフォルト設定に初期化します。
tic	terminfo の定義項目に対するコンパイラです。 これはソース形式の terminfo ファイルをバイナリ形式に変換し、ncurses ライブラリ内の処理ルーチンが利用できるようにします。terminfo ファイルは特定端末の特性に関する情報が記述されるものです。
toe	利用可能なすべての端末タイプを一覧表示します。 そこでは端末名と簡単な説明を示します。
tput	端末に依存する機能設定をシェルが利用できるようにします。 また端末のリセットや初期化、あるいは長い端末名称の表示も行います。
tset	端末の初期化に利用します。
libcurses	libncurses へのリンク。
libncurses	様々な方法により端末画面上に文字列を表示するための関数を提供します。 これらの関数を用いた具体例として、カーネルの make menuconfig の実行によって表示されるメニューがあります。
libform	フォームを実装するための関数を提供します。
libmenu	メニューを実装するための関数を提供します。
libpanel	パネルを実装するための関数を提供します。

6.20. Util-linux-ng-2.18

Util-linux パッケージは様々なユーティリティプログラムを提供します。 ファイルシステム、コンソール、パーティション、カーネルメッセージなどを取り扱うユーティリティです。

概算ビルド時間: 0.6 SBU
必要ディスク容量: 49 MB

6.20.1. FHS コンプライアンス情報

FHS では adjtime ファイルの配置場所として /etc ディレクトリではなく /var/lib/hwclock ディレクトリを推奨しています。hwclock プログラムを FHS 準拠とするために以下を実行します。

```
sed -e 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
    -i $(grep -rl '/etc/adjtime' .)
mkdir -pv /var/lib/hwclock
```

6.20.2. Util-linux-ng のインストール

```
./configure --enable-arch --enable-partx --enable-write
```

configure オプションの意味：

--enable-arch
arch プログラムをビルドします。

--enable-partx
addpart、delpart、partx プログラムをビルドします。

--enable-write
write プログラムをビルドします。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

6.20.3. Util-linux-ng の構成

インストールプログラム: addpart, agetty, arch, blkid, blockdev, cal, cfdisk, chkdupexe, chrt, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, delpart, dmesg, fallocate, fdformat, fdisk, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, getopt, hexdump, hwclock, i386, ionice, ipcmk, ipcrm, ipcs, isosize, ldattach, line, linux32, linux64, logger, look, losetup, lscpu, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, partx, pg, pivot_root, readprofile, rename, renice, rev, rtcwake, script, scriptreplay, setarch, setsid, setterm, sfdisk, swaplabel, swapoff (swapon へのリンク), swapon, switch_root, tailf, taskset, tunelp, ul, umount, unshare, uuidd, uuidgen, wall, whereis, wipefs, write
インストールライブラリ: libblkid.{a,so}, libmount.{a,so}, libuuid.{a,so}
インストールディレクトリ: /usr/share/getopt, /var/lib/hwclock

概略説明

addpart Linux カーネルに対して新しいパーティションの情報を通知します。
agetty tty ポートを開いてログイン名の入力を受け付けます。 そして login プログラムを起動します。
arch マシンアーキテクチャを表示します。
blkid ブロックデバイスの属性を見つけて表示するためのコマンドラインユーティリティー。

blockdev	コマンドラインからブロックデバイスの ioctl の呼び出しを行います。
cal	簡単なカレンダーを表示します。
cfdisk	指定されたデバイスのパーティションテーブルを操作します。
chkdupexe	重複している実行モジュールを探します。
chrt	リアルタイムプロセスの属性を操作します。
col	逆改行（resverse line feeds）を取り除きます。
colcrt	性能が不十分な端末のために nroff の出力結果から重ね書き（overstriking）や半改行（half-lines）を取り除きます。
colrm	指定されたカラムを取り除きます。
column	指定されたファイルの内容を複数カラムに整形します。
ctrlaltdel	ハードリセットまたはソフトリセットを行うために Ctrl+Alt+Del キー押下時の機能を設定します。
cytune	Cyclades カード用のシリアルラインドライバのパラメータを設定します。
ddate	ディスコルディア暦（Discordian）の日付を表示します。 または指定されたグレゴリオ暦（Gregorian）の日付をディスコルディア暦の日付に変換します。
delpart	Linux カーネルに対してパーティションが削除されているかどうかを確認します。
dmesg	カーネルのブートメッセージをダンプします。
fallocate	ファイルのための領域を事前割り当てします。
fdformat	フロッピーディスクの低レベル（low-level）フォーマットを行います。
fdisk	指定されたデバイスのパーティションテーブルを操作します。
findfs	ファイルシステムに対するラベルまたは UUID（Universally Unique Identifier）を使ってファイルシステムを検索します。
findmnt	libmount ライブラリに対するコマンドラインインターフェース。 mountinfo, fstab, mtab の各ファイルに対しての処理を行います。
flock	ファイルロックを取得して、ロックしたままコマンドを実行します。
fsck	ファイルシステムのチェックを行い、必要に応じて修復を行います。
fsck.cramfs	指定されたデバイス上の Cramfs ファイルシステムに対して一貫性検査（consistency check）を行います。
fsck.minix	指定されたデバイス上の Minix ファイルシステムに対して一貫性検査（consistency check）を行います。
fsfreeze	カーネルドライバ制御における FIFREEZE/FITHAW ioctl に対する単純なラッパープログラム。
getopt	指定されたコマンドラインのオプション引数を解析します。
hexdump	指定されたファイルを 16進数書式または他の指定された書式でダンプします。
hwclock	システムのハードウェアクロックを読み取ったり設定したりします。 このハードウェアクロックはリアルタイムクリック（Real-Time Clock; RTC）または BIOS（Basic Input-Output System）クロックとも呼ばれます。
i386	setarch へのシンボリックリンク。
ionice	プログラムに対する I/O スケジュールクラスとスケジュール優先度を取得または設定します。
ipcmk	様々な IPC リソースを生成します。
ipcrm	指定された IPC（Inter-Process Communication）リソースを削除します。
ipcs	IPC のステータス情報を提供します。
isosize	iso9660 ファイルシステムのサイズを表示します。
ldattach	シリアル回線（serial line）に対して回線規則（line discipline）を割り当てます。
line	単一行をコピーします。
linux32	setarch へのシンボリックリンク。
linux64	setarch へのシンボリックリンク。
logger	指定したメッセージをシステムログに出力します。
look	指定された文字列で始まる行を表示します。
losetup	ループデバイス（loop device）の設定と制御を行います。

lscpu	CPU アーキテクチャの情報を表示します。
mcookie	xauth のためのマジッククッキー（128ビットのランダムな16進数値）を生成します。
mkfs	デバイス上にファイルシステムを構築します。（通常はハードディスクパーティションに対して行います。）
mkfs.bfs	SCO (Santa Cruz Operations) の bfs ファイルシステムを生成します。
mkfs.cramfs	cramfs ファイルシステムを生成します。
mkfs.minix	Minix ファイルシステムを生成します。
mkswap	指定されたデバイスまたはファイルをスワップ領域として初期化します。
more	テキストを一度に一画面分だけ表示するフィルタプログラム。
mount	ファイルシステムツリー内の特定のディレクトリを、指定されたデバイス上のファイルシステムに割り当てます。
namei	指定されたパスに存在するシンボリックリンクを表示します。
partx	カーネルに対して、ディスク上にパーティションが存在するか、何番が存在するかを伝えます。
pg	テキストファイルを一度に一画面分表示します。
pivot_root	指定されたファイルシステムを、現在のプロセスに対する新しいルートファイルシステムにします。
readprofile	カーネルのプロファイリング情報を読み込みます。
rename	指定されたファイルの名称を変更します。
renice	実行中のプロセスの優先度を変更します。
rev	指定されたファイル内の行の並びを入れ替えます。
rtcwake	指定された起動時刻までの間、システムをスリープ状態とするモードを指定します。
script	端末セッション上での出力結果の写し（typescript）を生成します。
scriptreplay	タイミング情報（timing information）を利用して、出力結果の写し（typescript）を再生します。
setarch	新しいプログラム環境にて、表示されるアーキテクチャを変更します。 また設定フラグ（personality flag）の設定も行います。
setsid	新しいセッションで指定されたプログラムを実行します。
setterm	端末の属性を設定します。
sfdisk	ディスクパーティションテーブルを操作します。
swaplabel	スワップエリアの UUID とラベルを変更します。
swapoff	ページングまたはスワッピングに利用しているデバイスまたはファイルを無効にします。
swapon	ページングまたはスワッピングに利用しているデバイスまたはファイルを有効にします。 また現在利用されているデバイスまたはファイルを一覧表示します。
switch_root	別のファイルシステムを、マウントツリーのルートとして変更します。
tailf	ログファイルの更新を監視します。 ログファイルの最終の10行が表示され、ログファイルに新たに書き込みが行われると表示更新します。
taskset	プロセスの CPU 親和性（affinity）を表示または設定します。
tunelp	ラインプリンタのパラメータを設定します。
ul	使用中の端末にて、アンダースコア文字を、エスケープシーケンスを用いた下線文字に変換するためのフィルタ。
umount	システムのファイルツリーからファイルシステムを切断します。
unshare	上位の名前空間とは異なる名前空間にてプログラムを実行します。
uuidd	UUID ライブラリから利用されるデーモン。 時刻情報に基づく UUID を、安全にそして一意性を確保して生成します。
uuidgen	新しい UUID を生成します。 生成される UUID は当然、他に生成されている UUID とは異なり、自他システムでも過去現在にわたってもユニークなものです。
wall	ファイルの内容、あるいはデフォルトでは標準入力から入力された内容を、現在ログインしている全ユーザーの端末上に表示します。
whereis	指定されたコマンドの実行モジュール、ソース、man ページの場所を表示します。
wipefs	ファイルシステムのシグニチャをデバイスから消去します。

write 指定されたユーザーに対してメッセージを送信します。 ただし　そのユーザーがメッセージ受信が可能である場合に限ります。

libblkid デバイスの識別やトークンの抽出を行う処理ルーチンを提供します。

libuuid ローカルシステム内だけに限らずアクセスされるオブジェクトに対して、一意性が保証された識別子を生成する処理ルーチンを提供します。

6.21. E2fsprogs-1.41.12

E2fsprogs パッケージは ext2 ファイルシステムを扱うユーティリティを提供します。これは同時に ext3、ext4 ジャーナリングファイルシステムもサポートします。

概算ビルド時間: 0.5 SBU
必要ディスク容量: 45 MB

6.21.1. E2fsprogs のインストール

E2fsprogs パッケージは、ソースディレクトリ内にサブディレクトリを作ってビルドすることが推奨されています。

```
mkdir -v build
cd build
```

E2fsprogs をコンパイルするための準備をします。

```
../configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs --disable-libblkid --disable-libuuid \
    --disable-uuidd --disable-fsck
```

configure オプションの意味：

--with-root-prefix=""
e2fsck などのプログラムは、極めて重要なものです。 例えば /usr ディレクトリがマウントされていない時であっても、そういったプログラムは動作しなければなりません。 それらは /lib ディレクトリや /sbin ディレクトリに置かれるべきものです。 もしこのオプションの指定がなかったら、プログラムが /usr ディレクトリにインストールされてしまいます。

--enable-elf-shlibs
このオプションは、本パッケージ内のプログラムが利用する共有ライブラリを生成します。

--disable-*
このオプションは libuuid ライブラリ、 libblkid ライブラリ、 uuidd デーモン、 fsck ラッパーをいずれもビルドせずインストールしないようにします。 これらは Util-Linux-NG パッケージによって既にインストールされています。

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

E2fsprogs にて行われるテストの中には 256 MB のメモリ割り当てを行うものがあります。 この容量を確保できるだけの RAM がない場合は、十分なスワップ領域を確保することが推奨されています。 スワップ領域の生成と有効化については 2.3. 「ファイルシステムの生成」 と 2.4. 「新しいパーティションのマウント」 を参照してください。

実行モジュール、ドキュメント、共有ライブラリをインストールします。

```
make install
```

スタティックライブラリとヘッダファイルをインストールします。

```
make install-libs
```

スタティックライブラリへの書き込みを可能とします。 これは後にデバッグシンボルを取り除くために必要となります。

```
chmod -v u+w /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

本パッケージは gzip 圧縮された.info ファイルをインストールしますが、共通的な dir を更新しません。 そこで以下のコマンドにより gzip ファイルを解凍した上で dir ファイルを更新します。

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir \
            /usr/share/info/libext2fs.info
```

必要なら、以下のコマンドを実行して追加のドキュメントをインストールします。

```
makeinfo -o      doc/com_err.info ../lib/et/com_err.texinfo
install -v -m644 doc/com_err.info /usr/share/info
install-info --dir-file=/usr/share/info/dir \
             /usr/share/info/com_err.info
```

6.21.2. E2fsprogs の構成

インストールプログラム:	badblocks, chattr, compile_et, debugfs, dumpe2fs, e2freefrag, e2fsck, e2image, e2initrd_helper, e2label, e2undo, filefrag, fsck.ext2, fsck.ext3, fsck.ext4, fsck.ext4dev, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mkfs.ext4dev, mklost+found, resize2fs, tune2fs
インストールライブラリ:	libcom_err.{a,so}, libe2p.{a,so}, libext2fs.{a,so}, libss.{a,so}
インストールディレクトリ:	/usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/share/et, /usr/share/ss

概略説明

badblocks	デバイス（通常はディスクパーティション）の不良ブロックを検索します。
chattr	ext2 ファイルシステム上のファイル属性を変更します。 ext2 ファイルシステムのジャーナリング版である ext3 ファイルシステムにおいても変更を行います。
compile_et	エラーテーブルコンパイラ。 これはエラーコード名とメッセージの一覧を、 com_err ライブラリを利用する C ソースコードとして変換するものです。
debugfs	ファイルシステムデバッガ。 これは ext2 ファイルシステムの状態を調査し変更することができます。
dumpe2fs	指定されたデバイス上にあるファイルシステムについて、 スーパーブロックの情報とブロックグループの情報を表示します。
e2freefrag	空きディスク部分のフラグメンテーションに関する情報を表示します。
e2fsck	ext2 ファイルシステムと ext3 ファイルシステムをチェックし、必要なら修復を行うことができます。
e2image	ext2 ファイルシステムの重要なデータをファイルに保存します。
e2initrd_helper	指定されたファイルシステムの FS タイプを表示します。 デバイス名やラベルを指定することもできます。
e2label	指定されたデバイス上にある ext2 ファイルシステムのラベルを表示または変更します。
e2undo	デバイス上にある ext2/ext3/ext4 ファイルシステムの undo ログを再実行します。 これは e2fsprogs プログラムが処理に失敗した際に undo を行うこともできます。
filefrag	特定のファイルのフラグメンテーション化がどれほど進んでいるかを表示します。
fsck.ext2	デフォルトでは ext2 ファイルシステムをチェックします。 これは e2fsck へのハードリンクです。
fsck.ext3	デフォルトでは ext3 ファイルシステムをチェックします。 これは e2fsck へのハードリンクです。
fsck.ext4	デフォルトでは ext4 ファイルシステムをチェックします。 これは e2fsck へのハードリンクです。
fsck.ext4dev	デフォルトでは ext4 ファイルシステムの開発版をチェックします。 これは e2fsck へのハードリンクです。
logsave	コマンドの出力結果をログファイルに保存します。
lsattr	ext2 ファイルシステム上のファイル属性を一覧表示します。
mk_cmds	コマンド名とヘルプメッセージの一覧を、 サブシステムライブラリ libss を利用する C ソースコードとして変換するものです。
mke2fs	指定されたデバイス上に ext2 ファイルシステム、または ext3 ファイルシステムを生成します。
mkfs.ext2	デフォルトでは ext2 ファイルシステムを生成します。 これは mke2fs へのハードリンクです。
mkfs.ext3	デフォルトでは ext3 ファイルシステムを生成します。 これは mke2fs へのハードリンクです。

mkfs.ext4	デフォルトでは ext4 ファイルシステムを生成します。 これは mke2fs へのハードリンクです。
mkfs.ext4dev	デフォルトでは ext4 ファイルシステム開発版を生成します。 これは mke2fs へのハードリンクです。
mklost+found	ext2 ファイルシステム上に lost+found ディレクトリを生成するために利用します。 このコマンドはそのディレクトリに対してあらかじめディスクブロックの情報を割り当てておくことで、e2fsck コマンドの負荷を軽減します。
resize2fs	ext2 ファイルシステムを拡張または縮小するために利用します。
tune2fs	ext2 ファイルシステム上にて調整可能なシステムパラメータを調整します。
libcom_err	共通的なエラー表示ルーチン。
libe2p	以下のコマンド dumpe2fs、 chattr、 lsattr が利用します。
libext2fs	ユーザーレベルのプログラムが ext2 ファイルシステムを操作可能とするためのルーチンを提供します。
libss	debugfs コマンドが利用します。

6.22. Coreutils-8.5

Coreutils パッケージはシステムの基本的な特性を表示したり設定したりするためのユーティリティを提供します。

概算ビルド時間: 3.2 SBU
必要ディスク容量: 98 MB

6.22.1. Coreutils のインストール

このパッケージが提供するプログラムとして uname があります。 このプログラムは -p オプションを指定したとき、常に unknown を返すという問題があります。 インテルアーキテクチャの CPU に対して、以下のパッチによりこれを修正します。

```
case 'uname -m' in
 i?86 | x86_64) patch -Np1 -i ../coreutils-8.5-uname-2.patch ;;
esac
```

POSIX では Coreutils により生成されるプログラムは、マルチバイトロケールであっても、文字データを正しく取り扱うことを求めています。 以下のパッチは標準に準拠することと、国際化処理に関連するバグを解消することを行います。

```
patch -Np1 -i ../coreutils-8.5-i18n-1.patch
```

注記

このパッチには以前は多くのバグがありました。 新たなバグを発見したら、Coreutils の開発者に報告する前に、このパッチを適用せずにバグが再現するかどうかを確認してください。

Coreutils をコンパイルするための準備をします。

```
./configure --prefix=/usr \
    --enable-no-install-program=kill,uptime
```

configure オプションの意味：

--enable-no-install-program=kill,uptime
指定のプログラムは、後に他のパッケージからインストールするため Coreutils からはインストールしないことを指示します。

パッケージをコンパイルします。

```
make
```

テストスイートを実行しない場合は 「パッケージをインストールします。」 と書かれたところまで読み飛ばしてください。

テストスイートを実行します。 まずは root ユーザーに対するテストを実行します。

```
make NON_ROOT_USERNAME=nobody check-root
```

ここからのテストは nobody ユーザーにより実行します。 ただしいくつかのテストでは、複数のグループに属するユーザーを必要とします。 そのようなテストを確実に実施するために、一時的なグループを作って nobody ユーザーがそれに属するようにします。

```
echo "dummy:x:1000:nobody" >> /etc/group
```

特定のファイルのパーミッションを変更して root ユーザー以外でもコンパイルとテストができるようにします。

```
chown -Rv nobody .
```

テストを実行します。

```
su-tools nobody -s /bin/bash -c "make RUN_EXPENSIVE_TESTS=yes check"
```

一時的に作成したグループを削除します。

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

パッケージをインストールします。

```
make install
```

FHS が規定しているディレクトリにプログラムを移します。

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

LFS-ブートスクリプトパッケージにあるスクリプトでは、 head、 sleep、 nice に依存しているものがあります。ブート処理の初期段階においては /usr ディレクトリは認識されないため、上のプログラムはルートパーティションに移す必要があります。

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

6.22.2. Coreutils の構成

インストールプログラム:	base64, basename, cat, chcon, 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, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mktemp, mv, nice, nl, nohup, nproc, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, runcon, seq, sha1sum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami, yes
インストールライブラリ:	libstdbuf.so
インストールディレクトリ:	/usr/lib/coreutils

概略説明

base64	base64（RFC 3548）規格に従ってデータのエンコード、デコードを行います。
basename	ファイル名からパス部分と指定されたサフィックスを取り除きます。
cat	複数ファイルを連結して標準出力へ出力します。
chcon	ファイルやディレクトリに対してセキュリティコンテキスト（security context）を変更します。
chgrp	ファイルやディレクトリのグループ所有権を変更します。
chmod	指定されたファイルのパーミッションを、指定されたモードに変更します。 モードは、変更内容を表す文字表現か、8進数表現を用いることができます。
chown	ファイルやディレクトリの所有者またはグループを変更します。
chroot	指定したディレクトリを / ディレクトリとみなしてコマンドを実行します。
cksum	指定された複数のファイルについて、CRC（Cyclic Redundancy Check; 巡回冗長検査）チェックサム値とバイト数を表示します。
comm	ソート済の二つのファイルを比較して、一致しない固有の行と一致する行を三つのカラムに分けて出力します。
cp	ファイルをコピーします。
csplit	指定されたファイルを複数の新しいファイルに分割します。 分割は指定されたパターンか行数により行います。 そして分割後のファイルにはバイト数を出力します。
cut	指定されたフィールド位置や文字位置によってテキスト行を部分的に取り出します。
date	指定された書式により現在時刻を表示します。 またはシステム日付を設定します。
dd	指定されたブロックサイズとブロック数によりファイルをコピーします。 変換処理を行うことができます。
df	マウントされているすべてのファイルシステムに対して、ディスクの空き容量（使用量）を表示します。あるいは指定されたファイルを含んだファイルシステムについてのみの情報を表示します。
dir	指定されたディレクトリの内容を一覧表示します。（ls コマンドに同じ。）

dircolors	環境変数 LS_COLOR にセットするべきコマンドを出力します。 これは ls がカラー設定を行う際に利用します。
dirname	ファイル名から、ディレクトリ名以外のサフィックスを取り除きます。
du	カレントディレクトリ、指定ディレクトリ（サブディレクトリを含む）、指定された個々のファイルについて、それらが利用しているディスク使用量を表示します。
echo	指定された文字列を表示します。
env	環境設定を変更してコマンドを実行します。
expand	タブ文字を空白文字に変換します。
expr	表現式を評価します。
factor	指定された整数値すべてに対する素因数（prime factor）を表示します。
false	何も行わず処理に失敗します。これは常に失敗を意味するステータスコードを返して終了します。
fmt	指定されたファイル内にて段落を整形します。
fold	指定されたファイル内の行を折り返します。
groups	ユーザーの所属グループを表示します。
head	指定されたファイルの先頭10行（あるいは指定された行数）を表示します。
hostid	ホスト識別番号（16進数）を表示します。
id	現在のユーザーあるいは指定されたユーザーについて、有効なユーザーID、グループID、所属グループを表示します。
install	ファイルコピーを行います。その際にパーミッションモードを設定し、可能なら所有者やグループも設定します。
join	2つのファイル内にて共通項を持つ行を結合します。
link	指定された名称により、ファイルへのハードリンクを生成します。
ln	ファイルに対するハードリンク、あるいはソフトリンク（シンボリックリンク）を生成します。
logname	現在のユーザーのログイン名を表示します。
ls	指定されたディレクトリ内容を一覧表示します。
md5sum	MD5（Message Digest 5）チェックサム値を表示、あるいはチェックします。
mkdir	指定された名前のディレクトリを生成します。
mkfifo	指定された名前の FIFO（First-In, First-Out）を生成します。 これは UNIX の用語で 「名前付きパイプ（named pipe)」 とも呼ばれます。
mknod	指定された名前のデバイスノードを生成します。 デバイスノードはキャラクタ型特殊ファイル（character special file)、ブロック特殊ファイル（block special file)、FIFO です。
mktemp	安全に一時ファイルを生成します。 これはスクリプト内にて利用されます。
mv	ファイルあるいはディレクトリを移動、名称変更します。
nice	スケジューリング優先度を変更してプログラムを実行します。
nl	指定されたファイル内の行を数えます。
nohup	ハングアップに関係なくコマンドを実行します。 その出力はログファイルにリダイレクトされます。
nproc	プロセスが利用可能なプロセスユニット（processing unit）の数を表示します。
od	ファイル内容を 8進数または他の書式でダンプします。
paste	指定された複数ファイルを結合します。 その際には各行を順に並べて結合し、その間をタブ文字で区切ります。
pathchk	ファイル名が有効で移植可能であるかをチェックします。
pinky	軽量な finger クライアント。 指定されたユーザーに関する情報を表示します。
pr	ファイルを印刷するために、ページ番号を振りカラム整形を行います。
printenv	環境変数の内容を表示します。
printf	指定された引数を指定された書式で表示します。 C 言語の printf 関数に似ています。
ptx	指定されたファイル内のキーワードに対して整列済インデックス（permuted index）を生成します。
pwd	現在の作業ディレクトリ名を表示します。
readlink	指定されたシンボリックリンクの対象を表示します。

rm	ファイルまたはディレクトリを削除します。
rmdir	ディレクトリが空である時にそのディレクトリを削除します。
runcon	指定されたセキュリティコンテキストでコマンドを実行します。
seq	指定された範囲と増分に従って数値の並びを表示します。
sha1sum	160 ビットの SHA1（Secure Hash Algorithm 1）チェックサム値を表示またはチェックします。
sha224sum	224 ビットの SHA1 チェックサム値を表示またはチェックします。
sha256sum	256 ビットの SHA1 チェックサム値を表示またはチェックします。
sha384sum	384 ビットの SHA1 チェックサム値を表示またはチェックします。
sha512sum	512 ビットの SHA1 チェックサム値を表示またはチェックします。
shred	指定されたファイルに対して、複雑なパターンデータを繰り返し上書きすることで、データ復旧を困難なものにします。
shuf	テキスト行を入れ替えます。
sleep	指定時間だけ停止します。
sort	指定されたファイル内の行をソートします。
split	指定されたファイルを、バイト数または行数を指定して分割します。
stat	ファイルやファイルシステムのステータスを表示します。
stdbuf	本コマンド実行により、標準ストリームに対するバッファリング操作を変更します。
stty	端末回線の設定や表示を行います。
sum	指定されたファイルのチェックサムやブロック数を表示します。
sync	ファイルシステムのバッファを消去します。 変更のあったブロックは強制的にディスクに書き出し、スーパーブロック（super block）を更新します。
tac	指定されたファイルを逆順にして連結します。
tail	指定されたファイルの最終の10行（あるいは指定された行数）を表示します。
tee	標準入力を読み込んで、標準出力と指定ファイルの双方に出力します。
test	ファイルタイプの比較やチェックを行います。
timeout	指定時間内だけコマンドを実行します。
touch	ファイルのタイムスタンプを更新します。 そのファイルに対するアクセス時刻、更新時刻を現在時刻にするものです。 そのファイルが存在しなかった場合はゼロバイトのファイルを新規生成します。
tr	標準入力から読み込んだ文字列に対して、変換・圧縮・削除を行います。
true	何も行わず処理に成功します。これは常に成功を意味するステータスコードを返して終了します。
truncate	ファイルを指定されたサイズに縮小または拡張します。
tsort	トポロジカルソート（topological sort）を行います。 指定されたファイルの部分的な順序に従って並び替えリストを出力します。
tty	標準入力に接続された端末のファイル名を表示します。
uname	システム情報を表示します。
unexpand	空白文字をタブ文字に変換します。
uniq	連続する同一行を一行のみ残して削除します。
unlink	指定されたファイルを削除します。
users	現在ログインしているユーザー名を表示します。
vdir	ls -l と同じ。
wc	指定されたファイルの行数、単語数、バイト数を表示します。 複数ファイルが指定された場合はこれに加えて合計も出力します。
who	誰がログインしているかを表示します。
whoami	現在有効なユーザーIDに関連づいているユーザー名を表示します。
yes	処理が停止されるまで繰り返して 「y」 または指定文字を出力します。
libstdbuf	stdbuf が利用するライブラリ。

6.23. Iana-Etc-2.30

Iana-Etc パッケージはネットワークサービスやプロトコルのためのデータを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.3 MB

6.23.1. Iana-Etc のインストール

以下のコマンドを実行します。 これは IANA が提供している生のデータを正しい書式のデータとして変換し /etc/protocols ファイルと /etc/services ファイルとして生成します。

```
make
```

このパッケージにはテストスイートはありません。

パッケージをインストールします。

```
make install
```

6.23.2. Iana-Etc の構成

インストールファイル: /etc/protocols, /etc/services

概略説明

/etc/protocols	TCP/IP により利用可能な様々な DARPA インターネットプロトコル (DARPA Internet protocols) を記述しています。
/etc/services	インターネットサービスを分かりやすく表現した名称と、その割り当てポートおよびプロトコルの種類の対応情報を提供します。

6.24. M4-1.4.14

M4 パッケージはマクロプロセッサを提供します。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 14.2 MB

6.24.1. M4 のインストール

不足しているインクルードディレクティブを追加します。 これがないと、M4 が Glibc-2.12.1 においてビルドできません。

```
sed -i -e '/"m4.h"/a\
#include <sys/stat.h>' src/path.c
```

M4 をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするために以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.24.2. M4 の構成

インストールプログラム: m4

概略説明

m4 指定されたファイル内のマクロ定義を展開して、そのコピーを生成します。 マクロ定義には埋め込み (built-in) マクロとユーザー定義マクロがあり、いくらでも引数を定義することができます。 マクロ定義の展開だけでなく m4 には以下のような埋め込み関数があります。 指定ファイルの読み込み、Unix コマンド実行、整数演算処理、テキスト操作、再帰処理などです。 m4 プログラムはコンパイラのフロントエンドとして利用することができ、それ自体でマクロプロセッサとして用いることもできます。

6.25. Bison-2.4.3

Bison パッケージは構文解析ツールを提供します。

概算ビルド時間: 1.1 SBU
必要ディスク容量: 19.2 MB

6.25.1. Bison のインストール

Bison をコンパイルするための準備をします。

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

bison プログラムが $PATH 上にない場合に、この Bison の configure を行ってビルドすると、国際化されたエラーメッセージのサポートがないままビルドされてしまいます。 これを正すために以下の設定を追加します。

```
echo '#define YYENABLE_NLS 1' >> lib/config.h
```

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするなら以下を実行します。(約 0.5 SBU)

```
make check
```

パッケージをインストールします。

```
make install
```

6.25.2. Bison の構成

インストールプログラム: bison, yacc
インストールライブラリ: liby.a
インストールディレクトリ: /usr/share/bison

概略説明

bison 構文規則の記述に基づいて、テキストファイルの構造を解析するプログラムを生成します。 Bison は Yacc (Yet Another Compiler Compiler) の互換プログラムです。

yacc bison のラッパースクリプト。 yacc プログラムがあるなら bison を呼び出さずに yacc を実行します。 -y オプションが指定された時は bison を実行します。

liby.a Yacc 互換の関数として yyerror 関数と main 関数を含むライブラリです。 このライブラリはあまり使い勝手の良いものではありません。 ただし POSIX ではこれが必要になります。

6.26. Procps-3.2.8

Procps パッケージはプロセス監視を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 2.3 MB

6.26.1. Procps のインストール

watch コマンドにおいて、ユニコードに関する問題を修正するためにパッチを適用します。

```
patch -Np1 -i ../procps-3.2.8-watch_unicode-1.patch
```

Makefile におけるバグを修正します。 これは make-3.82 を利用した場合に Procps がビルドできない点を修正するものです。

```
sed -i -e 's@\*/module.mk@proc/module.mk ps/module.mk@' Makefile
```

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

6.26.2. Procps の構成

インストールプログラム: free, kill, pgrep, pkill, pmap, ps, pwdx, skill, slabtop, snice, sysctl, tload, top, uptime, vmstat, w, watch
インストールライブラリ: libproc.so

概略説明

free 物理メモリ、スワップメモリの双方において、メモリの使用量、未使用量を表示します。
kill プロセスに対してシグナルを送信します。
pgrep プロセスの名前などの属性によりプロセスを調べます。
pkill プロセスの名前などの属性によりプロセスに対してシグナルを送信します。
pmap 指定されたプロセスのメモリマップを表示します。
ps 現在実行中のプロセスを一覧表示します。
pwdx プロセスが実行されているカレントディレクトリを表示します。
skill 指定された条件に合致するプロセスに対してシグナルを送信します。
slabtop リアルタイムにカーネルのスラブキャッシュ（slab cache）情報を詳細に示します。
snice 指定された条件に合致するプロセスのスケジュール優先度（scheduleing priority）を表示します。
sysctl システム稼動中にカーネル設定を修正します。
tload システムの負荷平均（load average）をグラフ化して表示します。
top CPU をより多く利用しているプロセスの一覧を表示します。 これはリアルタイムにプロセッサの動作状況を逐次表示します。
uptime システムの稼動時間、ログインユーザー数、システム負荷平均（load average）を表示します。
vmstat 仮想メモリの統計情報を表示します。 そこではプロセス、メモリ、ページング、ブロック入出力（Input/Output; IO）トラップ、CPU 使用状況を表示します。
w どのユーザーがログインしていて、どこから、そしていつからログインしているかを表示します。
watch 指定されたコマンドを繰り返し実行します。 そしてその出力結果の先頭の一画面分を表示します。 出力結果が時間の経過とともにどのように変わるかを確認することができます。
libproc 本パッケージのほとんどのプログラムが利用している関数を提供します。

6.27. Grep-2.6.3

Grep パッケージはファイル内の検索を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 7.3 MB

6.27.1. Grep のインストール

Grep をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.27.2. Grep の構成

インストールプログラム: egrep, fgrep, grep

概略説明

egrep 拡張正規表現（extended regular expression）にマッチした行を表示します。
fgrep 固定文字列の一覧にマッチした行を表示します。
grep 基本的な正規表現に合致した行を出力します。

6.28. Readline-6.1

Readline パッケージは、コマンドラインの編集や履歴管理を行うライブラリを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 13.8 MB

6.28.1. Readline のインストール

Readline を再インストールすると、それまでの古いライブラリは <ライブラリ名>.old というファイル名でコピーされます。 これは普通は問題ないことですが ldconfig によるリンクに際してエラーを引き起こすことがあります。 これを避けるため以下の二つの sed コマンドを実行します。

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

Readline のドキュメントに示されるバージョン番号を適切なものにします。

```
sed -i -e 's/0x0600/0x0601/' \
       -e 's/6\.0/6.1/' \
       -e 's/RL_VERSION_MINOR\t0/RL_VERSION_MINOR\t1/' readline.h
```

Readline をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make SHLIB_LIBS=-lncurses
```

make オプションの意味：

SHLIB_LIBS=-lncurses
このオプションにより Readline を libncurses ライブラリ（その実体は libncursesw ライブラリ）にリンクします。

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

スタティックライブラリを適切なディレクトリに移動します。

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

次に /lib ディレクトリにある .so ファイルを削除して、それらを /usr/lib にリンクし直します。

```
rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.6 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.6 /usr/lib/libhistory.so
```

必要ならドキュメントをインストールします。

```
mkdir   -v      /usr/share/doc/readline-6.1
install -v -m644 doc/*.{ps,pdf,html,dvi} \
                 /usr/share/doc/readline-6.1
```

6.28.2. Readline の構成

インストールライブラリ: libhistory.{a,so}, libreadline.{a,so}
インストールディレクトリ: /usr/include/readline, /usr/share/readline, /usr/share/doc/readline-6.1

概略説明

libhistory 入力履歴を適切に再現するためのユーザーインターフェースを提供します。

libreadline　コマンドラインインターフェースを提供している様々なコマンドにおいて、適切なインターフェースを提供します。

6.29. Bash-4.1

Bash は Bourne-Again SHell を提供します。

概算ビルド時間: 1.4 SBU
必要ディスク容量: 35 MB

6.29.1. Bash のインストール

Bash のアップストリームにより報告あるいは修正された数種のバグフィックスを適用します。

```
patch -Np1 -i ../bash-4.1-fixes-2.patch
```

Bash をコンパイルするための準備を行います。

```
./configure --prefix=/usr --bindir=/bin \
    --htmldir=/usr/share/doc/bash-4.1 --without-bash-malloc \
    --with-installed-readline
```

configure オプションの意味：

--htmldir
このオプションは HTML ドキュメントをインストールするディレクトリを指定します。

--with-installed-readline
このオプションは Bash が持つ独自の readline ライブラリではなく、既にインストールした readline ライブラリを用いることを指示します。

パッケージをコンパイルします。

```
make
```

テストスィートを実行しない場合は 「パッケージをインストールします。」 と書かれた箇所まで読み飛ばしてください。

テストを実施するにあたっては nobody ユーザーによるソースツリーへの書き込みを可能とします。

```
chown -Rv nobody .
```

nobody ユーザーでテストを実行します。

```
su-tools nobody -s /bin/bash -c "make tests"
```

パッケージをインストールします。

```
make install
```

新たにコンパイルした bash プログラムを実行します。（この時点までに実行されていたものが置き換えられます。）

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

注記

ここで指定しているパラメータは、対話形式のログインシェルとして、またハッシュ機能を無効にして bash プロセスを起動します。 これにより新たに構築するプログラム類は構築後すぐに利用できることになります。

6.29.2. Bash の構成

インストールプログラム: bash, bashbug, sh（bash へのリンク）
インストールディレクトリ: /usr/share/doc/bash-4.1

概略説明

bash 広く活用されているコマンドインタープリタ。 処理実行前には、指示されたコマンドラインを様々に展開したり置換したりします。 この機能があるからこそ、インタープリタ機能を強力なものにしています。

bashbug	bash に関連したバグ報告を、標準書式で生成しメール送信することを補助するシェルスクリプトです。
sh	bash プログラムへのシンボリックリンク。 sh として起動された際には、かつてのバージョンである sh の起動時の動作と、出来るだけ同じになるように振舞います。 同時に POSIX 標準に適合するよう動作します。

6.30. Libtool-2.2.10

Libtool パッケージは GNU 汎用ライブラリをサポートするスクリプトを提供します。 これは複雑な共有ライブラリをラップして一貫した可搬性を実現します。

概算ビルド時間: 3.7 SBU
必要ディスク容量: 35 MB

6.30.1. Libtool のインストール

Libtool をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。(約 3.0 SBU)

```
make check
```

パッケージをインストールします。

```
make install
```

6.30.2. Libtool の構成

インストールプログラム: libtool,libtoolize
インストールライブラリ: libltdl.{a,so}
インストールディレクトリ: /usr/include/libltdl, /usr/share/libtool

概略説明

libtool 汎用的なライブラリ構築支援サービスを提供します。
libtoolize パッケージに対して libtool によるサポートを加える標準的手法を提供します。
libltdl dlopen を行うライブラリの複雑さを隠蔽します。

6.31. GDBM-1.8.3

GDBM パッケージは GNU データベースマネージャを提供します。 このデータベースはディスクファイル形式（disk file format）のデータベースで、キーとデータのペア情報を一つのファイルに保持します。 各レコードのデータはユニークキーによりインデックスづけされます。 テキストファイルに保存された状態に比べて、より早く情報を抽出することができます。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 2.7 MB

6.31.1. GDBM のインストール

GDBM をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

このパッケージにはテストスイートはありません。

パッケージをインストールします。

```
make install
```

さらに DBM、NDBM との互換性のあるヘッダファイルをインストールします。 LFS では取り扱っていないパッケージの中には、そのような dbm の古い処理ルーチンを使っているものがあるためです。

```
make install-compat
```

インストール時に多少の問題があるため直します。 info ファイルの目次に GDBM を追加するものです。

```
install-info --dir-file=/usr/info/dir /usr/info/gdbm.info
```

6.31.2. GDBM の構成

インストールライブラリ: libgdbm.{so,a}, libgdbm_compat.{so,a}

概略説明

libgdbm ハッシュデータベースを取り扱う関数を提供します。

6.32. Inetutils-1.8

Inetutils パッケージはネットワーク制御を行う基本的なプログラムを提供します。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 17 MB

6.32.1. Inetutils のインストール

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

configure オプションの意味：

--disable-ifconfig
このオプションは ifconfig プログラムをインストールしないようにします。 このプログラムはネットワークインターフェースを設定するために利用するものです。 LFS では IPRoute2 パッケージが提供する ip コマンドを使うことにしています。

--disable-logger
このオプションは logger プログラムをインストールしないようにします。 このプログラムはシステムログデーモンに対してメッセージ出力を行うスクリプトにて利用されます。 ここでこれをインストールしないのは、後に Util-linux パッケージにおいて、以前のバージョンをインストールするためです。

--disable-syslogd
このオプションは Inetutils がシステムログデーモンをインストールしないようにします。 これらは Sysklogd パッケージにおいてインストールします。

--disable-whois
このオプションは whois のクライアントプログラムをインストールしないようにします。 このプログラムはもはや古いものです。 より良い whois プログラムのインストール手順については BLFS ブックにて説明しています。

--disable-servers
このオプションは Inetutils パッケージに含まれる様々なネットワークサーバーをインストールしないようにします。 これらのサーバーは基本的な LFS システムには不要なものと考えられます。 サーバーの中には本質的にセキュアでないものがあり、信頼のあるネットワーク内でのみしか安全に扱うことができないものもあります。 より詳細な情報は http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html を参照してください。 サーバーの多くは、これに代わる他の適切なものが存在します。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

プログラムのいくつかを FHS コンプライアントが定めるディレクトリに移動させます。

```
mv -v /usr/bin/{hostname,ping,ping6} /bin
mv -v /usr/bin/traceroute /sbin
```

6.32.2. Inetutils の構成

インストールプログラム: ftp, hostname, ping, ping6, rcp, rexec, rlogin, rsh, talk, telnet, tftp, traceroute

概略説明

ftp ファイル転送プロトコル（file transfer protocol）に基づくプログラム。
hostname ホスト名の表示または設定を行います。
ping エコーリクエスト（echo-request）パケットを送信し、返信にどれだけ要したかを表示します。

ping6	IPv6 ネットワーク向けの ping
rcp	リモートファイルコピーを行います。
rexec	リモートホスト上にてコマンドを実行します。
rlogin	リモートログインを行います。
rsh	リモートシェルを起動します。
talk	他ユーザーとのチャットに利用します。
telnet	TELNET プロトコルインターフェース。
tftp	軽量なファイル転送プログラム。(trivial file transfer program)
traceroute	処理起動したホストからネットワーク上の他のホストまで、送出したパケットの経由ルートを追跡します。 その合間に検出されたすべての hops (= ゲートウェイ) も表示します。

6.33. Perl-5.12.1

Perl パッケージは Perl 言語 (Practical Extraction and Report Language) を提供します。

概算ビルド時間: 5.5 SBU
必要ディスク容量: 171 MB

6.33.1. Perl のインストール

Perl の設定ファイルが /etc/hosts ファイルを参照するので、まずはこのファイルを生成します。 このファイルはテストスイートを実行する際にも利用されます。

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

ここでビルドするバージョンの Perl は Compress::Raw::Zlib モジュールをビルドします。 デフォルトではビルドの際に Zlib のソースを内部的にコピーします。 以下のコマンドは、既にインストールされている Zlib ライブラリを用いるようにします。

```
sed -i -e "s|BUILD_ZLIB\s*= True|BUILD_ZLIB = False|"           \
       -e "s|INCLUDE\s*= ./zlib-src|INCLUDE    = /usr/include|" \
       -e "s|LIB\s*= ./zlib-src|LIB        = /usr/lib|"        \
    cpan/Compress-Raw-Zlib/config.in
```

Perl のビルド設定を完全に制御したい場合は、以下のコマンドから 「-des」 オプションを取り除くことで、手作業により操作を進めます。 Perl が自動的に判別するデフォルト設定に従うので良いのであれば、以下のコマンドにより Perl をコンパイルするための準備をします。

```
sh Configure -des -Dprefix=/usr \
                  -Dvendorprefix=/usr           \
                  -Dman1dir=/usr/share/man/man1 \
                  -Dman3dir=/usr/share/man/man3 \
                  -Dpager="/usr/bin/less -isR"  \
                  -Duseshrplib
```

configure オプションの意味：

-Dvendorprefix=/usr
このオプションは各種の perl モジュールをどこにインストールするかを指定します。

-Dpager="/usr/bin/less -isR"
このオプションは perldoc プログラムが less プログラムを呼び出す際のエラーを正します。

-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3
まだ Groff をインストールしていないので Configure スクリプトが Perl の man ページを必要としないと判断してしまいます。 このオプションを指定することによりその判断を正します。

-Duseshrplib
Perl モジュールの中で必要とされる共有ライブラリ libperl をビルドします。

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。(約 2.5 SBU)

```
make test
```

パッケージをインストールします。

```
make install
```

6.33.2. Perl の構成

インストールプログラム:	a2p, c2ph, config_data, corelist, cpan, cpan2dist, cpanp, cpanp-run-perl, dprofpp, enc2xs, find2perl, h2ph, h2xs, instmodsh, libnetcfg, perl, perl5.12.1 (perl へのリンク), perlbug, perldoc, perlivp, perlthanks (perlbug へのリンク), piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, prove, psed (s2p へのリンク), pstruct (c2ph へのリンク), ptar, ptardiff, s2p, shasum, splain, xsubpp
インストールライブラリ:	ここでは列記できないほどの数多くのライブラリ
インストールディレクトリ:	/usr/lib/perl5

概略説明

a2p	awk スクリプトを Perl スクリプトに変換します。
c2ph	cc -g -S によって生成されるような C 言語構造体をダンプします。
config_data	Perl モジュールの設定を検索または変更します。
corelist	Module::CoreList に対するコマンドラインフロントエンド。
cpan	コマンドラインから CPAN (Comprehensive Perl Archive Network) との通信を行います。
cpan2dist	CPANPLUS の配布物生成ツール。
cpanp	CPANPLUS ランチャー。
cpanp-run-perl	Spawn プロセスにおいて出力処理が行われた後に、出力バッファをクリアするために利用するPerl スクリプト。
dprofpp	Perl プロファイルデータを表示します。
enc2xs	Unicode キャラクターマッピングまたは Tcl エンコーディングファイルから、 Perl の Encode 拡張モジュールを構築します。
find2perl	find コマンドを Perl に変換します。
h2ph	C 言語のヘッダーファイル .h を Perl のヘッダーファイル .ph に変換します。
h2xs	C 言語のヘッダーファイル .h を Perl 拡張 (Perl extension) に変換します。
instmodsh	インストールされている Perl モジュールを調査するシェルスクリプト。 インストールされたモジュールから tarball を作ることもできます。
libnetcfg	libnet ライブラリの設定に利用します。
perl	C 言語、 sed、 awk、 sh の持つ機能を寄せ集めて出来上がった言語。
perl5.12.1	perl へのハードリンク。
perlbug	Perl およびそのモジュールに関するバグ報告を生成して、電子メールを送信します。
perldoc	pod フォーマットのドキュメントを表示します。 pod フォーマットは Perl のインストールツリーあるいは Perl スクリプト内に埋め込まれています。
perlivp	Perl Installation Verification Procedure のこと。 Perl とライブラリが正しくインストールできているかを調べるものです。
perlthanks	感謝のメッセージ (Thank you messages) を電子メールで Perl 開発者に送信します。
piconv	キャラクターエンコーディングを変換する iconv の Perl バージョン。
pl2pm	Perl4 の .pl ファイルを Perl5 の .pm モジュールファイルへの変換を行うツール。
pod2html	pod フォーマットから HTML フォーマットに変換します。
pod2latex	pod フォーマットから LaTeX フォーマットへ変換します。
pod2man	pod データを *roff の入力ファイル形式に変換します。
pod2text	pod データをアスキーテキスト形式に変換します。
pod2usage	ファイル内に埋め込まれた pod ドキュメントから使用方法の記述部分を表示します。
podchecker	pod 形式の文書ファイルに対して文法をチェックします。
podselect	pod ドキュメントに対して指定したセクションを表示します。
prove	Test::Harness モジュールのテストを行うコマンドラインツール。

psed	ストリームエディタ sed の Perl バージョン。
pstruct	cc -g -S によって生成されるような C 言語構造体をダンプします。
ptar	Perl で書かれた tar 相当のプログラム。
ptardiff	アーカイブの抽出前後を比較する Perl プログラム。
s2p	sed スクリプトを Perl スクリプトに変換します。
shasum	SHA チェックサム値を表示またはチェックします。
splain	Perl スクリプトの警告エラーの診断結果を詳細（verbose）に出力するために利用します。
xsubpp	Perl の XS コードを C 言語コードに変換します。

6.34. Autoconf-2.67

Autoconf パッケージは、ソースコードを自動的に設定するシェルスクリプトの生成を行うプログラムを提供します。

概算ビルド時間: 4.8 SBU
必要ディスク容量: 12.4 MB

6.34.1. Autoconf のインストール

Autoconf をコンパイルするための準備を行います。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

このテストはおよそ 4.7 SBU ほど要します。そのうちの 6つのテストは Automake を利用するものであるためスキップされます。 すべてのテストを網羅したいなら、Automake をインストールした後に、再度テストを実行することが必要です。

パッケージをインストールします。

```
make install
```

6.34.2. Autoconf の構成

インストールプログラム: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, ifnames
インストールディレクトリ: /usr/share/autoconf

概略説明

autoconf	ソースコードを提供するソフトウェアパッケージを自動的に設定する（configure する）シェルスクリプトを生成します。 これにより数多くの Unix 互換システムへの適用を可能とします。 生成される設定（configure）スクリプトは独立して動作します。 つまりこれを実行するにあたっては autoconf プログラムを必要としません。
autoheader	C言語の #define 文を configure が利用するためのテンプレートファイルを生成するツール。
autom4te	M4 マクロプロセッサに対するラッパー。
autoreconf	autoconf と automake のテンプレートファイルが変更された時に、自動的に autoconf、autoheader、 aclocal、 automake、 gettextize、 libtoolize を無駄なく適正な順で実行します。
autoscan	ソフトウェアパッケージに対する configure.in ファイルの生成をサポートします。 ディレクトリツリー内のソースファイルを調査して、共通的な可搬性に関わる問題を見出します。 そして configure.scan ファイルを生成して、そのパッケージの configure.in ファイルの雛形として提供します。
autoupdate	configure.in ファイルにおいて、かつての古い autoconf マクロが利用されている場合に、それを新しいマクロに変更します。
ifnames	ソフトウェアパッケージにおける configure.in ファイルの記述作成をサポートします。 これはそのパッケージが利用する C プリプロセッサの条件ディレクティブの識別子を出力します。 可搬性を考慮した構築ができている場合は、本プログラムが configure スクリプトにおいて何をチェックするべきかを決定してくれます。 また autoscan によって生成された configure.in ファイルでの過不足を調整する働きもします。

6.35. Automake-1.11.1

Automake パッケージは Autoconf が利用する Makefile などを生成するプログラムを提供します。

概算ビルド時間: 18.3 SBU
必要ディスク容量: 28.8 MB

6.35.1. Automake のインストール

Automake をコンパイルするための準備をします。

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.11.1
```

パッケージをコンパイルします。

```
make
```

ビルド結果をテストするには、以下を実行します。

```
make check
```

このテストには 10 SBU ほど要します。

パッケージをインストールします。

```
make install
```

6.35.2. Automake の構成

インストールプログラム: acinstall, aclocal, aclocal-1.11.1, automake, automake-1.11.1, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree, ylwrap
インストールディレクトリ: /usr/share/aclocal-1.11, /usr/share/automake-1.11, /usr/share/doc/automake-1.11.1

概略説明

acinstall	aclocal 風の M4 ファイルをインストールするスクリプト。
aclocal	configure.in ファイルの内容に基づいて aclocal.m4 ファイルを生成します。
aclocal-1.11.1	aclocal へのハードリンク。
automake	Makefile.am ファイルから Makefile.in ファイルを自動生成するツール。 パッケージ内のすべての Makefile.in ファイルを作るには、このプログラムをトップディレクトリから実行します。configure.in ファイルを調べて、適切な Makefile.am ファイルを検索します。そして対応する Makefile.in ファイルを生成します。
automake-1.11.1	automake へのハードリンク。
compile	コンパイラのラッパースクリプト。
config.guess	指定されたビルドタイプ、ホストタイプ、ターゲットタイプに対しての正規化した「三つ組」を推定するスクリプト。
config.sub	設定を検証するサブルーチンスクリプト。
depcomp	プログラムをコンパイルするためのスクリプトで、コンパイル結果を得ると同時に依存情報も生成します。
elisp-comp	Emacs Lisp コードをバイトコンパイルします。
install-sh	プログラムやスクリプトやデータファイルをインストールするスクリプト。
mdate-sh	ファイルやディレクトリの更新時刻を表示するスクリプト。
missing	インストール中に GNU プログラムが存在しなかった場合に、共通のスタブ（stub）プログラムとして動作するスクリプト。
mkinstalldirs	ディレクトリツリーを生成するスクリプト。
py-compile	Python プログラムをコンパイルします。
symlink-tree	ディレクトリツリーに対するシンボリックリンクのツリーを生成するスクリプト。
ylwrap	lex と yacc に対するラッパースクリプト。

6.36. Bzip2-1.0.5

Bzip2 パッケージはファイル圧縮、伸長（解凍）を行うプログラムを提供します。 テキストファイルであれば、これまでよく用いられてきた gzip に比べて bzip2 の方が圧縮率の高いファイルを生成できます。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 6.4 MB

6.36.1. Bzip2 のインストール

以下の２つのパッチを適用します。 本パッケージのドキュメントをインストールするとともに、ハードコーディングされているバージョン番号を修正します。

```
patch -Np1 -i ../bzip2-1.0.5-install_docs-1.patch
patch -Np1 -i ../bzip2-1.0.5-version_fixes-1.patch
```

以下のコマンドによりシンボリックリンクを相対的なものとしてインストールします。

```
sed -i 's@\(ln -s -f \)$(PREFIX)/bin/@\1@' Makefile
```

Bzip2 をコンパイルするための準備をします。

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

make パラメータの意味：

-f Makefile-libbz2_so
このパラメータは Bzip2 のビルドにあたって通常の Makefile ファイルではなく Makefile-libbz2_so ファイルを利用することを指示します。 これはダイナミックライブラリ libbz2.so ライブラリをビルドし、Bzip2 の各種プログラムをこれにリンクします。

パッケージのコンパイルとテストを行います。

```
make
```

パッケージをインストールします。

```
make PREFIX=/usr install
```

共有化された bzip2 実行モジュールを /bin ディレクトリにインストールします。 また必要となるシンボリックリンクを生成し不要なものを削除します。

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

6.36.2. Bzip2 の構成

インストールプログラム: bunzip2 (bzip2 へのリンク), bzcat (bzip2 へのリンク), bzcmp (bzdiff へのリンク), bzdiff, bzegrep (bzgrep へのリンク), bzfgrep (bzgrep へのリンク), bzgrep, bzip2, bzip2recover, bzless (bzmore へのリンク), bzmore
インストールライブラリ: libbz2.{a,so}
インストールディレクトリ: /usr/share/doc/bzip2-1.0.5

概略説明

bunzip2 bzip2 で圧縮されたファイルを解凍します。
bzcat 解凍結果を標準出力に出力します。
bzcmp bzip2 で圧縮されたファイルに対して cmp を実行します。

bzdiff	bzip2 で圧縮されたファイルに対して diff を実行します。
bzegrep	bzip2 で圧縮されたファイルに対して egrep を実行します。
bzfgrep	bzip2 で圧縮されたファイルに対して fgrep を実行します。
bzgrep	bzip2 で圧縮されたファイルに対して grep を実行します。
bzip2	ブロックソート法（バロウズ-ホイラー変換）とハフマン符号化法を用いてファイル圧縮を行います。圧縮率は、従来用いられてきた 「Lempel-Ziv」 アルゴリズムによるもの、例えば gzip コマンドによるものに比べて高いものです。
bzip2recover	壊れた bzip2 ファイルの復旧を試みます。
bzless	bzip2 で圧縮されたファイルに対して less を実行します。
bzmore	bzip2 で圧縮されたファイルに対して more を実行します。
libbz2*	ブロックソート法（バロウズ-ホイラー変換）による可逆的なデータ圧縮を提供するライブラリ。

6.37. Diffutils-3.0

Diffutils パッケージはファイルやディレクトリの差分を表示するプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 6.3 MB

6.37.1. Diffutils のインストール

Diffutils をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

ビルド結果をテストするなら以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.37.2. Diffutils の構成

インストールプログラム: cmp, diff, diff3, sdiff

概略説明

cmp 二つのファイルを比較して、どこが異なるか、あるいは何バイト異なるかを示します。
diff 二つのファイルまたは二つのディレクトリを比較して、ファイル内のどの行に違いがあるかを示します。
diff3 三つのファイルの各行を比較します。
sdiff 二つのファイルを結合して対話的に結果を出力します。

6.38. Gawk-3.1.8

Gawk パッケージはテキストファイルを操作するプログラムを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 19 MB

6.38.1. Gawk のインストール

Gawk をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

必要ならドキュメントをインストールします。

```
mkdir -v /usr/share/doc/gawk-3.1.8
cp    -v doc/{awkforai.txt,*.{eps,pdf,jpg}} \
         /usr/share/doc/gawk-3.1.8
```

6.38.2. Gawk の構成

インストールプログラム: awk (gawk へのリンク), gawk, gawk-3.1.8, grcat, igawk, pgawk, pgawk-3.1.8, pwcat
インストールディレクトリ: /usr/lib/awk, /usr/share/awk

概略説明

awk gawk へのリンク。
gawk テキストファイルを操作するプログラム。 これは awk の GNU インプリメンテーションです。
gawk-3.1.8 gawk へのハードリンク。
grcat グループデータベースファイル /etc/group をダンプします。
igawk gawk に対してファイルをインクルードする機能を付与します。
pgawk gawk のプロファイル版。
pgawk-3.1.8 pgawk へのハードリンク。
pwcat パスワードデータベースファイル /etc/passwd をダンプします。

6.39. File-5.04

File パッケージは、指定されたファイルの種類を決定するユーティリティを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 9.5 MB

6.39.1. File のインストール

File をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.39.2. File の構成

インストールプログラム: file
インストールライブラリ: libmagic.{a,so}

概略説明

file 指定されたファイルの種類判別を行います。 処理にあたってはいくつかのテスト、すなわちファイルシステムテスト、マジックナンバーテスト、言語テストを行います。

libmagic マジックナンバーによりファイル判別を行うルーチンを含みます。 file プログラムがこれを利用しています。

6.40. Findutils-4.4.2

Findutils パッケージはファイル検索を行うプログラムを提供します。 このプログラムはディレクトリツリーを再帰的に検索したり、データベースの生成・保守・検索を行います。（データベースによる検索は再帰的検索に比べて処理速度は速いものですが、データベースが最新のものに更新されていない場合は信頼できない結果となります。）

概算ビルド時間: 0.5 SBU
必要ディスク容量: 22 MB

6.40.1. Findutils のインストール

Findutils をコンパイルするための準備をします。

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

configure オプションの意味：

--localstatedir
locate データベースの場所を FHS コンプライアンスが定めているディレクトリ /var/lib/locate に変更します。

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするなら以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

LFS ブートスクリプトパッケージでは、いくつかのスクリプトが find を利用しています。 /usr ディレクトリはブート処理の初めでは認識できないため、このプログラムはルートパーティションに置く必要があります。 同じく updatedb スクリプトは明示的なパスを修正する必要があります。

```
mv -v /usr/bin/find /bin
sed -i 's/find:=${BINDIR}/find:=\/bin/' /usr/bin/updatedb
```

6.40.2. Findutils の構成

インストールプログラム: bigram, code, find, frcode, locate, oldfind, updatedb, xargs
インストールディレクトリ: /usr/lib/findutils

概略説明

bigram かつて利用されていたコマンドで locate データベースを生成します。
code かつて利用されていたコマンドで locate データベースを生成します。 これは frcode の前身です。
find 指定された条件に合致するファイルを、指定されたディレクトリツリー内から検索します。
frcode updatedb コマンドから呼び出され、ファイル名の一覧を圧縮します。 これは前置圧縮（front-compression）を行うもので、データベースサイズを 1/4 から 1/5 に減らします。
locate ファイル名データベースを検索して、指定された文字列を含むもの、または検索パターンに合致するものを表示します。
oldfind find の古い版であり、find とは異なるアルゴリズムを用いています。
updatedb locate データベースを更新します。 これはすべてのファイルシステムを検索します。（検索非対象とする設定がない限りは、マウントされているすべてのファイルシステムを対象とします。）そして検索されたファイル名をデータベースに追加します。
xargs 指定されたコマンドに対してファイル名の一覧を受け渡して実行します。

6.41. Flex-2.5.35

Flex パッケージは、字句パターンを認識するプログラムを生成するユーティリティを提供します。

概算ビルド時間: 0.7 SBU
必要ディスク容量: 28 MB

6.41.1. Flex のインストール

C++ のスキャナ生成に含まれるバグを修正するためのパッチを適用します。 これがないと GCC-4.5.1 を用いた時にスキャナに関するコンパイルに失敗します。

```
patch -Np1 -i ../flex-2.5.35-gcc44-1.patch
```

Flex をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするために以下を実行します。(約 0.5 SBU)

```
make check
```

パッケージをインストールします。

```
make install
```

他のパッケージの中には lex ライブラリが /usr/lib ディレクトリにあるものとして動作しています。 これに対応するためシンボリックリンクを作成します。

```
ln -sv libfl.a /usr/lib/libl.a
```

プログラムの中には flex コマンドが用いられず、その前身である lex コマンドを実行しようとするものがあります。そういったプログラムへ対応するために lex という名のラッパースクリプトを生成します。 このスクリプトは lex のエミュレーションモードとして flex を実行します。

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex

exec /usr/bin/flex -l "$@"

# End /usr/bin/lex
EOF
chmod -v 755 /usr/bin/lex
```

必要ならドキュメントファイル flex.pdf をインストールします。

```
mkdir -v /usr/share/doc/flex-2.5.35
cp    -v doc/flex.pdf \
         /usr/share/doc/flex-2.5.35
```

6.41.2. Flex の構成

インストールプログラム: flex, lex
インストールライブラリ: libfl.a, libfl_pic.a

概略説明

flex テキスト内のパターンを認識するためのプログラムを生成するツール。 これは多彩なパターン検索の規則構築を可能とします。 これを利用することで特別なプログラムの生成が不要となります。

lex　　lex のエミュレーションモードとして flex を実行するスクリプト。
libfl.a　flex ライブラリ。

6.42. Gettext-0.18.1.1

Gettext パッケージは国際化を行うユーティリティを提供します。 各種プログラムに対して NLS (Native Language Support) を含めてコンパイルすることができます。 つまり各言語による出力メッセージが得られることになります。

概算ビルド時間: 5.8 SBU
必要ディスク容量: 125 MB

6.42.1. Gettext のインストール

Gettext をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/gettext-0.18.1.1
```

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするなら (3 SBU 程度の処理時間を要しますが) 以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.42.2. Gettext の構成

インストールプログラム:	autopoint, config.charset, config.rpath, envsubst, gettext, gettext.sh, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, recode-sr-latin, xgettext
インストールライブラリ:	libasprintf.{a,so}, libgettextlib.so, libgettextpo.{a,so}, libgettextsrc.so, preloadable_libintl.so
インストールディレクトリ:	/usr/lib/gettext, /usr/share/doc/gettext-0.18.1.1, /usr/share/gettext

概略説明

autopoint	Gettext 標準のインフラストラクチャーファイル (infrastructure file) をソースパッケージ内にコピーします。
config.charset	システム依存の、キャラクターエンコーディングのエイリアス対応表を出力します。
config.rpath	システムに依存する変数一覧を出力します。 その変数とは、実行モジュールにおける共有ライブラリの検索パスをどのように設定するかを示すものです。
envsubst	環境変数をシェル書式の文字列として変換します。
gettext	メッセージカタログ内の翻訳文を参照し、メッセージをユーザーの利用言語に変換します。
gettext.sh	主に gettext におけるシェル関数ライブラリとして機能します。
gettextize	パッケージの国際化対応を始めるにあたり、標準的な Gettext 関連ファイルを、指定されたパッケージのトップディレクトリにコピーします。
hostname	様々な書式のネットワークホスト名を表示します。
msgattrib	翻訳カタログ内のメッセージの属性に応じて、そのメッセージを抽出します。 またメッセージの属性を操作します。
msgcat	指定された .po ファイルを連結します。
msgcmp	二つの .po ファイルを比較して、同一の msgid による文字定義が両者に含まれているかどうかをチェックします。
msgcomm	指定された .po ファイルにて共通のメッセージを検索します。
msgconv	翻訳カタログを別のキャラクターエンコーディングに変換します。
msgen	英語用の翻訳カタログを生成します。
msgexec	翻訳カタログ内の翻訳文すべてに対してコマンドを適用します。

msgfilter	翻訳カタログ内の翻訳文すべてに対してフィルター処理を適用します。
msgfmt	翻訳カタログからバイナリメッセージカタログを生成します。
msggrep	指定された検索パターンに合致する、あるいは指定されたソースファイルに属する翻訳カタログの全メッセージを出力します。
msginit	新規に .po ファイルを生成します。 その時にはユーザーの環境設定に基づいてメタ情報を初期化します。
msgmerge	二つの翻訳ファイルを一つにまとめます。
msgunfmt	バイナリメッセージカタログを翻訳テキストに逆コンパイルします。
msguniq	翻訳カタログ中に重複した翻訳がある場合にこれを統一します。
ngettext	出力メッセージをユーザーの利用言語に変換します。 特に複数形のメッセージを取り扱います。
recode-sr-latin	セルビア語のテキストに対し、キリル文字からラテン文字にコード変換します。
xgettext	指定されたソースファイルから、翻訳対象となるメッセージ行を抽出して、翻訳テンプレートとして生成します。
libasprintf	autosprintf クラスを定義します。 これは C++ プログラムにて利用できる C 言語書式の出力ルーチンを生成するものです。〈string〉文字列と〈iostream〉ストリームを利用します。
libgettextlib	様々な Gettext プログラムが利用している共通的ルーチンを提供するプライベートライブラリです。 これは一般的な利用を想定したものではありません。
libgettextpo	.po ファイルの出力に特化したプログラムを構築する際に利用します。 Gettext が提供する標準的なアプリケーション（msgcomm、 msgcmp、 msgattrib、 msgen）などでは処理出来ないものがある場合に、このライブラリを利用します。
libgettextsrc	様々な Gettext プログラムが利用している共通的ルーチンを提供するプライベートライブラリです。 これは一般的な利用を想定したものではありません。
preloadable_libintl	LD_PRELOAD が利用するライブラリ。 翻訳されていないメッセージを収集（log）する libintl をサポートします。

6.43. Groff-1.20.1

Groff パッケージはテキストを処理して整形するプログラムを提供します。

概算ビルド時間: 0.7 SBU
必要ディスク容量: 66 MB

6.43.1. Groff のインストール

Groff はデフォルトの用紙サイズを設定する環境変数 PAGE を参照します。 米国のユーザーであれば PAGE=letter と設定するのが適当です。 その他のユーザーなら PAGE=A4 とするのが良いかもしれません。 このデフォルト用紙サイズはコンパイルにあたって設定されます。 「A4」 なり 「letter」 なりの値は /etc/papersize ファイルにて設定することも可能です。

Groff をコンパイルするための準備をします。

```
PAGE=<paper_size> ./configure --prefix=/usr
```

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make docdir=/usr/share/doc/groff-1.20.1 install
```

xman のようなドキュメント関連プログラムが正しく動作するように、以下のようなシンボリックリンクを作成します。

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

6.43.2. Groff の構成

インストールプログラム: addftinfo, afmtodit, chem, eqn, eqn2graph, gdiffmk, geqn (eqn へのリンク), grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (tbl へのリンク), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfroff, pfbtops, pic, pic2graph, post-grohtml, preconv, pre-grohtml, refer, roff2dvi, roff2html, roff2pdf, roff2ps, roff2text, roff2x, soelim, tbl, tfmtodit, troff
インストールディレクトリ: /usr/lib/groff, /usr/share/doc/groff-1.20.1, /usr/share/groff

概略説明

addftinfo troff のフォントファイルを読み込んで groff システムが利用する付加的なフォントメトリック情報を追加します。

afmtodit groff と grops が利用するフォントファイルを生成します。

chem 化学構造図 (chemical structure diagrams) を生成するための Groff プロセッサ。

eqn troff の入力ファイル内に埋め込まれている記述式をコンパイルして troff が解釈できるコマンドとして変換します。

eqn2graph troff の EQN (数式) を、刈り込んだ (crop した) イメージに変換します。

gdiffmk groff、nroff、troff の入力ファイルを比較して、その差異を出力します。

geqn eqn へのリンク。

grap2graph grap ダイアグラムを、刈り込んだ (crop した) ビットマップイメージに変換します。

grn gremlin 図を表すファイルを処理するための groff プリプロセッサ。

grodvi TeX の dvi フォーマットを生成するための groff ドライバプログラム。

groff groff 文書整形システムのためのフロントエンド。 通常は troff プログラムを起動し、指定されたデバイスに適合したポストプロセッサを呼び出します。

groffer groff ファイルや man ページを X 上や TTY 端末上に表示します。

grog	入力ファイルを読み込んで、印刷時には groff コマンドオプションのどれが必要かを推定します。コマンドオプションは -e、 -man、 -me、 -mm、 -ms、 -p、 -s のいずれかです。 そしてそのオプションを含んだ groff コマンドを表示します。
grolbp	Canon CAPSL プリンタ（LBP-4 または LBP-8 シリーズのレーザープリンタ）に対する groff ドライバプログラム。
grolj4	HP LaserJet 4 プリンタにて利用される PCL5 フォーマットの出力を生成する groff のドライバプログラム。
grops	GNU troff の出力を PostScript に変換します。
grotty	GNU troff の出力を、タイプライタ風のデバイスに適した形式に変換します。
gtbl	tbl へのリンク。
hpftodit	HP のタグ付けが行われたフォントメトリックファイルから、 groff -Tlj4 コマンドにて利用されるフォントファイルを生成します。
indxbib	指定されたファイル内に示される参考文献データベース（bibliographic database）に対しての逆引きインデックス（inverted index）を生成します。 これは refer、 lookbib、 lkbib といったコマンドが利用します。
lkbib	指定されたキーを用いて参考文献データベースを検索し、合致したすべての情報を表示します。
lookbib	（標準入力が端末であれば）標準エラー出力にプロンプトを表示して、標準入力から複数のキーワードを含んだ一行を読み込みます。 そして指定されたファイルにて示される参考文献データベース内に、そのキーワードが含まれるかどうかを検索します。 キーワードが含まれるものを標準出力に出力します。入力がなくなるまでこれを繰り返します。
mmroff	groff 用の単純なプリプロセッサ。
neqn	数式を ASCII（American Standard Code for Information Interchange）形式で出力します。
nroff	groff を利用して nroff コマンドをエミュレートするスクリプト。
pdfroff	groff を利用して pdf 文書ファイルを生成します。
pfbtops	.pfb フォーマットの PostScript フォントを ASCII フォーマットに変換します。
pic	troff または TeX の入力ファイル内に埋め込まれた図の記述を、 troff または TeX が処理できるコマンドの形式に変換します。
pic2graph	PIC ダイアグラムを、刈り込んだ（crop した）イメージに変換します。
post-grohtml	GNU troff の出力を HTML に変換します。
preconv	入力ファイルのエンコーディングを GNU troff が取り扱うものに変換します。
pre-grohtml	GNU troff の出力を HTML に変換します。
refer	ファイル内容を読み込んで、そのコピーを標準出力へ出力します。 ただし引用文を表す .[と .] で囲まれた行、および引用文をどのように処理するかを示したコマンドを意味する .R1 と .R2 で囲まれた行は、コピーの対象としません。
roff2dvi	roff ファイルを DVI フォーマットに変換します。
roff2html	roff ファイルを HTML フォーマットに変換します。
roff2pdf	roff ファイルを PDF フォーマットに変換します。
roff2ps	roff ファイルを ps ファイルに変換します。
roff2text	roff ファイルをテキストファイルに変換します。
roff2x	roff ファイルを他のフォーマットに変換します。
soelim	入力ファイルを読み込んで .so ファイル　の形式で記述されている行を、記述されている ファイルだけに置き換えます。
tbl	troff 入力ファイル内に埋め込まれた表の記述を troff が処理できるコマンドの形式に変換します。
tfmtodit	コマンド groff -Tdvi を使ってフォントファイルを生成します。
troff	Unix の troff コマンドと高い互換性を持ちます。 通常は groff コマンドを用いて本コマンドが起動されます。 groff コマンドは、プリプロセッサ、ポストプロセッサを、適切な順で適切なオプションをつけて起動します。

6.44. GRUB-1.98

GRUB パッケージは GRand Unified Bootloader を提供します。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 27.6 MB

6.44.1. GRUB のインストール

GRUB をコンパイルするための準備をします。

```
./configure --prefix=/usr          \
            --sysconfdir=/etc      \
            --disable-grub-emu-usb \
            --disable-grub-fstest  \
            --disable-efiemu
```

--disable オプションは、LFS で本当に必要となる機能やテストプログラムだけを生成するもので、ビルド結果を最小限に抑えます。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

GRUB を使ってシステムのブート起動設定を行う方法については 8.4. 「GRUB を用いたブートプロセスの設定」 で説明しています。

6.44.2. GRUB の構成

インストールプログラム: grub-bin2h, grub-editenv, grub-install, grub-mkconfig, grub-mkdevicemap, grub-mkelfimage, grub-mkimage, grub-mkisofs, grub-mkpasswd-pbkdf2, grub-mkrelpath, grub-mkrescue, grub-probe, grub-reboot, grub-script-check, grub-set-default, grub-setup
インストールディレクトリ: /usr/lib/grub, /etc/grub.d, /usr/share/grub

概略説明

grub-bin2h	バイナリファイルを C ヘッダファイルに変換します。
grub-editenv	環境ブロック（environment block）を編集するツール。
grub-install	指定したドライブに GRUB をインストールします。
grub-mkconfig	GRUB の設定ファイルを生成します。
grub-mkdevicemap	デバイスマップファイルを自動的に生成します。
grub-mkelfimage	GRUB のブートイメージ（bootable image）を生成します。
grub-mkimage	GRUB のブートイメージを生成します。
grub-mkisofs	ブータブルな ISO イメージを生成します。
grub-mkpasswd-pbkdf2	ブートメニューにて利用する、PBKDF2 により暗号化されたパスワードを生成します。
grub-mkrelpath	システムのパスをルートからの相対パスとします。
grub-mkrescue	フロッピーディスク用の GRUB のブートイメージを生成します。
grub-probe	指定されたパスやデバイスに対するデバイス情報を検証（probe）します。
grub-reboot	デフォルトのブートメニューを設定します。 これは次にブートした時だけ有効なものです。
grub-script-check	GRUB の設定スクリプトにおける文法をチェックします。
grub-set-default	デフォルトのブートメニューを設定します。
grub-setup	デバイスからのブートを行うためにイメージファイルをセットアップします。

6.45. Gzip-1.4

Gzip パッケージはファイルの圧縮、伸長（解凍）を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 3.3 MB

6.45.1. Gzip のインストール

Gzip をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

ルートファイルシステム上に置く必要のないプログラムを移動させます。

```
mv -v /bin/{gzexe,uncompress,zcmp,zdiff,zegrep} /usr/bin
mv -v /bin/{zfgrep,zforce,zgrep,zless,zmore,znew} /usr/bin
```

6.45.2. Gzip の構成

インストールプログラム: gunzip, gzexe, gzip, uncompress, zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, znew

概略説明

gunzip gzip により圧縮されたファイルを解凍します。

gzexe 自動解凍形式の実行ファイルを生成します。

gzip Lempel-Ziv (LZ77) 方式により指定されたファイルを圧縮します。

uncompress 圧縮されたファイルを解凍します。

zcat gzip により圧縮されたファイルを解凍して標準出力へ出力します。

zcmp gzip により圧縮されたファイルに対して cmp を実行します。

zdiff gzip により圧縮されたファイルに対して diff を実行します。

zegrep gzip により圧縮されたファイルに対して egrep を実行します。

zfgrep gzip により圧縮されたファイルに対して fgrep を実行します。

zforce 指定されたファイルが gzip により圧縮されている場合に、強制的に拡張子 .gz を付与します。 こうすることで gzip は再度の圧縮を行わないようになります。 これはファイル転送によってファイル名が切り詰められてしまった場合に活用することができます。

zgrep gzip により圧縮されたファイルに対して grep を実行します。

zless gzip により圧縮されたファイルに対して less を実行します。

zmore gzip により圧縮されたファイルに対して more を実行します。

znew compress フォーマットの圧縮ファイルを gzip フォーマットのファイルとして再圧縮します。 つまり .Z から .gz への変換を行います。

6.46. IPRoute2-2.6.35

IPRoute2 パッケージは IPV4 ベースの基本的または応用的ネットワーク制御を行うプログラムを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 5.7 MB

6.46.1. IPRoute2 のインストール

本パッケージにて提供している arpd プログラムは Berkeley DB に依存しています。 arpd はベースとする Linux システムにとって普通は必要となりません。 そこで Berkeley DB への依存を取り除くために、以下の sed コマンドを実行します。 arpd プログラムを必要とする場合は BLFS ブックの http://www.linuxfromscratch.org/blfs/view/svn/server/databases.html#db に示される Berkeley DB の構築手順に従ってください。

```
sed -i '/^TARGETS/s@arpd@@g' misc/Makefile
```

ip route get コマンドが、何も出力を生成しないバグを修正します。

```
sed -i '1289i\\tfilter.cloned = 2;' ip/iproute.c
```

パッケージをコンパイルします。

```
make DESTDIR=
```

make オプションの意味：

DESTDIR=
このオプションにより IPRoute2 の実行モジュール類を適切なディレクトリにインストールします。 デフォルトでは DESTDIR は /usr ディレクトリに設定されています。

このパッケージにテストスイートはありますが、このテストの前提条件からすると chroot 環境のもとでは信頼のあるテスト結果を得ることには無理があります。 もし LFS システムを構築した後にテストスイートを実施したいなら、カーネル設定において /proc/config.gz CONFIG_IKCONFIG_PROC ("General setup" -> "Enable access to .config through /proc/config.gz") のサポートを有効にしてカーネルをビルドしてください。 そしてサブディレクトリ testsuite/ にて 'make alltests' を実行してください。

パッケージをインストールします。

```
make DESTDIR= SBINDIR=/sbin MANDIR=/usr/share/man \
     DOCDIR=/usr/share/doc/iproute2-2.6.35 install
```

6.46.2. IPRoute2 の構成

インストールプログラム: ctstat (lnstat へのリンク), genl, ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (lnstat へのリンク), ss, tc
インストールディレクトリ: /etc/iproute2, /lib/tc, /usr/share/doc/iproute2-2.6.35, /usr/lib/tc

概略説明

ctstat 接続ステータスの表示ユーティリティ。

genl

ifcfg ip コマンドに対するシェルスクリプトラッパー。 http://www.skbuff.net/iputils/ にて提供されている iputils パッケージの arping プログラムと rdisk プログラムを利用します。

ifstat インターフェースの統計情報を表示します。 インターフェースによって送受信されたパケット量が示されます。

ip 主となる実行モジュールで、複数の機能性を持ちます。
ip link <デバイス名> はデバイスのステータスを参照し、またステータスの変更を行います。
ip addr はアドレスとその属性を参照し、新しいアドレスの追加、古いアドレスの削除を行います。
ip neighbor は、隣接ルーター (neighbor) の割り当てや属性を参照し、隣接ルーターの項目追加や古いものの削除を行います。
ip rule は、ルーティングポリシー (routing policy) を参照し、変更を行います。
ip route は、ルーティングテーブル (routing table) を参照し、ルーティングルール (routing table rule) を変更します。

ip tunnel は、IP トンネル（IP tunnel）やその属性を参照し、変更を行います。
ip maddr は、マルチキャストアドレス（multicast address）やその属性を参照し、変更を行います。
ip mroute は、マルチキャストルーティング（multicast routing）の設定、変更、削除を行います。
ip monitor は、デバイスの状態、アドレス、ルートを継続的に監視します。

lnstat　Linux のネットワーク統計情報を提供します。 これはかつての rtstat プログラムを汎用的に機能充足を図ったプログラムです。

nstat　ネットワーク統計情報を表示します。

routef　ip route のコンポーネント。 これはルーティングテーブルをクリアします。

routel　ip route のコンポーネント。 これはルーティングテーブルの一覧を表示します。

rtacct　/proc/net/rt_acct の内容を表示します。

rtmon　ルート監視ユーティリティー。

rtpr　ip -o コマンドにより出力される内容を読みやすい形に戻します。

rtstat　ルートステータスの表示ユーティリティー。

ss　netstat コマンドと同じ。 アクティブな接続を表示します。

tc　トラフィック制御プログラム（Traffic Controlling Executable）。 これは QOS（Quality Of Service）と COS（Class Of Service）を実装するプログラムです。
tc qdisc は、キューイング規則（queueing discipline）の設定を行います。
tc class は、キューイング規則スケジューリング（queueing discipline scheduling）に基づくクラスの設定を行います。
tc estimator は、ネットワークフローを見積もります。
tc filter は、QOS/COS パケットのフィルタリング設定を行います。
tc policy は、QOS/COS ポリシーの設定を行います。

6.47. Kbd-1.15.2

Kbd パッケージは、キーテーブル（key-table）ファイルとキーボードユーティリティを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 16.0 MB

6.47.1. Kbd のインストール

バックスペース（backspace）キーとデリート（delete）キーは Kbd パッケージのキーマップ内では一貫した定義にはなっていません。 以下のパッチは i386 用のキーマップについてその問題を解消します。

```
patch -Np1 -i ../kbd-1.15.2-backspace-1.patch
```

パッチを当てればバックスペースキーの文字コードは 127 となり、デリートキーはよく知られたエスケープコードを生成することになります。

Kbd をコンパイルするための準備をします。

```
./configure --prefix=/usr --datadir=/lib/kbd
```

configure オプションの意味：

--datadir=/lib/kbd
このオプションによりキーボードレイアウトのデータを /usr/share/kbd ディレクトリではなく、ルートパーティションとなるようにします。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

注記

ベラルーシ語のような言語において Kbd パッケージは正しいキーマップを提供せず、 ISO-8859-5 エンコーディングで CP1251 キーマップであるものとして扱われます。 そのような言語ユーザーは個別に正しいキーマップをダウンロードして設定する必要があります。

LFS-ブートスクリプトパッケージの中には kbd_mode、 loadkeys、 openvt、 setfont に依存しているものがあります。 システム起動時の初期段階において /usr ディレクトリは参照不能であるため、上の実行モジュールはルートパーティションに置く必要があります。

```
mv -v /usr/bin/{kbd_mode,loadkeys,openvt,setfont} /bin
```

必要ならドキュメントをインストールします。

```
mkdir -v /usr/share/doc/kbd-1.15.2
cp -R -v doc/* \
        /usr/share/doc/kbd-1.15.2
```

6.47.2. Kbd の構成

インストールプログラム: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (psfxtable へのリンク), psfgettable (psfxtable へのリンク), psfstriptable (psfxtable へのリンク), psfxtable, resizecons, setfont, setkeycodes, setleds, setmetamode, showconsolefont, showkey, unicode_start, unicode_stop
インストールディレクトリ: /lib/kbd

概略説明

chvt 現在表示されている仮想端末を切り替えます。

deallocvt	未使用の仮想端末への割り当てを開放します。
dumpkeys	キーボード変換テーブル（keyboard translation table）の情報をダンプします。
fgconsole	アクティブな仮想端末数を表示します。
getkeycodes	カーネルのスキャンコード-キーコード（scancode-to-keycode）マッピングテーブルを表示します。
kbd_mode	キーボードモードの表示または設定を行います。
kbdrate	キーボードのリピート速度（repeat rate）と遅延時間（delay rate）を設定します。
loadkeys	キーボード変換テーブル（keyboard translation tables）をロードします。
loadunimap	カーネルのユニコード-フォント（unicode-to-font）マッピングテーブルをロードします。
mapscrn	かつてのプログラムです。 これはユーザー定義の文字マッピングテーブルをコンソールドライバーにロードするために利用します。 現在では setfont を利用します。
openvt	新しい仮想端末（virtual terminal; VT）上でプログラムを起動します。
psfaddtable	psfxtable へのリンク。
psfgettable	psfxtable へのリンク。
psfstriptable	psfxtable へのリンク。
psfxtable	コンソールフォント用のユニコード文字テーブルを取り扱います。
resizecons	カーネルが認識しているコンソールサイズを変更します。
setfont	EGA（Enhanced Graphic Adapter）フォントや VGA（Video Graphics Array）フォントを変更します。
setkeycodes	カーネルのスキャンコード-キーコード（scancode-to-keycode）マッピングテーブルの項目をロードします。 キーボード上に特殊キーがある場合に利用します。
setleds	キーボードフラグや LED（Light Emitting Diode）を設定します。
setmetamode	キーボードのメタキー（meta-key）設定を定義します。
showconsolefont	現在設定されている EGA/VGA コンソールスクリーンフォントを表示します。
showkey	キーボード上にて押下されたキーのスキャンコード、キーコード、ASCII コードを表示します。
unicode_start	キーボードとコンソールをユニコードモードにします。 キーマップファイルが ISO-8859-1 エンコーディングで書かれている場合にのみこれを利用します。 他のエンコーディングの場合、このプログラムの出力結果は正しいものになりません。
unicode_stop	キーボードとコンソールをユニコードモードから戻します。

6.48. Less-436

Less パッケージはテキストファイルビューアを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.9 MB

6.48.1. Less のインストール

Less をコンパイルするための準備をします。

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

configure オプションの意味：

--sysconfdir=/etc
本パッケージによって作成されるプログラムが /etc ディレクトリにある設定ファイルを参照するように指示します。

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

6.48.2. Less の構成

インストールプログラム: less, lessecho, lesskey

概略説明

less ファイルビューアまたはページャ。 指示されたファイルの内容を表示します。 表示中にはスクロール、文字検索、移動が可能です。

lessecho Unix システム上のファイル名において * や ? といったメタ文字（meta-characters）を展開するために必要となります。

lesskey less におけるキー割り当てを設定するために利用します。

6.49. Make-3.82

Make パッケージは、 パッケージ類をコンパイルするためのプログラムを提供します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 9.7 MB

6.49.1. Make のインストール

Make をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.49.2. Make の構成

インストールプログラム: make

概略説明

make パッケージの構成要素に対して、どれを(再)コンパイルするかを自動判別し、対応するコマンドを実行します。

6.50. Man-DB-2.5.7

Man-DB パッケージは man ページを検索したり表示したりするプログラムを提供します。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 22 MB

6.50.1. Man-DB のインストール

特定の man ページを表示する際に発生する問題を修正するために、以下のパッチを適用します。

```
patch -Np1 -i ../man-db-2.5.7-fix_man_assertion-1.patch
```

Man-DB をコンパイルするための準備をします。

```
./configure --prefix=/usr --libexecdir=/usr/lib \
    --docdir=/usr/share/doc/man-db-2.5.7 --sysconfdir=/etc --disable-setuid \
    --with-browser=/usr/bin/lynx --with-vgrind=/usr/bin/vgrind \
    --with-grap=/usr/bin/grap
```

configure オプションの意味

--disable-setuid
これは man プログラムが man ユーザーに対して setuid を実行しないようにします。

--with-...
この三つのオプションはデフォルトで利用するプログラムを指定します。 lynx はテキストベースの Web ブラウザです。（BLFS でのインストール手順を参照してください。）vgrind はプログラムソースを Groff の入力形式に変換します。 grap は Groff 文書においてグラフを組版するために利用します。 vgrind と grap は man ページを見るだけであれば必要ありません。 これらは LFS や BLFS には含まれません。 もし利用したい場合は LFS の構築を終えた後に自分でインストールしてください。

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには、以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.50.2. LFS における英語以外のマニュアルページ

以下に示す表は /usr/share/man/<ll> 配下にインストールされる man ページとそのエンコーディングを示します。
Man-DB は man ページが UTF-8 エンコーディングかどうかを正しく認識します。

表 6.1. 8 ビット man ページのキャラクタエンコーディング

言語（コード）	エンコーディング	言語（コード）	エンコーディング
デンマーク語（da）	ISO-8859-1	クロアチア語（hr）	ISO-8859-2
ドイツ語（de）	ISO-8859-1	ハンガリー語（hu）	ISO-8859-2
英語（en）	ISO-8859-1	日本語（ja）	EUC-JP
スペイン語（es）	ISO-8859-1	韓国語（ko）	EUC-KR
エストニア語（et）	ISO-8859-1	リトアニア語（lt）	ISO-8859-13
フィンランド語（fi）	ISO-8859-1	ラトビア語（lv）	ISO-8859-13
フランス語（fr）	ISO-8859-1	マケドニア語（mk）	ISO-8859-5
アイルランド語（ga）	ISO-8859-1	ポーランド語（pl）	ISO-8859-2
ガリシア語（gl）	ISO-8859-1	ルーマニア語（ro）	ISO-8859-2
インドネシア語（id）	ISO-8859-1	ロシア語（ru）	KOI8-R
アイスランド語（is）	ISO-8859-1	スロバキア語（sk）	ISO-8859-2
イタリア語（it）	ISO-8859-1	スロベニア語（sl）	ISO-8859-2
ノルウェー語 ブークモール（Norwegian Bokmal; nb）	ISO-8859-1	セルビア Latin（sr@latin）	ISO-8859-2
オランダ語（nl）	ISO-8859-1	セルビア語（sr）	ISO-8859-5
ノルウェー語 ニーノシュク（Norwegian Nynorsk; nn）	ISO-8859-1	トルコ語（tr）	ISO-8859-9
ノルウェー語（no）	ISO-8859-1	ウクライナ語（uk）	KOI8-U
ポルトガル語（pt）	ISO-8859-1	ベトナム語（vi）	TCVN5712-1
スウェーデン語（sv）	ISO-8859-1	中国語 簡体字（Simplified Chinese）（zh_CN）	GBK
ベラルーシ語（be）	CP1251	中国語 簡体字（Simplified Chinese），シンガポール（zh_SG）	GBK
ブルガリア語（bg）	CP1251	中国語 繁体字（Traditional Chinese），香港（zh_HK）	BIG5HKSCS
チェコ語（cs）	ISO-8859-2	中国語 繁体字（Traditional Chinese）（zh_TW）	BIG5
ギリシア語（el）	ISO-8859-7		

注記

上に示されていない言語によるマニュアルページはサポートされません。

6.50.3. Man-DB の構成

インストールプログラム:	accessdb, apropos（whatis へのリンク）, catman, lexgrog, man, mandb, manpath, whatis, zsoelim
インストールディレクトリ:	/usr/lib/man-db, /usr/share/doc/man-db

概略説明

accessdb	whatis データベースの内容をダンプして読みやすい形で出力します。
apropos	whatis データベースを検索して、指定した文字列を含むシステムコマンドの概略説明を表示します。
catman	フォーマット済マニュアルページを生成、更新します。
lexgrog	指定されたマニュアルページについて、一行のサマリー情報を表示します。

man	指定されたマニュアルページを整形して表示します。
mandb	whatis データベースを生成、更新します。
manpath	$MANPATH の内容を表示します。 あるいは（$MANPATH が設定されていない場合は）man.conf 内の設定とユーザー設定に基づいて適切な検索パスを表示します。
whatis	whatis データベースを検索して、指定されたキーワードを含むシステムコマンドの概略説明を表示します。
zsoelim	ファイルの内容を読み込んで、 .so file の形で書かれている記述行を、その file の内容に置き換えます。

6.51. Module-Init-Tools-3.12

Module-Init-Tools パッケージは、Linux カーネル 2.5.47 以上においてカーネルモジュールを扱うプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 8.6 MB

6.51.1. Module-Init-Tools のインストール

Man ページの生成が不要であるにも関わらず再生成してしまう不備を修正するために、適切な他の Man ページを指し示すような修正を行います。

```
echo '.so man5/modprobe.conf.5' > modprobe.d.5
```

本パッケージのテストスイートは開発者の必要を満たす目的で構築されています。 make check を実行すると、特別な形で modprobe プログラムがビルドされます。 しかしこれは普通に用いるには無意味なものです。 テストスイートを実行するなら（約 0.2 SBU）以下のコマンドを実行します。（make clean コマンドはソースツリーをきれいなものとするために必要で、次に再コンパイルして通常利用するプログラムをビルドします。）

```
./configure
make check
./tests/runtests
make clean
```

Module-Init-Tools をコンパイルするための準備をします。

```
./configure --prefix=/ --enable-zlib-dynamic --mandir=/usr/share/man
```

パッケージをコンパイルします。

```
make
```

パッケージをインストールします。

```
make INSTALL=install install
```

make パラメータの意味：

INSTALL=install
インストールする実行ファイル類が既に存在している場合、普通 make install を実行しただけではそれらをインストールしません。 このオプションを指定することでその動作を変更します。 つまりデフォルトのインストールラッパースクリプトを用いるのではなく install コマンドを用いるようにします。

6.51.2. Module-Init-Tools の構成

インストールプログラム: depmod, insmod, insmod.static, lsmod, modinfo, modprobe, rmmod

概略説明

depmod	存在しているモジュール内に含まれるシンボル名に基づいて、モジュールの依存関係を記述したファイル（dependency file）を生成します。 これは modprobe が、必要なモジュールを自動的にロードするために利用します。
insmod	稼動中のカーネルに対してロード可能なモジュールをインストールします。
insmod.static	スタティックライブラリによってコンパイルされた insmod コマンド。
lsmod	その時点でロードされているモジュールを一覧表示します。
modinfo	カーネルモジュールに関連付いたオブジェクトファイルを調べて、出来る限りの情報を表示します。
modprobe	depmod によってモジュールの依存関係を記述したファイル（dependency file）が生成されます。これを使って関連するモジュールを自動的にロードします。
rmmod	稼動中のカーネルからモジュールをアンロードします。

6.52. Patch-2.6.1

Patch パッケージは「パッチ」ファイルを適用することにより、ファイルの修正・生成を行うプログラムを提供します。 「パッチ」ファイルは diff プログラムにより生成されます。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 1.9 MB

6.52.1. Patch のインストール

ed プログラムを必要とするテストスイートの実行を行わないよう、パッチを適用します。

```
patch -Np1 -i ../patch-2.6.1-test_fix-1.patch
```

Patch をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

6.52.2. Patch の構成

インストールプログラム: patch

概略説明

patch パッチファイルに従って対象ファイルを修正します。 パッチファイルは通常 diff コマンドによって修正前後の違いが列記されているものです。 そのような違いを対象ファイルに適用することで patch はパッチを適用したファイルを生成します。

6.53. Psmisc-22.12

Psmisc パッケージは稼動中プロセスの情報表示を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.5 MB

6.53.1. Psmisc のインストール

peekfd が x86_64 アーキテクチャではビルドできないバグを修正します。

```
sed -i 's@#include <sys\/user.h>@#include <bits\/types.h>\n&@' configure
```

Psmisc をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

killall プログラムと fuser プログラムを、FHS が規定しているディレクトリに移動します。

```
mv -v /usr/bin/fuser /bin
mv -v /usr/bin/killall /bin
```

6.53.2. Psmisc の構成

インストールプログラム: fuser, killall, peekfd, prtstat, pstree, pstree.x11 (pstree へのリンク)

概略説明

fuser	指定されたファイルまたはファイルシステムを利用しているプロセスのプロセス ID (PID) を表示します。
killall	プロセス名を用いてそのプロセスを終了 (kill) させます。 指定されたコマンドを起動しているすべてのプロセスに対してシグナルが送信されます。
peekfd	PID を指定することによって、稼動中のそのプロセスのファイルディスクリプタを調べます。
prtstat	プロセスに関する情報を表示します。
pstree	稼動中のプロセスをツリー形式で表示します。
pstree.x11	pstree と同じです。 ただし終了時には確認画面が表示されます。

6.54. Shadow-4.1.4.2

Shadow パッケージはセキュアなパスワード管理を行うプログラムを提供します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 30 MB

6.54.1. Shadow のインストール

注記

もっと強力なパスワードを利用したい場合は http://www.linuxfromscratch.org/blfs/view/svn/postlfs/cracklib.html にて示している Cracklib パッケージを参照してください。 Cracklib パッケージは Shadow パッケージよりも前にインストールします。 その場合 Shadow パッケージの configure スクリプトでは **--with-libcrack** パラメータをつけて実行する必要があります。

groups コマンドとその man ページをインストールしないようにします。 これは Coreutils パッケージにて、より良いバージョンが提供されているからです。

```
sed -i 's/groups$(EXEEXT) //' src/Makefile.in
find man -name Makefile.in -exec sed -i 's/groups\.1 / /' {} \;
```

中国語と韓国語の man ページはインストールしません。 Man-DB がそれらを正しく扱うことができないためです。

```
sed -i -e 's/ ko//' -e 's/ zh_CN zh_TW//' man/Makefile.in
```

パスワード暗号化に関して、デフォルトの crypt 手法ではなく、より強力な MD5 手法を用いることにします。 こうしておくと 8文字以上のパスワード入力が可能となります。 またメールボックスを収めるディレクトリとして Shadow ではデフォルトで **/var/spool/mail** ディレクトリを利用していますが、これは古いものであるため **/var/mail** ディレクトリに変更します。

```
sed -i -e 's@#ENCRYPT_METHOD DES@ENCRYPT_METHOD MD5@' \
       -e 's@/var/spool/mail@/var/mail@' etc/login.defs
```

注記

Cracklib のサポートを含めて Shadow をビルドする場合は以下を実行します。

```
sed -i 's@DICTPATH.*@DICTPATH\t/lib/cracklib/pw_dict@' \
    etc/login.defs
```

Shadow をコンパイルするための準備をします。

```
./configure --sysconfdir=/etc
```

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make install
```

不適切なディレクトリにインストールされるプログラムを移動させます。

```
mv -v /usr/bin/passwd /bin
```

6.54.2. Shadow の構成

このパッケージには、ユーザーやグループの追加・修正・削除、そのパスワードの設定・変更、その他の管理操作を行うユーティリティが含まれます。 パスワードのシャドウイング（password shadowing）というものが何を意味するのか、その詳細についてはこのパッケージのソース内にある **doc/HOWTO** を参照してください。 Shadow によるサポート

を利用する場合、パスワード認証を必要とするプログラム（ディスプレイマネージャ、FTP プログラム、POP3、デーモン、など）は Shadow に準拠したものでなければなりません。 つまりそれらのプログラムが、シャドウ化された（shadowed）パスワードを受け入れて動作しなければならないということです。

Shadow によるパスワードの利用を有効にするために、以下のコマンドを実行します。

```
pwconv
```

また Shadow によるグループパスワードを有効にするために、以下を実行します。

```
grpconv
```

Shadow の useradd コマンドに対する通常の設定には、注意すべき点があります。 まず useradd コマンドによりユーザーを生成する場合のデフォルトの動作では、ユーザー名と同じグループを自動生成します。 ユーザーID（UID）とグループID（GID）は 1000 以上が割り当てられます。 useradd コマンドの利用時に特にパラメータを与えなければ、追加するユーザーのグループは新たな固有グループが生成されることになります。 この動作が不適当であれば useradd コマンドの実行時に -g パラメータを利用することが必要です。 デフォルトで適用されるパラメータは /etc/default/useradd ファイルに定義されています。 このファイルのパラメータ定義を必要に応じて書き換えてください。

/etc/default/useradd のパラメータ説明

GROUP=1000
: このパラメータは /etc/group ファイルにて設定されるグループIDの先頭番号を指定します。 必要なら任意の数値に設定することもできます。 useradd コマンドは既存の UID 値、GID 値を再利用することはありません。 このパラメータによって定義された数値が実際に指定された場合、この値以降で利用可能な値が利用されます。 また useradd コマンドの実行時に、パラメータ -g を利用せず、かつグループID 1000 のグループが存在していなかった場合は、以下のようなメッセージが出力されます。 useradd: unknown GID 1000（"GID 1000 が不明です"）このメッセージは無視することができます。 この場合グループIDには 1000 が利用されます。

CREATE_MAIL_SPOOL=yes
: このパラメータは useradd コマンドの実行によって、追加されるユーザー用のメールボックスに関するファイルが生成されます。 useradd コマンドは、このファイルのグループ所有者を mail（グループID 0660）に設定します。メールボックスに関するファイルを生成したくない場合は、以下のコマンドを実行します。

```
sed -i 's/yes/no/' /etc/default/useradd
```

6.54.3. root パスワードの設定

root ユーザーのパスワードを設定します。

```
passwd root
```

6.54.4. Shadow の構成

インストールプログラム:	chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmems, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgrp, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg（newgrp へのリンク）, su, useradd, userdel, usermod, vigr（vipw へのリンク）, vipw
インストールディレクトリ:	/etc/default

概略説明

chage	ユーザーのパスワード変更を行うべき期間を変更します。
chfn	ユーザーのフルネームや他の情報を変更します。
chgpasswd	グループのパスワードをバッチモードにて更新します。
chpasswd	ユーザーのパスワードをバッチモードにて更新します。
chsh	ユーザーのデフォルトのログインシェルを変更します。
expiry	現時点でのパスワード失効に関する設定をチェックし更新します。
faillog	ログイン失敗のログを調査します。 ログインの失敗を繰り返すことでアカウントがロックされる際の、最大の失敗回数を設定します。 またその失敗回数をリセットします。
gpasswd	グループに対してメンバーや管理者を追加・削除します。
groupadd	指定した名前でグループを生成します。

groupdel	指定された名前のグループを削除します。
groupmems	スーパーユーザー権限を持たなくても、自分自身のグループのメンバーリストを管理可能とします。
groupmod	指定されたグループの名前や GID を修正します。
grpck	グループファイル /etc/group と /etc/gshadow の整合性を確認します。
grpconv	通常のグループファイルから Shadow グループファイルを生成・更新します。
grpunconv	/etc/gshadow ファイルを元に /etc/group ファイルを更新し /etc/gshadow ファイルを削除します。
lastlog	全ユーザーの中での最新ログインの情報、または指定ユーザーの最新ログインの情報を表示します。
login	ユーザーのログインを行います。
logoutd	ログオン時間とポートに対する制限を実施するためのデーモン。
newgrp	ログインセッション中に現在の GID を変更します。
newusers	ユーザーアカウントの情報を生成または更新します。
nologin	ユーザーアカウントが利用不能であることをメッセージ表示します。 利用不能なユーザーアカウントに対するデフォルトシェルとして利用することを意図しています。
passwd	ユーザーアカウントまたはグループアカウントに対するパスワードを変更します。
pwck	パスワードファイル /etc/passwd と /etc/shadow の整合性を確認します。
pwconv	通常のパスワードファイルを元に shadow パスワードファイルを生成・更新します。
pwunconv	/etc/shadow ファイルを元に /etc/passwd ファイルを更新し /etc/shadow を削除します。
sg	ユーザーの GID を指定されたグループにセットした上で、指定されたコマンドを実行します。
su	ユーザー ID とグループ ID を変更してシェルを実行します。
useradd	指定した名前で新たなユーザーを生成します。 あるいは新規ユーザーのデフォルトの情報を更新します。
userdel	指定されたユーザーアカウントを削除します。
usermod	指定されたユーザーのログイン名、UID (User Identification)、利用シェル、初期グループ、ホームディレクトリなどを変更します。
vigr	/etc/group ファイル、あるいは /etc/gshadow ファイルを編集します。
vipw	/etc/passwd ファイル、あるいは /etc/shadow ファイルを編集します。

6.55. Sysklogd-1.5

Sysklogd パッケージは、例えばカーネルが異常発生時に出力するログのような、システムログメッセージを取り扱うプログラムを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 0.5 MB

6.55.1. Sysklogd のインストール

パッケージをコンパイルします。

```
make
```

このパッケージにテストスイートはありません。

パッケージをインストールします。

```
make BINDIR=/sbin install
```

6.55.2. Sysklogd の設定

以下を実行して /etc/syslog.conf ファイルを生成します。

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

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

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

6.55.3. Sysklogd の構成

インストールプログラム: klogd, syslogd

概略説明

klogd カーネルメッセージを受け取り出力するシステムデーモン。

syslogd システムプログラムが出力するログ情報を出力します。 出力されるログ情報には少なくとも処理日付、ホスト名が出力されます。 また通常はプログラム名も出力されます。 ただこれはログ出力デーモンがどれだけ信頼のおけるものであるかに依存する情報です。

6.56. Sysvinit-2.88dsf

Sysvinit パッケージは、システムの起動、実行、シャットダウンを制御するプログラムを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 1 MB

6.56.1. Sysvinit のインストール

ランレベルが変更される時（例えばシステムが停止する時）init コマンドは各種のプロセスに対して停止シグナル（termination signals）を送信します。 ただしその対象は init プログラム自身が起動したプロセスであり、新たなランレベルでは起動しないプロセスです。 一方で init コマンドが出力するメッセージは 「Sending processes the TERM signal」（プロセスに対して TERM シグナルを送信します）というものです。 このメッセージは、その時点にて稼動中の全プロセスに対してシグナルを送信しているかのように誤解してしまいます。 これを正すためにソースを修正して 「Sending processes configured via /etc/inittab the TERM signal」（/etc/inittab で設定されているプロセスに対して TERM シグナルを送信します）というメッセージに置き換えます。

```
sed -i 's@Sending processes@& configured via /etc/inittab@g' \
    src/init.c
```

wall コマンドは Util-linux-ng パッケージにおいてメンテナンスされており、既にインストールが出来ています。 そこで Sysvinit が提供する wall コマンドはインストールせず、その man ページもインストールしないようにします。

```
sed -i -e 's/utmpdump wall/utmpdump/' \
       -e 's/mountpoint.1 wall.1/mountpoint.1/' src/Makefile
```

パッケージをコンパイルします。

```
make -C src
```

本パッケージにテストスイートはありません。

パッケージをインストールします。

```
make -C src install
```

6.56.2. Sysvinit の設定

以下のコマンドを実行して新しい /etc/inittab ファイルを生成します。

```
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
```

6.56.3. Sysvinit の構成

インストールプログラム: bootlogd, fstab-decode, halt, init, killall5, last, lastb (last へのリンク), mesg, mountpoint, pidof (killall5 へのリンク), poweroff (halt へのリンク), reboot (halt へのリンク), runlevel, shutdown, sulogin, telinit (init へのリンク), utmpdump

概略説明

bootlogd ブート時のメッセージをログファイルに出力します。

fstab-decode fstab 形式の (fstab-encoded の) 引数とともにコマンドを実行します。

halt ランレベルが既に 0 ではない通常の起動状態の場合に shutdown をオプション -h をつけて実行します。 そしてカーネルに対してシステム停止を指示します。 システムが停止される状況は /var/log/wtmp ファイルに記録されます。

init カーネルがハードウェアを初期化した後に、最初に起動するプロセスです。 ブート処理がこのプロセスに引き継がれ、指示されたプロセスをすべて起動していきます。

killall5 プロセスすべてに対してシグナルを送信します。 ただし自分のセッション内の起動プロセスは除きます。 つまり本コマンドを実行したスクリプトは停止されません。

last ユーザーの最新のログイン (ログアウト) の情報を表示します。 これは /var/log/wtmp ファイルの終わりから調べているものです。 またシステムブート、シャットダウン、ランレベルの変更時の情報も示します。

lastb ログインに失敗した情報を表示します。 これは /var/log/btmp に記録されています。

mesg 現在のユーザーの端末に対して、他のユーザーがメッセージ送信できるかどうかを制御します。

mountpoint 指定されたディレクトリがマウントポイントであるかどうかをチェックします。

pidof 指定されたプログラムの PID を表示します。

poweroff カーネルに対してシステムの停止を指示し、コンピュータの電源を切ります。 (halt を参照してください。)

reboot	カーネルに対してシステムの再起動を指示します。(halt を参照してください。)
runlevel	現在のランレベルと直前のランレベルを表示します。 最新のランレベルは /var/run/utmp ファイルに記録されています。
shutdown	システムの終了を安全に行います。 その際にはプロセスすべてへのシグナル送信を行い、ログインユーザーへの通知も行います。
sulogin	root ユーザーでのログインを行います。 通常は init が起動するもので、システムがシングルユーザーモードで起動する際に利用されます。
telinit	init に対してランレベルの変更を指示します。
utmpdump	指定されたログファイル内の情報を分かりやすく表示します。

6.57. Tar-1.23

Tar パッケージはアーカイブプログラムを提供します。

概算ビルド時間: 1.9 SBU
必要ディスク容量: 21.2 MB

6.57.1. Tar のインストール

最新ソースにて発生するバグを修正します。

```
sed -i /SIGPIPE/d src/tar.c
```

新たな tar ファイルを生成する際に、バッファオーバーフローが発生するバグを修正します。 この修正は gcc-4.5 以上においてパッケージをビルドする際に必要となります。

```
patch -Np1 -i ../tar-1.23-overflow_fix-1.patch
```

Tar をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするために以下を実行します。(約 1 SBU)

```
sed -i '35 i\
AT_UNPRIVILEGED_PREREQ' tests/remfiles01.at
make check
```

パッケージをインストールします。

```
make install
```

6.57.2. Tar の構成

インストールプログラム: rmt, tar

概略説明

rmt プロセス間通信のコネクションを通じて磁気テープドライブを遠隔操作します。

tar アーカイブの生成、アーカイブからのファイル抽出、アーカイブの内容一覧表示を行います。 アーカイブは tarball とも呼ばれます。

6.58. Texinfo-4.13a

Texinfo パッケージは info ページへの読み書き・変換を行うプログラムを提供します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 21 MB

6.58.1. Texinfo のインストール

Texinfo をコンパイルするための準備をします。

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

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

必要なら TeX システムに属するコンポーネント類をインストールします。

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

make パラメータの意味：

TEXMF=/usr/share/texmf
Makefile 変数である TEXMF に TeX ツリーのルートディレクトリを設定します。 これは後に TeX パッケージをインストールするための準備です。

ドキュメントシステム Info は、 メニュー項目の一覧を単純なテキストファイルに保持しています。 そのファイルは /usr/share/info/dir にあります。 残念ながら数々のパッケージの Makefile は、既にインストールされている info ページとの同期を取る処理を行わない場合があります。 /usr/share/info/dir の再生成を必要とするなら、以下のコマンドを実行してこれを実現します。

```
cd /usr/share/info
rm -v dir
for f in *
do install-info $f dir 2>/dev/null
done
```

6.58.2. Texinfo の構成

インストールプログラム: info, infokey, install-info, makeinfo, pdftexi2dvi, texi2dvi, texi2pdf, texindex
インストールディレクトリ: /usr/share/texinfo

概略説明

info	info ページを見るために利用します。 これは man ページに似ていますが、単に利用可能なコマンドラインオプションを説明するだけのものではなく、おそらくはもっと充実しています。 例えば man bison と info bison を比較してみてください。
infokey	Info のカスタマイズ情報を設定したソースファイルをバイナリ形式にコンパイルします。
install-info	info ページをインストールします。 info 索引ファイルにある索引項目も更新します。
makeinfo	指定された Texinfo ソースファイルを Info ページ、プレーンテキスト、HTML ファイルに変換します。
pdftexi2dvi	指定された Texinfo ドキュメントファイルを PDF (Portable Document Format) ファイルに変換します。

texi2dvi	指定された Texinfo ドキュメントファイルを、デバイスに依存しない印刷可能なファイルに変換します。
texi2pdf	指定された Texinfo ドキュメントファイルを PDF（Portable Document Format）ファイルに変換します。
texindex	Texinfo 索引ファイルの並び替えを行います。

6.59. Udev-161

Udev パッケージはデバイスノードを動的に生成するプログラムを提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 9.3 MB 、また testfiles については 37 MB

6.59.1. Udev のインストール

udev-config という Tar アーカイブファイルには Udev パッケージをビルドする際の LFS 独自のファイルが含まれています。 以下のようにしてこのファイルを Udev ソースディレクトリに展開します。

```
tar -xvf ../udev-config-20100128.tar.bz2
```

udev-testfiles という Tar アーカイブファイルには Udev のテストに必要なファイルが含まれます。 このアーカイブファイル内には、見かけ上 37MB のファイルが含まれますが、実際に要するディスク容量は 7MB 以下です。

```
tar -xvf ../udev-161-testfiles.tar.bz2 --strip-components=1
```

デバイスやディレクトリのいくつかはシステム起動時に必要になりますが、起動処理の初期段階であるために Udev はそれらを認識できません。 そこでそれらのデバイスまたはディレクトリを生成します。

```
install -dv /lib/{firmware,udev/devices/{pts,shm}}
mknod -m0666 /lib/udev/devices/null c 1 3
```

パッケージをコンパイルするための準備をします。

```
./configure --prefix=/usr \
    --sysconfdir=/etc --sbindir=/sbin \
    --with-rootlibdir=/lib --libexecdir=/lib/udev \
    --disable-extras --disable-introspection
```

configure オプションの意味：

--with-rootlibdir=/lib
このオプションは libudev ライブラリのインストール先を指定します。 このライブラリは /lib ディレクトリにインストールする必要があります。 デフォルトでは --rootlibdir は /usr/lib ディレクトリとなっていますが、/usr ディレクトリが認識できるようになる前の、ブート起動時に Udev が認識できなければならないためです。

--libexecdir=/lib/udev
このオプションは Udev の内部ルールやヘルパープログラムのインストール先を指定します。

--disable-extras
このオプションは、ヘルパープログラムやその他の追加プログラムをインストールしないことを指定します。 追加プログラムには、さらに外部ライブラリが必要となり、それらは LFS ベースシステムでは取り扱っていません。 詳しくは Udev の README ファイルを参照してください。

--disable-introspection
このオプションは Udev のイントロスペクション（introspection）機能を無効にします。 この機能は、LFS システムにてインストールするパッケージではなく、別のパッケージにて必要となるものです。 詳しくは Udev の README ファイルを参照してください。

パッケージをコンパイルします。

```
make
```

本パッケージのテストを実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

空のドキュメントディレクトリを削除します。

```
rmdir -v /usr/share/doc/udev
```

LFS 固有のカスタムルールファイルをインストールします。

```
cd udev-config-20100128
make install
```

LFS 固有のカスタムルールファイルについて説明しているドキュメントをインストールします。

```
make install-doc
```

6.59.2. Udev の構成

インストールプログラム:	ata_id, cdrom_id, collect, create_floppy_devices, edd_id, firmware.sh, fstab_import, path_id, scsi_id, udevadm, udevd, usb_id, write_cd_rules, write_net_rules
インストールライブラリ:	libudev.{a,so}
インストールディレクトリ:	/etc/udev, /lib/udev, /lib/firmware

概略説明

ata_id	ATA ドライブに対するユニークな文字列と追加情報（uuid、ラベル）を Udev に提供します。
cdrom_id	CD-ROM ドライブや DVD-ROM ドライブの情報を Udev に提供します。
collect	現在の uevent の ID と（すべての対象 uevent に対する）ID のリストを与えることで、現在の ID を登録し、すべての対象 ID が既に登録済みであるかどうかを示します。
create_floppy_devices	CMOS タイプに基づく、すべてのフロッピーデバイスを生成します。
edd_id	BIOS ディスクドライブに対する EDD ID を Udev に提供する。
firmware.sh	ファームウェアをデバイスにアップロードします。
fstab_import	/etc/fstab に記述された項目の中から現在のデバイスに合致するものを探し出し、その情報を Udev に提供します。
path_id	デバイスへのパスとして、可能な限り最も短くユニークなハードウェアパスを提供します。
scsi_id	特定のデバイスに対する SCSI INQUIRY コマンド送信の結果として得られるデータに基づく、ユニークな SCSI 識別子を Udev に対して提供します。
udevadm	汎用的な Udev 管理ツール。 udevd デーモンの制御、Udev データベースデータの提供、uevent の監視、uevent の完了までの待機、Udev 設定のテスト、指定デバイスに対する uevent の起動、といったことを行います。
udevd	ネットワークソケット上の uevent を待ち受けるデーモン。 デバイスを生成し、その uevent に対応する外部プログラムを起動します。
usb_id	USB デバイスに関する情報を Udev に対して提供します。
write_cd_rules	光学ドライブに対する固定的な名称を定めた Udev ルールを生成するためのスクリプト。（7.10. 「デバイスへのシンボリックリンクの生成」 も参照のこと。）
write_net_rules	ネットワークインターフェースに対する固定的な名称を定めた Udev ルールを生成するためのスクリプト。（7.13. 「ネットワークスクリプトの設定」 も参照のこと。）
libudev	Udev デバイス情報のインターフェースライブラリ。
/etc/udev	Udev 設定ファイル、デバイスのパーミッション、デバイス命名規則を定めます。

6.60. Vim-7.3

Vim パッケージは強力なテキストエディタを提供します。

概算ビルド時間: 1.0 SBU
必要ディスク容量: 87 MB

Vim の代替ソフトウェア

もし Emacs、Joe、Nano など他のエディタを用いたい場合は http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html に示される手順に従ってインストールしてください。

6.60.1. Vim のインストール

設定ファイル vimrc がインストールされるデフォルトディレクトリを /etc に変更します。

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

Vim をコンパイルするための準備をします。

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

configure オプションの意味：

--enable-multibyte
このオプションは、マルチバイトエンコーディングによるファイルの編集をサポートする指示を行います。 マルチバイト文字を用いるロケールにとってはこれが必要です。 例えば Fedora Core のようにデフォルトで UTF-8 を採用している Linux ディストリビューションにおいては、新規に生成するテキストファイルを編集できるようにするために、このオプションを指定することが有用です。

パッケージをコンパイルします。

```
make
```

コンパイル結果をテストするには以下を実行します。

```
make test
```

このテストスイートは数多くのバイナリデータを端末画面上に出力します。 これは端末画面の設定によっては問題を引き起こします。 これを避けるには出力をリダイレクトしてログファイルに出力するようにしてください。

パッケージをインストールします。

```
make install
```

たいていのユーザーは vim ではなく vi を使うようです。 vi を入力しても vim が実行されるように、実行モジュールに対するシンボリックリンクを作成します。 さらに指定された言語による man ページへのシンボリックリンクも作成します。

```
ln -sv vim /usr/bin/vi
for L in  /usr/share/man/{,*/}man1/vim.1; do
    ln -sv vim.1 $(dirname $L)/vi.1
done
```

デフォルトでは Vim のドキュメントが /usr/share/vim にインストールされます。 以下のようなシンボリックリンクを生成することで /usr/share/doc/vim-7.3 へアクセスしてもドキュメントが参照できるようにし、他のパッケージが配置するドキュメントの場所と整合を取ります。

```
ln -sv ../vim/vim73/doc /usr/share/doc/vim-7.3
```

LFS システムに対して X ウィンドウシステムをインストールする場合 X のインストールの後で Vim を再コンパイルする必要があります。 Vim には GUI 版があり X や他のライブラリがインストールされていて 初めて構築できるためです。 この作業の詳細については Vim のドキュメントと BLFS ブックの http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim に示されている Vim のインストール説明のページを参照してください。

6.60.2. Vim の設定

デフォルトで vim は Vi 非互換モード（vi-incompatible mode）で起動します。 他のエディタを使ってきたユーザーにとっては、よく分からないものかもしれません。 以下の設定における 「nocompatible」（非互換）は、Vi の新しい機能を利用することを意味しています。 もし 「compatible」（互換）モードに変更したい場合は、この設定ファイルの冒頭にて行っておくことが必要です。 このモード設定は他の設定を置き換えるものとなることから、まず初めに行っておかなければならないものだからです。 以下のコマンドを実行して vim の設定ファイルを生成します。

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

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

" End /etc/vimrc
EOF
```

set nocompatible と設定しておくと vi 互換モードでの動作に比べて有用な動作となります。（これがデフォルトになっています。）その設定の記述から 「no」 の文字を取り除けば、旧来の vi コマンドの動作となります。 set backspace=2 を設定しておくと、行を超えてもバックスペースキーによる編集が可能となります。 またインデントが自動的に行われ、コマンド起動時には自動的に挿入モードとなります。 syntax on パラメータを指定すれば vim の文法ハイライト（syntax highlighting）機能が有効になります。 最後にある if 文は、set background=dark を指定した場合に、特定の端末エミュレータ上において vim が背景色を誤って認識しないようにするためのものです。 エミュレータの背景色が黒色であった場合に、より適切なハイライトが実現できます。

この他に利用できるオプションについては、以下のコマンドを実行することで出力される説明を参照してください。

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

注記

Vim がインストールするスペルファイル（spell files）はデフォルトでは英語に対するものだけです。 必要とする言語のスペルファイルをインストールするなら ftp://ftp.vim.org/pub/vim/runtime/spell/ から、特定の言語、エンコーディングによる *.spl ファイル、またオプションとして *.sug ファイルをダウンロードしてください。 そしてそれらのファイルを /usr/share/vim/vim73/spell/ ディレクトリに保存してください。

スペルファイルを利用するには /etc/vimrc ファイルにて、例えば以下のような設定が必要になります。

```
set spelllang=en,ru
set spell
```

詳しくは、上で説明した URL にて提供されている README ファイルを参照してください。

6.60.3. Vim の構成

インストールプログラム:	ex (vim へのリンク), rview (vim へのリンク), rvim (vim へのリンク), vi (vim へのリンク), view (vim へのリンク), vim, vimdiff (vim へのリンク), vimtutor, xxd
インストールディレクトリ:	/usr/share/vim

概略説明

ex	vim を ex モードで起動します。
rview	view の機能限定版。 シェルは起動できず、サスペンドも行うことはできません。
rvim	vim の機能限定版。 シェルは起動できず、サスペンドも行うことはできません。
vi	vim へのリンク。
view	vim を読み込み専用モード（read-only mode）で起動します。

vim	エディタ。
vimdiff	vim により、同一ファイルにおける２つまたは３つの版を同時に編集し、差異を表示します。
vimtutor	vim の基本的なキー操作とコマンドについて教えてくれます。
xxd	指定されたファイルの内容を 16進数ダンプとして変換します。 逆の変換も行うことができるため、バイナリパッチにも利用されます。

6.61. デバッグシンボルについて

プログラムやライブラリの多くは、デフォルトではデバッグシンボルを含めてコンパイルされています。（gcc の -g オプションが用いられています。）デバッグ情報を含めてコンパイルされたプログラムやライブラリは、デバッグ時にメモリアドレスが参照できるだけでなく、処理ルーチンや変数の名称も知ることができます。

しかしそういったデバッグ情報は、プログラムやライブラリのファイルサイズを極端に大きくします。 以下にデバッグシンボルが占める割合の例を示します。

- デバッグシンボルを含んだ bash の実行ファイル： 1200 KB
- デバッグシンボルを含まない bash の実行ファイル： 480 KB
- デバッグシンボルを含んだ Glibc と GCC の関連ファイル（/lib と /usr/lib）: 87 MB
- デバッグシンボルを含まない Glibc と GCC の関連ファイル: 16MB

利用するコンパイラや C ライブラリの違いによって、生成されるファイルのサイズは異なります。 デバッグシンボルを含む、あるいは含まないサイズを比較した場合、その差は 2倍から 5倍の違いがあります。

プログラムをデバッグするユーザーはそう多くはありません。 デバッグシンボルを削除すればディスク容量はかなり節減できます。 次節ではプログラムやライブラリからデバッグシンボルを取り除く（strip する）方法を示します。

6.62. 再度のストリップ

対象ユーザーがプログラマではなく、プログラム類をデバッグするような使い方をしないのであれば、実行ファイルやライブラリに含まれるデバッグシンボルを削除しても構いません。 そうすれば 90 MB ものサイズ削減を図ることができます。 たとえデバッグできなくなっても困らないはずです。

以下に示すコマンドは、いとも簡単なものです。 ただし入力つづりは簡単に間違いやすいので、もし誤った入力をするとシステムを利用不能にしてしまいます。 したがって strip コマンドを実行する前に、現時点の LFS システムのバックアップを取っておくことをお勧めします。

ストリップを実行する前には、ストリップしようとしている実行ファイルが実行中でないことを十分確認してください。 また 6.4. 「Chroot 環境への移行」 に示したコマンドにより chroot 環境に入っているかどうか定かでない場合は、 いったんログアウトしてください。

```
logout
```

再度 chroot 環境に入ります。

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

以下により実行バイナリやライブラリを安全にストリップします。

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

ファイルフォーマットが認識できないファイルがいくつも警告表示されますが、無視して構いません。 この警告は、処理したファイルが実行モジュールではなくスクリプトファイルであることを示しています。

ディスク容量が極端に少ない場合は /{,usr/}{bin,sbin} ディレクトリにある実行モジュールに対して --strip-all オプションを用いることもできます。 この場合さらに数 MB の容量を節約できます。 ただしこれをライブラリに対して用いてはなりません。 これを用いてしまうとライブラリが破壊されてしまいます。

6.63. 仕切り直し

それまで入っていた chroot 環境からいったん抜け出て、以下の chroot コマンドにより入り直します。

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

上を実行するのは /tools ディレクトリがもう必要ないからです。 ですから /tools ディレクトリが一切無くてよいなら削除しても構いません。

注記

/tools ディレクトリを削除すると、ツールチェーンのテストに用いていた Tcl、Expect、DejaGNU も削除することになります。 後々これらのプログラムを用いるなら、再度コンパイルとインストールを行う必要があります。 BLFS ブックにてその手順を説明しているので http://www.linuxfromscratch.org/blfs/ を参照してください。

仮想カーネルファイルシステムを、手動により、あるいはリブートによりアンマウントした場合は chroot 環境に入る前にそれらがマウントされていることを確認してください。 その作業手順は 6.2.2. 「/dev のマウントと有効化」 と 6.2.3. 「仮想カーネルファイルシステムのマウント」 で説明しています。

第7章 ブートスクリプトの設定

7.1. はじめに

この章では LFS ブートスクリプトパッケージのインストールと設定について説明します。 スクリプトのほとんどは修正する必要がありませんが、 中にはハードウェアに依存する情報を取り扱うため追加設定を要するものもあります。

System V系のスクリプトが広く用いられていることから、本書でもこれを利用します。 これとは別の方法として BSD 系の初期化スクリプトがあり http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt にて説明されています。また LFS メーリングリストで 「depinit」 という語を検索してみれば、さらに別の方法が示されていますので確認してください。

初期化スクリプトに関して別の方法をとるのであれば、本章は読み飛ばして 第8章 に進んでください。

7.2. LFS-ブートスクリプト-20100627

LFS-ブートスクリプトパッケージは LFS システムの起動、終了時に利用するスクリプトを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 468 KB

7.2.1. LFS ブートスクリプトのインストール

パッケージをインストールします。

```
make install
```

7.2.2. LFS ブートスクリプトの構成

インストールスクリプト: checkfs, cleanfs, console, consolelog, functions, halt, ifdown, ifup, localnet, modules, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysctl, sysklogd, template, udev, udev_retry
インストールディレクトリ: /etc/rc.d, /etc/sysconfig

概略説明

checkfs ファイルシステムがマウントされる前にその整合性をチェックします。（ただしジャーナルファイルシステムとネットワークベースのファイルシステムは除きます。）

cleanfs リブートの際に不要となるファイルを削除します。 例えば /var/run/ ディレクトリや /var/lock/ ディレクトリの配下にあるファイルです。 /var/run/utmp ファイルは再生成されます。 また /etc/nologin、 /fastboot、 /forcefsck がおそらく存在しており、これらは削除されます。

console 必要となるキーボードレイアウトに対しての正しいキーマップテーブルをロードします。 同時にスクリーンフォントもセットします。

consolelog カーネルのログレベルを設定して、コンソールに出力されるメッセージを制御します。

functions 共通的な関数を提供します。 例えばエラーやステータスのチェックなどであり、これはブートスクリプトの多くが利用します。

halt システムを停止します。

ifdown ネットワークデバイスを停止するネットワークスクリプトをサポートします。

ifup ネットワークデバイスを起動するネットワークスクリプトをサポートします。

localnet システムのホスト名とローカルループバックデバイスを設定します。

modules /etc/sysconfig/modules にて一覧設定されているカーネルモジュールをロードします。 その際には引数が指定され利用されます。

mountfs ファイルシステムをすべてマウントします。 ただし noauto が設定されているものやネットワークベースのファイルシステムは除きます。

mountkernfs 仮想カーネルファイルシステムをマウントします。 例えば proc などです。

network ネットワークカードなどのネットワークインターフェースを設定します。 そして（可能であれば）デフォルトゲートウェイを設定します。

rc ランレベルを制御するマスタースクリプト。 他のブートスクリプトを一つずつ実行します。 その際には実行されるシンボリックの名前によって実行順序を決定します。

reboot システムを再起動します。

sendsignals システムが再起動または停止する前に、プロセスすべてが停止していることを確認します。

setclock ハードウェアクロックが UTC 時刻に設定されていなければ、カーネルクロックをローカル時刻としてリセットします。

static ネットワークインターフェースに対して固定 IP（Internet Protocol）アドレスを割り当てるために必要となる機能を提供します。

swap スワップファイルやスワップパーティションを有効または無効にします。

sysctl /etc/sysctl.conf ファイルが存在している場合、実行中のカーネルに対してシステム設定値をロードします。

sysklogd システムログデーモンおよびカーネルログデーモンの起動と停止を行います。

template	他のデーモン用としてブートスクリプトを生成するためのテンプレート。
udev	/dev ディレクトリを準備して Udev を起動します。
udev_retry	Udev の uevent が失敗した場合にこれを再実行します。 そして必要に応じて、生成されたルールファイルを /dev/.udev から /etc/udev/rules.d へコピーします。

7.3. ブートスクリプトはどのようにして動くのか

Linux では SysVinit という特別なブート機能があり ランレベル（run-levels）という考え方に基づいています。 ランレベルの扱いはシステムによって異なりますので、ある Linux において動作しているからといって LFS においても全く同じように動くわけではありません。 LFS では独自の方法でこれを取り入れることにします。 ただし標準として受け入れられるような方法を取ります。

SysVinit（これ以降は 「init」 と表現します）はランレベルという仕組みにより動作します。 ランレベルには7つのレベル（0 から 6）があります。（実際にはランレベルはそれ以上あるのですが、特殊な場合であって普通は利用されません。 詳しくは **init(8)** を参照してください。）各レベルは、コンピュータの起動時における処理動作に対応づいており、デフォルトのランレベルは 3 となっています。 ランレベルの詳細を以下に説明します。

```
0: コンピュータの停止
1: シングルユーザーモード
2: マルチユーザーモード、ネットワークなし
3: マルチユーザーモード、ネットワークあり
4: 将来の拡張用として予約されています。 3 と同じものとして扱われます。
5: 4 と同様。通常（X の xdm や KDE の kdm のような）GUI ログインに用いられます。
6: コンピュータの再起動
```

ランレベルを変更するには init **<runlevel>** を実行します。 **<runlevel>** はランレベルを示す数字です。 例えばコンピュータを再起動するには init 6 コマンドを実行します。 これは reboot コマンドのエイリアスとなっています。 同様に init 0 は halt のエイリアスです。

/etc/rc.d ディレクトリの配下には複数のサブディレクトリがあります。 そのディレクトリ名は **rc?.d** のようになっています。（? はランレベルの数字を表します。）また **rcsysinit.d** というサブディレクトリもあります。 それらサブディレクトリ内には数多くのシンボリックリンクがあります。 シンボリックリンクの先頭一文字には K や S が用いられ、続いて二桁の数値文字がつけられています。 K はサービスの停止（kill）、S はサービスの起動（start）を意味します。 二桁の数字はスクリプトの起動順を定めるもので、00 から 99 までが割振られ、小さな数字から順に実行されます。 init コマンドによってランレベルが変更される時は、そのランレベルに応じて必要なサービスが起動するか停止することになります。

スクリプトファイルは **/etc/rc.d/init.d** ディレクトリにあります。 実際の処理はここにあるファイルが用いられます。 これらに対してはシンボリックリンクが用意されています。 サービスの起動と停止を行うシンボリックリンクは **/etc/rc.d/init.d** ディレクトリにあるスクリプトを指し示しています。 このようにしているのは、各スクリプトが **start**、 **stop**、 **restart**、 **reload**、 **status** といった様々なパラメータにより呼び出されるためです。 K の名前を持つシンボリックリンクが起動されるということは **stop** パラメータをつけて該当するスクリプトが実行されるということです。 同様に S の名前を持つシンボリックリンクが起動されるということは **start** パラメータをつけて呼び出されるということになります。

上の説明には例外があります。 **rc0.d** ディレクトリと **rc6.d** ディレクトリにある、S で始まるシンボリックリンクはサービスを何も起動させません。 **stop** パラメータが与えられ、何らかのサービスを停止します。 ユーザーがシステムを再起動したり停止したりする際には、サービスを起動させる必要はないわけで、システムを停止するだけで済むからです。

スクリプトに対するパラメータは以下のとおりです。

start
サービスを起動します。

stop
サービスを停止します。

restart
サービスをいったん停止し再起動します。

reload
サービスの設定ファイルを更新します。 設定ファイルが変更されたものの、サービスの再起動は必要ではない場合に利用します。

status
サービスがどの PID 値で動いているかを表示します。

ブート機能を動作させる方法は自由に取り決めて設定して構いません。 このシステムはつまるところあなた自身のシステムだからです。 上に示したファイル類はブート機能を定めた一例に過ぎません。

7.4. Setclock スクリプトの設定

setclock スクリプトはハードウェアクロックから時刻を読み取ります。 ハードウェアクロックは BIOS クロック、あるいは CMOS（Complementary Metal Oxide Semiconductor）クロックとしても知られているものです。 ハードウェアクロックが UTC に設定されていると setclock スクリプトは /etc/localtime ファイルを参照して、ハードウェアクロックの示す時刻をローカル時刻に変換します。 /etc/localtime ファイルは hwclock プログラムに対して、ユーザーがどのタイムゾーンに位置するかを伝えます。 ハードウェアクロックが UTC に設定されているかどうかを知る方法はないので、手動で設定を行う必要があります。

setclock スクリプトは udev によって起動されます。この時というのはブート時であり、カーネルがハードウェアを検出する時です。 停止パラメータを与えて手動でこのスクリプトを実行することもできます。 その場合 CMOS クロックに対してシステム時刻が保存されます。

ハードウェアクロックが UTC に設定されているかどうか忘れた場合は hwclock --localtime --show を実行すれば確認できます。 このコマンドにより、ハードウェアクロックに基づいた現在時刻が表示されます。 その時刻が手元の時計と同じ時刻であれば、ローカル時刻として設定されているわけです。 一方それがローカル時刻でなかった場合は、おそらくは UTC に設定されているからでしょう。 hwclock によって示された時刻からタイムゾーンに応じた一定時間を加減してみてください。 例えばタイムゾーンが MST であった場合、これは GMT -0700 なので、7時間を加えればローカル時刻となります。

ハードウェアクロックが UTC 時刻として設定されていない 場合は、以下に示す変数 UTC の値を 0（ゼロ）にしてください。

以下のコマンドを実行して /etc/sysconfig/clock ファイルを新規に作成します。

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# Set this to any options you might need to give to hwclock,
# such as machine hardware clock type for Alphas.
CLOCKPARAMS=

# End /etc/sysconfig/clock
EOF
```

LFS において時刻の取り扱い方を示した分かりやすいヒントが http://www.linuxfromscratch.org/hints/downloads/files/time.txt にあります。 そこではタイムゾーン、UTC、環境変数 TZ などについて説明しています。

7.5. Linux コンソールの設定

この節ではブートスクリプト console、 consolelog の設定方法について説明します。 このスクリプトはキーボードマップ、コンソールフォント、カーネルログレベルを設定します。 非アスキー文字（例えば著作権、ポンド記号、ユーロ記号など）を使わず、キーボードが US 配列であるなら、本節は読み飛ばしてください。 console ブートスクリプトの設定ファイルが存在しない場合は、このスクリプトは何も行いません。

console スクリプトと consolelog スクリプトは、設定情報を /etc/sysconfig/console ファイルから読み込みます。まずは利用するキーボードマップとスクリーンフォントを定めます。 様々な言語に応じた設定方法については http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html を参照してください。 よく分からない場合は /lib/kbd ディレクトリを見て、正しいキーマップとスクリーンフォントを探してください。 マニュアルページ loadkeys(1) と setfont(8) を見て、これらのプログラムに対する適切な引数を決定してください。

/etc/sysconfig/console ファイルの各行には、変数 = "値" という記述を行います。 そして変数には以下に示すものが利用可能です。

LOGLEVEL
: この変数は、コンソールに出力されるカーネルメッセージのログレベルを指定するもので dmesg コマンドにより設定されます。 有効な設定値は "1"（メッセージ出力なし）から "8" まであり、デフォルトは "7" です。

KEYMAP
: この変数は loadkeys プログラムに対する引数を指定します。 このプログラムは 「es」 などのキーマップをロードします。 この変数がセットされていない場合、ブートスクリプトは loadkeys プログラムを実行せず、デフォルトのカーネルキーマップが用いられます。

KEYMAP_CORRECTIONS
この変数は（あまり利用されませんが）loadkeys プログラムを二度目に呼び出す際の引数を指定します。 普通のキーマップでは十分な設定にならない時の微調整を行うために利用します。 例えばユーロ記号がキーマップの中に含まれておらずこれを付け加える場合には、この変数に対して 「euro2」 を設定します。

FONT
この変数は setfont プログラムへの引数を指定します。 一般にこの変数にはフォント名、「-m」、アプリケーションキャラクタマップ（application character map）を順に指定します。 例えばフォントとして「lat1-16」、アプリケーションキャラクタマップとして 「8859-1」 を指定する場合、この変数には 「lat1-16 -m 8859-1」 を設定します。（これは米国にて適当な設定となります。）UTF-8 モードの場合、カーネルは UTF-8 キーマップ内の 8 ビットキーコードを変換するためにアプリケーションキャラクタマップを利用します。 したがって ”-m” パラメータには、キーマップ内キーコードのエンコーディングを指定する必要があります。

UNICODE
コンソールを UTF-8 モードにするには、この変数を 「1」、 「yes」、 「true」 のいずれかに指定します。 UTF-8 ベースのロケールであればこの設定を行います。 そうでないロケールにおいて設定するのは不適切です。

LEGACY_CHARSET
キーボードレイアウトの多くに対して、Kbd パッケージは標準的な Unicode キーマップを提供していません。 この変数にて UTF-8 ではないキーマップのエンコーディングが指定されていたら console ブートスクリプトは利用可能な UTF-8 キーマップに変換します。

以下はいくつかの設定例です。

- Unicode を用いない設定では、普通は KEYMAP 変数と FONT 変数のみを定めます。 例えばポーランド語の設定であれば以下のようになります。

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="pl2"
FONT="lat2a-16 -m 8859-2"

# End /etc/sysconfig/console
EOF
```

- 上で述べたように、普通のキーマップの設定に対して多少の修正を必要とする場合もあります。 以下の例はドイツ語のキーマップにユーロ記号を加える例です。

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
FONT="lat0-16 -m 8859-15"

# End /etc/sysconfig/console
EOF
```

- 以下は Unicode を用いたブルガリア語の設定例です。 通常のキーマップが存在しているものと仮定しています。

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="LatArCyrHeb-16"

# End /etc/sysconfig/console
EOF
```

- 上の例においては 512 個のグリフを持つ LatArCyrHeb-16 フォントを利用しています。 この場合、フレームバッファを利用していなければ Linux コンソール上に鮮やかな色づけを行うことは出来なくなります。 フレームバッファがない状態で文字フォントを変更することなく色づけを適切に行いたい場合は、以下に示すように 256 個のグリフを持った、この言語に固有のフォントを用いる方法もあります。

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="cyr-sun16"

# End /etc/sysconfig/console
EOF
```

- 以下の例では ISO-8859-15 から UTF-8 へのキーマップ変換の自動化（keymap autoconversion）を指定し、Unicode におけるデッドキー（dead keys）を有効にするものです。

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
FONT="LatArCyrHeb-16 -m 8859-15"

# End /etc/sysconfig/console
EOF
```

- キーマップにデッドキー（dead keys）を持つものがあります。 そのキー自身は文字を意味するものではなく、次のキー入力による文字に対するアクセント記号をつける目的のものなどです。 または複合的な入力規則を定義するもの、例えば 「Ctrl+.、A、E を入力することで Æ を得るもの」 があります。 Linux-2.6.35.4 ではキーマップに応じてデッドキーや複合的な入力規則を解釈します。 ただしこれが正しく動作するのは、元の文字がマルチバイトではない場合に限ります。 このような欠点は西欧のキーマップでは問題にはなりません。 アクセント記号なら、アクセント記号がついていない ASCII 文字を使ったり、ASCII 文字を二つ使って工夫したりするからです。 しかし UTF-8 モードでは問題になります。 例えばギリシャ語にて 「alpha」 の文字の上にアクセント記号を付けたい場合が問題です。 これを解決するには、一つには UTF-8 の利用を諦めることであり、もう一つは X ウィンドウシステムを使うことで、そのような入力処理の制約を解消することです。
- 中国語、日本語、韓国語などを利用する場合 Linux コンソールにはそれらの文字を表示できません。 この言語を利用するユーザーは X ウィンドウシステムを使ってください。 そこで用いるフォントは、必要となるコード範囲の文字を有しており、入力メソッドも用意されています。（例えば SCIM は数多くの言語入力をサポートしています。）

注記

/etc/sysconfig/console ファイルは Linux のテキストコンソール上の言語設定を行うだけです。 X ウィンドウシステム、SSH セッション、シリアルコンソールでのキーボードレイアウトや端末フォントの設定とは無関係です。 それらに対しては、上に列記した最後の二項目における制約は適用されません。

日本語訳情報

日本の方であれば 「日本語106キーボード」 をほぼ間違いなくお使いかと思いますので KEYMAP 変数には「jp106」 を設定することになるでしょう。 FONT 変数について訳者は十分な知識がありません。 ここに何を設定すべきか分からない（調べていない）ため、何も設定しないでいる状態です。 訳者は LFS システム構築後は SSH 接続によりシステムアクセスしており、その場合ここでのフォントの設定がどうであろうと（おそらく）無関係であるため、あまり気にせずにいます。 何か情報を頂けるようであればご教示よろしくお願いいたします。

訳者が行っている設定は以下のとおりです。

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="jp106"

# End /etc/sysconfig/console
EOF
```

7.6. Sysklogd スクリプトの設定

sysklogd スクリプトは syslogd プログラムをパラメータ **-m 0** で実行します。 このオプションは syslogd がデフォルトで 20分おきにログファイルに対して周期的にタイムスタンプを書き込む機能を無効にします。 この機能を有効にしたい場合は **sysklogd** スクリプトを書き換えてください。詳しくは **man syslogd** を入力して man ページを参照してください。

7.7. /etc/inputrc ファイルの生成

inputrc ファイルはキーボードに応じたキーボードマップを定めます。 このファイルは入力に関連するライブラリ Readline が利用するもので、このライブラリは Bash などのシェルから呼び出されます。

ユーザー固有のキーボードマップを必要となるのはまれなので、以下の **/etc/inputrc** ファイルによって、ログインユーザーすべてに共通するグローバルな定義を生成します。 各ユーザーごとにこのデフォルト定義を上書きする必要が出てきた場合は、ユーザーのホームディレクトリに **.inputrc** ファイルを生成して、修正マップを定義することもできます。

inputrc ファイルの設定方法については info bash により表示される Readline Init File の節に詳しい説明があります。 info readline にも有用な情報があります。

以下はグローバルな inputrc ファイルの一般的な定義例です。 コメントをつけて各オプションを説明しています。 コメントはコマンドと同一行に記述することはできません。 以下のコマンドを実行してこのファイルを生成します。

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

# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On

# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the value
# contained in the 1st argument to the readline specific functions
"\eOd": backward-word
"\eOc": forward-word

# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line

# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# End /etc/inputrc
EOF
```

7.8. Bash シェルの初期起動ファイル

シェルプログラムである /bin/bash（これ以降は単に 「シェル」 と表現します）は、初期起動ファイルをいくつも利用して環境設定を行います。 個々のファイルにはそれぞれに目的があり、ログインや対話環境を様々に制御します。 /etc ディレクトリにあるファイルは一般にグローバルな設定を行います。 これに対応づいたファイルがユーザーのホームディレクトリにある場合は、グローバルな設定を上書きします。

対話型ログインシェルは /bin/login プログラムを利用して /etc/passwd ファイルを読み込み、ログインが成功することで起動します。 同じ対話型でも非ログインシェルの場合は [prompt]$/bin/bash のようなコマンドラインからの入力を経て起動します。 非対話型のシェルはシェルスクリプト動作中に実行されます。 非対話型であるのは、スクリプトの実行の最中にユーザーからの入力を待つことがないためです。

より詳しい情報は info bash の Bash Startup Files and Interactive Shells の節を参照してください。

/etc/profile ファイルと ~/.bash_profile ファイルは、対話型のログインシェルとして起動した時に読み込まれます。

本節の終わりに示す /etc/profile ファイルは言語を設定するために必要となる環境変数を定義します。 これを設定することによって以下の内容が定められます。

- プログラムの出力結果を指定した言語で得ることができます。
- キャラクタを英字、数字、その他のクラスに分類します。 この設定は、英語以外のロケールにおいて、コマンドラインに非アスキー文字が入力された場合に bash が正しく入力を受け付けるために必要となります。
- 各国ごとに正しくアルファベット順が並ぶようにします。
- 適切なデフォルト用紙サイズを設定します。
- 通貨、日付、時刻を正しい書式で出力するように設定します。

以下において <ll> と示しているものは、言語を表す２文字の英字（例えば 「en」）に、また <CC> は、国を表す２文字の英字（例えば 「GB」）にそれぞれ置き換えてください。 <charmap> は、選択したロケールに対応したキャラクタマップ（charmap）に置き換えてください。 オプションの修飾子として 「@euro」 といった記述もあります。

以下のコマンドを実行すれば Glibc が取り扱うロケールを一覧で見ることができます。

```
locale -a
```

キャラクタマップにはエイリアスがいくつもあります。 例えば 「ISO-8859-1」 は 「iso8859-1」 や 「iso88591」 として記述することもできます。 ただしアプリケーションによってはエイリアスを正しく取り扱うことができないものがあります。（「UTF-8」 の場合、「UTF-8」 と書かなければならず、これを 「utf8」 としてはならない場合があります。）そこでロケールに対する正規の名称を選ぶのが最も無難です。 正規の名称は以下のコマンドを実行すれば分かります。 ここで <locale name> は locale -a コマンドの出力から得られたロケールを指定します。（本書の例では 「en_GB.iso88591」 としています。）

```
LC_ALL=<locale name> locale charmap
```

「en_GB.iso88591」 ロケールの場合、上のコマンドの出力は以下となります。

```
ISO-8859-1
```

出力された結果が 「en_GB.ISO-8859-1」 に対するロケール設定として用いるべきものです。 こうして探し出したロケールは動作確認しておくことが重要です。 Bash の起動ファイルに記述するのはその後です。

```
LC_ALL=<locale name> locale language
LC_ALL=<locale name> locale charmap
LC_ALL=<locale name> locale int_curr_symbol
LC_ALL=<locale name> locale int_prefix
```

上のコマンドを実行すると、言語名やロケールに応じたキャラクタエンコーディングが出力されます。 また通貨や各国ごとの国際電話番号プレフィックスも出力されます。 コマンドを実行した際に以下のようなメッセージが表示されたら、第6章にてロケールをインストールしていないか、あるいはそのロケールが Glibc のデフォルトのインストールではサポートされていないかのいずれかです。

```
locale: Cannot set LC_* to default locale: No such file or directory
```

このエラーが発生したら localedef コマンドを使って、目的とするロケールをインストールするか、別のロケールを選ぶ必要があります。 これ以降の説明では Glibc がこのようなエラーを生成していないことを前提に話を進めます。

LFS には含まれない他のパッケージにて、指定したロケールをサポートしていないものがあります。 例えば X ライブラリ（X ウィンドウシステムの一部）では、内部ファイルに指定されたキャラクタマップ名に合致しないロケールを利用した場合に、以下のようなメッセージを出力します。

```
Warning: locale not supported by Xlib, locale set to C
```

Xlib ではキャラクタマップはたいてい、英大文字とダッシュ記号を用いて表現されます。 例えば ”iso88591” ではなく ”ISO-8859-1” となります。 ロケール設定におけるキャラクタマップ部分を取り除いてみれば、適切なロケール設定を見出すことができます。 これはまた locale charmap コマンドを使って、設定を変えてみてロケールを指定してみれば確認できます。 例えば ”de_DE.ISO-8859-15@euro” という設定を ”de_DE@euro” に変えてみて Xlib がそのロケールを認識するかどうか確認してみてください。

これ以外のパッケージでも、パッケージが求めるものとは異なるロケール設定がなされた場合に、適切に処理されないケースがあります。（そして必ずしもエラーメッセージが表示されない場合もあります。）そういったケースでは、利用している Linux ディストリビューションがどのようにロケール設定をサポートしているかを調べてみると、有用な情報が得られるかもしれません。

適切なロケール設定が決まったら /etc/profile ファイルを生成します。

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

export LANG=<ll>_<CC>.<charmap><@modifiers>

# End /etc/profile
EOF
```

ロケール設定の 「C」（デフォルト）と 「en_US」（米国の英語利用ユーザーに推奨）は異なります。 「C」 は US-ASCII 7 ビットキャラクタセットを用います。 もし最上位ビットがセットされたキャラクタがあれば不適当なものとして取り扱います。 例えば ls コマンドにおいてクエスチョン記号が表示されることがあるのはこのためです。 また Mutt や Pine などにより電子メールが送信される際に、そういった文字は RFC には適合しないメールとして送信されます。 送信された文字は 「不明な 8ビット（unknown 8-bit)」 として示されます。 そこで 8ビット文字を必要としない場合には 「C」 ロケールを指定してください。

UTF-8 ベースのロケールは多くのプログラムにおいてサポートされていません。 この問題については http://www.linuxfromscratch.org/blfs/view/svn/introduction/locale-issues.html にて説明しており、可能なものは解決を図っていこうとしているところです。

7.9. LFS システムにおけるデバイスとモジュールの扱い

第6章にて Udev パッケージをインストールしました。 このパッケージがどのように動作するかの詳細を説明する前に、デバイスを取り扱うかつての方法について順を追って説明していきます。

Linux システムは一般に、スタティックなデバイス生成方法を採用していました。 この方法では /dev のもとに膨大な量の（場合によっては何千にもおよぶ）デバイスノードが生成されます。 現実に存在するハードウェアデバイスが存在するかどうかに関わらずです。 これは MAKEDEV スクリプトを通じて生成されます。 このスクリプトからは mknod プログラムが呼び出されますが、その呼び出しは、この世に存在するありとあらゆるデバイスのメジャー/マイナー番号を用いて行われます。

Udev による方法では、カーネルが検知したデバイスだけがデバイスノードとなります。 デバイスノードはシステムが起動するたびに生成されることになるので、 tmpfs ファイルシステム上に保存されます。（tmpfs は仮想ファイルシステムであり、メモリ上に置かれます。）デバイスノードの情報はさほど多くないので、消費するメモリ容量は無視できるほど少ないものです。

7.9.1. 開発経緯

2000年2月に新しいファイルシステム devfs がカーネル 2.3.46 に導入され、2.4系の安定版カーネルにて利用できるようになりました。 このファイルシステムはカーネルのソース内に含まれ実現されていましたが、デバイスを動的に生成するこの手法は、主要なカーネル開発者の十分な支援は得られませんでした。

devfs が採用した手法で問題になるのは、主にデバイスの検出・生成・命名の方法です。 特にデバイスの命名方法がおそらく最も重大な問題です。 一般的に言えることとして、デバイス名が変更可能であるならデバイス命名の規則はシステム管理者が考えることであって、特定の開発者に委ねるべきことではありません。 また devfs にはその設計に起因した競合の問題があるため、根本的にカーネルを修正しなければ解消できる問題ではありません。 そこで長い間、保守されることがなかったために非推奨（deprecated）として位置づけられ、最終的に 2006年6月にはカーネルから取り除かれました。

開発版の 2.5 系カーネルと、後にリリースされた安定版のカーネル 2.6 系を経て、新しい仮想ファイルシステム sysfs が登場しました。 sysfs が実現したのは、システムのハードウェア設定をユーザー空間のプロセスとして表に出したことです。 ユーザー空間での設定を可視化したことによって devfs が為していたことを、ユーザー空間にて現実に見ることが可能になったわけです。

7.9.2. Udev の実装

7.9.2.1. Sysfs ファイルシステム

sysfs ファイルシステムについては上で簡単に触れました。 sysfs はどのようにしてシステム上に存在するデバイスを知るのか、そしてどのデバイス番号が利用されるのか。 そこが知りたいところです。 カーネルに直接組み込まれて構築されたドライバでは、対象のオブジェクトがカーネルによって検出されたものとしてそのオブジェクトを sysfs に登録します。 モジュールとしてコンパイルされたドライバでは、その登録がモジュールのロード時に行われます。 sysfs ファイルシステムが（/sys に）マウントされると、組み込みのドライバによって sysfs に登録されたデータは、ユーザー空間のプロセスとデバイスノード生成を行う udevd にて利用可能となります。

7.9.2.2. Udev ブートスクリプト

初期起動スクリプト S10udev は、Linux のブート時にデバイスノード生成を受け持ちます。 このスクリプトは /sbin/hotplug のデフォルトから uevent ハンドラを取り除きます。 この時点でカーネルは、他の実行モジュールを呼び出す必要がないからです。 そのかわりに、カーネルが起動する uevent をネットリンクソケット（netlink socket）上で待ち受けます。 そしてブートスクリプトが /lib/udev/devices 内にある静的なデバイスノードをすべて /dev にコピーします。 デバイスやディレクトリ、シンボリックリンクがこの時点で利用可能になっていないと、システム起動の初期段階において動的デバイスを扱う処理が動作しないためです。 あるいは udevd 自身がそれを必要とするからでもあります。 /lib/udev/devices 内に静的なデバイスノードを生成することで、動的デバイスを取り扱うことができないデバイスも動作させることができます。 こうしてブートスクリプトは Udev デーモン、つまり udevd を起動し、それがどのような uevent であっても対応できるものとなります。 最後にブートスクリプトはカーネルに対して、すべてのデバイスにおいて既に登録されている uevent を再起動させ、 udevd がそれを待ち受けるものとなります。

7.9.2.3. デバイスノードの生成

Udev はデバイスのメジャー番号、マイナー番号を認識するために /sys ディレクトリ内の sysfs の情報を参照します。 例えば /sys/class/tty/vcs/dev には 「7:0」 という文字があります。 この文字は udevd が利用するもので、メジャー番号が 7、マイナー番号が 0 のデバイスノードを生成します。 /dev ディレクトリ配下に生成されるノードの名称とパーミッションは、 /etc/udev/rules.d/ ディレクトリにある各種ファイルが指定する規則に従って決まります。 それらのファイルは番号付けがされています。 LFS-ブートスクリプトパッケージにおける方法に似ています。 Udev がデバイスを生成しようとしてその生成規則が見つけられなかった場合は、デフォルトのパーミッションは 660、デフォルトの所有者は root:root となります。 Udev におけるデバイス生成規則を設定するファイルについて、その文法を示したドキュメントが /usr/share/doc/udev-161/writing_udev_rules/index.html にあります。

7.9.2.4. モジュールのロード

モジュールとしてコンパイルされたデバイスドライバの場合、デバイス名の別名が作り出されています。 その別名は modinfo プログラムを使えば確認することができます。 そしてこの別名は、モジュールがサポートするバス固有の識別子に関連づけられます。 例えば snd-fm801 ドライバは、ベンダーID 0x1319 とデバイスID 0x0801 の PCI ドライバをサポートします。 そして 「pci:v00001319d00000801sv*sd*bc04sc01i*」 というエイリアスがあります。 たいていのデバイスでは、sysfs を通じてドライバがデバイスを扱うものであり、ドライバのエイリアスをバスドライバが提供します。 /sys/bus/pci/devices/0000:00:0d.0/modalias ファイルならば 「pci:v00001319d00000801sv00001319sd00001319bc04sc01i00」 という文字列を含んでいるはずです。 Udev が提供するデフォルトの生成規則によって udevd から /sbin/modprobe が呼び出されることになり、その際には uevent に関する環境変数 MODALIAS の設定内容が利用されます。（この環境変数の内容は sysfs 内の modalias ファイルの内容と同じはずです。）そしてワイルドカードが指定されているならそれが展開された上で、エイリアス文字列に合致するモジュールがすべてロードされることになります。

上の例で forte ドライバがあったとすると、snd-fm801 の他にそれもロードされてしまいます。 これは古いものでありロードされて欲しくないものです。 不要なドライバのロードを防ぐ方法については後述しているので参照してください。

カーネルは、ネットワークプロトコル、ファイルシステム、NLS サポートといった各種モジュールも、要求に応じてロードすることもできます。

7.9.2.5. ホットプラグ可能な/ダイナミックなデバイスの扱い

USB（Universal Serial Bus）で MP3 プレイヤーを接続しているような場合、カーネルは現在そのデバイスが接続されているということを認識しており、uevent が生成済の状態にあります。 その uevent は上で述べたように udevd が取り扱うことになります。

7.9.3. モジュールロードとデバイス生成の問題

自動的にデバイスが生成される際には、いくつか問題が発生します。

7.9.3.1. カーネルモジュールが自動的にロードされない問題

Udev がモジュールをロードできるためには、バス固有のエイリアスがあって、バスドライバが sysfs に対して適切なエイリアスを提供していることが必要です。 そうでない場合は、別の手段を通じてモジュールのロードを仕組まなければなりません。 Linux-2.6.35.4 においての Udev は、 INPUT、IDE、PCI、USB、SCSI、SERIO、FireWire の各デバイスに対するドライバをロードします。 それらのデバイスドライバが適切に構築されているからです。

目的のデバイスドライバが Udev に対応しているかどうかは、 modinfo コマンドに引数としてモジュール名を与えて実行します。 /sys/bus ディレクトリ配下にあるそのデバイス用のディレクトリを見つけ出して、 modalias ファイルが存在しているかどうかを見ることで分かります。

sysfs に modalias ファイルが存在しているなら、そのドライバはデバイスをサポートし、デバイスとの直接のやり取りが可能であることを表します。 ただしエイリアスを持っていなければ、それはドライバのバグです。 その場合は Udev に頼ることなくドライバをロードするしかありません。 そしてそのバグが解消されるのを待つしかありません。

/sys/bus ディレクトリ配下の対応するディレクトリ内に modalias ファイルがなかったら、これはカーネル開発者がそのバス形式に対する modalias のサポートをまだ行っていないことを意味します。 Linux-2.6.35.4 では ISA バスがこれに該当します。 最新のカーネルにて解消されることを願うしかありません。

Udev は snd-pcm-oss のような 「ラッパー（wrapper)」 ドライバや loop のような、現実のハードウェアに対するものではないドライバは、ロードすることができません。

7.9.3.2. カーネルモジュールが自動的にロードされず Udev もロードしようとしない問題

「ラッパー（wrapper)」 モジュールが単に他のモジュールの機能を拡張するだけのものであるなら（例えば snd-pcm-oss は snd-pcm の機能拡張を行うもので、 OSS アプリケーションに対してサウンドカードを利用可能なものにするだけのものであるため）modprobe の設定によってラッパーモジュールを先にロードし、その後でラップされるモジュールがロードされるようにします。 これは以下のように /etc/modprobe.d/<filename>.conf ファイル内にて 「install」の記述行を加えることで実現します。

```
install snd-pcm /sbin/modprobe -i snd-pcm ; \
    /sbin/modprobe snd-pcm-oss ; true
```

問題のモジュールがラッパーモジュールではなく、単独で利用できるものであれば、 S05modules ブートスクリプトを編集して、システム起動時にこのモジュールがロードされるようにします。 これは /etc/sysconfig/modules ファイルにて、そのモジュール名を単独の行に記述することで実現します。 この方法はラッパーモジュールに対しても動作しますが、この場合は次善策となります。

7.9.3.3. Udev が不必要なモジュールをロードする問題

不必要なモジュールはこれをビルドしないことにするか、あるいは /etc/modprobe.d/blacklist.conf ファイルにブラックリスト（blacklist）として登録してください。 例えば forte モジュールをブラックリストに登録するには以下のようにします。

```
blacklist forte
```

ブラックリストに登録されたモジュールは modprobe コマンドを使えば手動でロードすることもできます。

7.9.3.4. Udev が不正なデバイスを生成する、または誤ったシンボリックリンクを生成する問題

デバイス生成規則が意図したデバイスに合致していないと、この状況が往々にして起こります。 例えば生成規則の記述が不十分であった場合、SCSI ディスク（本来望んでいるデバイス）と、それに対応づいたものとしてベンダーが提供する SCSI ジェネリックデバイス（これは誤ったデバイス）の両方に生成規則が合致してしまいます。 記述されている生成規則を探し出して正確に記述してください。 その際には udevadm info コマンドを使って情報を確認してください。

7.9.3.5. Udev 規則が不審な動きをする問題

この問題は、一つ前に示したものが別の症状となって現れたものかもしれません。 そのような理由でなく、生成規則が正しく sysfs の属性を利用しているのであれば、それはカーネルの処理タイミングに関わる問題であって、カーネルを修正すべきものです。 今の時点では、該当する sysfs の属性の利用を待ち受けるような生成規則を生成し、 /etc/udev/rules.d/10-wait_for_sysfs.rules ファイルにそれを追加することで対処できます。（/etc/udev/rules.d/10-wait_for_sysfs.rules ファイルがなければ新規に生成します。）もしこれを実施してうまくいった場合は LFS 開発メーリングリストにお知らせください。

7.9.3.6. Udev がデバイスを生成しない問題

ここでは以下のことを前提としています。 まずドライバがカーネル内に静的に組み入れられて構築されているか、あるいは既にモジュールとしてロードされていること。 そして Udev が異なった名前のデバイスを生成していないことです。

Udev がデバイスノード生成のために必要となる情報を知るためには、カーネルドライバが sysfs に対して属性データを提供していなければなりません。 これはカーネルツリーの外に配置されるサードパーティ製のドライバであれば当たり前のことです。 したがって /lib/udev/devices において、適切なメジャー・マイナー番号を用いた静的なデバイス

ノードを生成してください。（カーネルのドキュメント devices.txt またはサードパーティベンダーが提供するドキュメントを参照してください。）この静的デバイスノードは、S10udev ブートスクリプトによって /dev にコピーされます。

7.9.3.7. 再起動後にデバイスの命名順がランダムに変わってしまう問題

これは Udev の設計仕様に従って発生するもので、uevent の扱いとモジュールのロードが平行して行われるためです。このために命名順が予期できないものになります。 これを 「固定的に」 することはできません。 ですからカーネルがデバイス名を固定的に定めるようなことを求めるのではなく、シンボリックリンクを用いた独自の生成規則を作り出して、そのデバイスの固定的な属性を用いた固定的な名前を用いる方法を取ります。 固定的な属性とは例えば、Udev によってインストールされる様々な *_id という名のユーティリティが出力するシリアル番号などです。 設定例については 7.10. 「デバイスへのシンボリックリンクの生成」 や 7.13. 「ネットワークスクリプトの設定」 を参照してください。

7.9.4. 参考情報

さらに参考になるドキュメントが以下のサイトにあります：

- devfs のユーザー空間での実装方法 http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- sysfs ファイルシステム http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf
- より詳細なドキュメントへのリンク http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html

7.10. デバイスへのシンボリックリンクの生成

7.10.1. CD-ROM のシンボリックリンク

後にインストールしていくソフトウェア（例えばメディアプレーヤーなど）では、/dev/cdrom や /dev/dvd といったシンボリックリンクを必要とするものがあります。 これらはそれぞれ CD-ROM、DVD-ROM を指し示しています。 こういったシンボリックリンクは /etc/fstab ファイルに設定しておくのが便利です。 Udev が提供するスクリプトファイルで、ルールファイル（rules files）を生成するものがあります。 そのルールファイルは、各デバイスの性能に応じてシンボリックファイルを構成します。 もっともこのスクリプトファイルを利用する際には、二つ存在する動作モードのいずれを用いるかを決めなければなりません。

一つは 「パス（by-path）」 モードです。 これは USB デバイスやファームウェアデバイスに対してデフォルトで利用されます。 これによって作り出されるルールは CD や DVD デバイスに対して物理パスが用いられます。 二つめは 「ID（by-id）」 モードです。 デフォルトで IDE や SCSI デバイスに利用されます。 このモードで作り出されるルールは CD や DVD デバイス自身が持つ識別文字列が用いられます。 パスは Udev の path_id スクリプトによって決定します。 一方、識別文字列は ata_id プログラムまたは scsi_id プログラムによってハードウェアから読み出されます。 ata_id、 scsi_id のいずれであるかは、そのデバイスによって決まります。

二つの方法にはそれぞれに利点があります。 どちらの方法が適切であるかは、デバイスがどのように変更されるかによります。 デバイスに対する物理パス（そのデバイスが接続しているポートやスロット）を変更したい場合、例えば IDE ポートや USB コネクタを切り替えたいような場合、 「ID（by-id）」 モードを使うべきです。 一方、デバイスの識別文字列を変えたい場合、つまりデバイスが故障したために、同等の性能の新しいデバイスを同一コネクタに接続しようとする場合は、 「パス（by-path）」 モードを使うべきです。

いずれの変更の可能性もあるならば、より変更の可能性の高いケースに従ってモードを選ぶべきです。

重要項目

外部接続のデバイス（例えば USB 接続の CD ドライブなど）はパス（by-path）モードを用いるべきではありません。 そのようなデバイスは接続するたびに外部ポートが新しくなり、物理パスが変わってしまうためです。 こういった外部接続のデバイスを物理パスで認識させ Udev ルールを構成した場合は、あらゆるデバイスがこの問題を抱えることになります。 これは CD や DVD ドライブだけに限った話ではありません。

Udev スクリプトが利用しているキーの値を確認したい場合は /sys ディレクトリ配下を確認します。 例えば CD-ROM デバイスについては /sys/block/hdd を確認します。 そして以下のようなコマンドを実行します。

```
udevadm test /sys/block/hdd
```

出力結果には *_id というプログラム名を示した行がたくさん表示されます。 「ID (by-id)」 モードは ID_SERIAL 値が存在して空でなければこれを利用します。 そうでない時は ID_MODEL と ID_REVISION を利用します。 「パス (by-path)」 モードは ID_PATH の値を利用します。

デフォルトモードが利用状況に合わない場合は、 /lib/udev/rules.d/75-cd-aliases-generator.rules ファイルに対して以下のように修正を行います。 mode の部分は 「by-id」 か 「by-path」 に置き換えます。

```
sed -i -e 's/write_cd_rules/& mode/' \
    /lib/udev/rules.d/75-cd-aliases-generator.rules
```

ここでルールファイルやシンボリックリンクを作成する必要はありません。 この時点ではホストの /dev ディレクトリに対して LFS システムに向けてのバインドマウント (bind-mounted) を行っており、ホスト上にシンボリックリンクが存在していると仮定しているからです。 ルールファイルとシンボリックリンクは LFS システムを初めてブートした時に生成されます。

もっとも CD-ROM デバイスが複数あると、ブート時に生成されるシンボリックリンクが、ホスト利用時に指し示されていたものとは異なる場合が発生します。 デバイスの検出順は予測できないものだからです。 LFS システムを初めて起動した時の割り当ては、たぶん固定的に行われるはずです。 つまりこのことは、ホストシステムと LFS システムの双方で、シンボリックリンクが同じデバイスを指し示すことが必要である場合にのみ問題となります。 これが必要であるなら、生成されている /etc/udev/rules.d/70-persistent-cd.rules ファイルを起動後に調査して (おそらくは編集して) 割り当てられたシンボリックリンクが望むものになっているかどうかを確認してください。

7.10.2. 重複するデバイスの取り扱い方

7.9. 「LFS システムにおけるデバイスとモジュールの扱い」 で説明したように、 /dev 内に同一機能を有するデバイスがあったとすると、その検出順は本質的にランダムです。 例えば USB 接続のウェブカメラと TV チューナーがあったとして、 /dev/video0 がウェブカメラを、また /dev/video1 がチューナーをそれぞれ参照していたとしても、システム起動後はその順が逆になることがあります。 サウンドカードやネットワークカードを除いた他のハードウェアであれば、 Udev ルールを適切に記述することで、固定的なシンボリックリンクを作り出すことができます。 ネットワークカードについては、別途 7.13. 「ネットワークスクリプトの設定」 にて説明しています。 またサウンドカードの設定方法は BLFS にて説明しています。

利用しているデバイスに上の問題の可能性がある場合 (お使いの Linux ディストリビューションではそのような問題がなかったとしても) /sys/class ディレクトリや /sys/block ディレクトリ配下にある対応ディレクトリを探してください。 ビデオデバイスであれば /sys/class/video4linux/videoX といったディレクトリです。 そしてそのデバイスを一意に特定する識別情報を確認してください。 (通常はベンダー名、プロダクトID、シリアル番号などです。)

```
udevadm info -a -p /sys/class/video4linux/video0
```

シンボリックリンクを生成するルールを作ります。

```
cat > /etc/udev/rules.d/83-duplicate_devs.rules << "EOF"

# Persistent symlinks for webcam and tuner
KERNEL=="video*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", \
    SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f", ATTRS{vendor}=="0x109e", \
    SYMLINK+="tvtuner"

EOF
```

こうしたとしても /dev/video0 と /dev/video1 はチューナーとウェブカメラのいずれかをランダムに指し示すことに変わりありません。 (したがって直接このデバイス名を使ってはなりません。) しかしシンボリックリンク /dev/tvtuner と /dev/webcam は常に正しいデバイスを指し示すようになります。

7.11. localnet スクリプトの設定

localnet スクリプトの行う作業の 1 つが、システムのホスト名を定めることです。 この設定は /etc/sysconfig/network ファイルにて行います。

以下のコマンドにより /etc/sysconfig/network ファイルを生成しホスト名を定めます。

```
echo "HOSTNAME=<lfs>" > /etc/sysconfig/network
```

<lfs> の部分はコンピュータに与える名称に置き換えてください。 ここには完全修飾ドメイン名 (Fully Qualified Domain Name; FQDN) を記述しないでください。 それは次節に示す /etc/hosts ファイルにて設定します。

7.12. /etc/hosts ファイルの設定

ネットワークカードの準備ができたら完全修飾ドメイン名（fully-qualified domain name; FQDN）とそのエイリアス名を決定して /etc/hosts ファイルに記述します。 記述書式は以下のとおりです。

```
IP_address myhost.example.org aliases
```

インターネットに公開されていないコンピュータである場合（つまり登録ドメインであったり、あらかじめ IP アドレスが割り当てられていたりする場合。 普通のユーザーはこれを持ちません。）IP アドレスはプライベートネットワーク IP アドレスの範囲で指定します。 以下がそのアドレス範囲です。

```
Private Network Address Range      Normal Prefix
10.0.0.1 - 10.255.255.254           8
172.x.0.1 - 172.x.255.254           16
192.168.y.1 - 192.168.y.254         24
```

x は 16 から 31、y は 0 から 255 の範囲の数値です。

IP アドレスの例は 192.168.11.1 となります。 また FQDN の例としては lfs.example.org となります。

ネットワークカードを用いない場合でも FQDN の記述は行ってください。 特定のプログラムが動作する際に必要となることがあるからです。

以下のようにして /etc/hosts ファイルを生成します。

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
<192.168.1.1> <HOSTNAME.example.org> [alias1] [alias2 ...]

# End /etc/hosts (network card version)
EOF
```

<192.168.1.1> や <HOSTNAME.example.org> の部分は利用状況に応じて書き換えてください。（ネットワーク管理者から IP アドレスを指定されている場合や、既存のネットワーク環境に接続する場合など。）エイリアスの記述（alias1, alias2 ）は省略しても構いません。

ネットワークカードを設定しない場合は、以下のようにして /etc/hosts ファイルを生成します。

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 <HOSTNAME.example.org> <HOSTNAME> localhost

# End /etc/hosts (no network card version)
EOF
```

7.13. ネットワークスクリプトの設定

本節はネットワークカードを設定する場合にのみ作業を行っていきます。

ネットワークカードを利用しない場合は、ネットワークカードに関連する設定ファイルを生成する必要はありません。その場合は、ランレベルに対するすべてのディレクトリ（/etc/rc.d/rc*.d）から network シンボリックリンクを削除してください。

7.13.1. ネットワークインターフェースに対する固定名称の作成

Udev やモジュラー化されたネットワークドライバにおいて、ネットワークインターフェースの番号の割振りは再起動により変更されます。 ドライバモジュールの読み込みが並列で行われるためランダムになるからです。 例えば Intel 製と Realtek 製の二つのネットワークカードを持つコンピュータにおいて、 Intel 製が eth0、 Realtek 製が eth1 となったとします。 しかし時にはシステムの再起動によって番号割り振りが逆転することもあります。 これを避けるには Udev ルールを生成して、ネットワークカードの MAC アドレスに基づいて固定的に名称を定める方法があります。

ブートを繰り返しても特定のデバイスには同一の名前が割り当たるようなルール記述を試しに生成します。 まずは以下を実行します。

```
for NIC in /sys/class/net/* ; do
    INTERFACE=${NIC##*/} udevadm test --action=add $NIC
done
```

そして /etc/udev/rules.d/70-persistent-net.rules ファイルを見て、どのネットワークデバイスにどんな名前が割り当てられているかを確認します。

```
cat /etc/udev/rules.d/70-persistent-net.rules
```

このファイルの先頭にはコメントが数行あり、続いてそれぞれの NIC に対する行があります。 NIC ごとの記述では一行めがコメントで、そのハードウェア ID が記されています。（PCI カードである場合、PCI ベンダとデバイス ID が記述されます。）またドライバが検出できている場合には、カッコ書きでドライバ名も示されます。 ハードウェア ID もドライバ名も、インターフェースに対して与えられる名称とは無関係で、単に分かりやすくするために記されているにすぎません。 二行めは Udev ルールであり、その NIC を定め、名称を割り当てている記述です。

Udev ルールはいくつかのキー項目で構成され、それぞれがカンマで区切られるか、場合によっては空白文字で区切られています。 このキー項目とその内容は以下のようになります。

- SUBSYSTEM=="net" - ネットワークカードではないデバイスは無視することを指示します。
- ACTION=="add" - uevent の add イベントではないものは無視することを指示します。（uevent の "remove" イベントや "change" イベントも発生しますが、これらはネットワークインターフェースの名前を変更するものではありません。）
- DRIVERS=="?*" - Udev に対して VLAN やブリッジサブインターフェース（bridge sub-interfaces）を無視することを指示します。（サブインターフェースにはドライバがないためです。）サブインターフェースに名前が割り当てられたとすると、親デバイスの名前と衝突してしまうため、サブインターフェースの名前割り当てはスキップされます。
- ATTR{address} - このキーの値は NIC の MAC アドレスを表します。
- ATTR{type}=="1" - 特定のワイヤレスドライバでは複数の仮想インターフェースが生成されますが、そのうちの主となるインターフェースにのみルールが合致するようにします。 二つめ以降のインターフェースに対する処理は、VLAN やブリッジサブインターフェースがスキップされるのと同じくスキップされます。 名前割り当てが行われてしまうと名前衝突を起こすためです。
- KERNEL=="eth*" - 複数のネットワークインターフェースを有するマシンを取り扱うためのルールを加えます。 このルールでは全インターフェースに同一の MAC アドレスが用いられます。（PS3 などがそういったマシンになります。）各インターフェースに対して個別の命名が行われたとすると Udev はそれぞれを別のものとして取り扱います。 これはたいていの Linux From Scratch ユーザーにとって必要ありません。 ただそうなったとしても問題はありません。
- NAME - Udev がインターフェースに対して割り当てる名前をキーの値として指定します。

NAME に定義される値が重要です。 どのネットワークカードにどんな名前が割り当てられているかをよく確認してください。 そして以下において設定ファイルを生成する際には NAME に定義されている名称を利用してください。

7.13.2. ネットワークインターフェースに対する設定ファイルの生成

どのネットワークインターフェースを起動させるかは /etc/sysconfig/network-devices ディレクトリ配下のネットワークスクリプトにより設定します。 そのディレクトリには、設定を行ないたい各ネットワークインターフェースに対するサブディレクトリを準備します。 例えばネットワークインターフェースの名が 「xyz」 である場合 ifconfig.xyz というサブディレクトリとします。 このサブディレクトリ内にはネットワークインターフェースの属性、つまり IP アドレスやサブネットマスクなどを定義したファイルを置きます。

以下のコマンドは、例として eth0 デバイスに対しての ipv4 ファイルを生成するものです。

```
cd /etc/sysconfig/network-devices
mkdir -v ifconfig.eth0
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

各変数の値は各ファイルごとに適切なものに設定してください。 ONBOOT 変数を 「yes」 に設定している場合、システム起動時にネットワークスクリプトが実行され、ネットワークインターフェースカード（network interface card; NIC）を有効にします。 「yes」 以外に設定している場合、ネットワークスクリプトは NIC に対して何も行わないため NIC は有効にはなりません。

SERVICE 変数はIP アドレスの取得方法を指定します。 LFS-ブートスクリプトは IP アドレス割り当て方法をモジュール化しています。 そして /etc/sysconfig/network-devices/services ディレクトリに追加でファイルを生成すれば、他の IP アドレス割り当て方法をとることもできます。 通常は DHCP（Dynamic Host Configuration Protocol）において利用されるものです。 これについては BLFS ブックにて説明しています。

GATEWAY 変数は、デフォルトゲートウェイが存在するならその IP アドレスを指定します。 存在しない場合は、の変数設定を行っている一行をコメントにします。

PREFIX 変数はサブネットマスクにて用いられるビット数を指定します。 IP アドレスの各オクテット（octet）は 8 ビットで構成されます。 例えばサブネットマスクが 255.255.255.0 である場合、ネットワーク番号（network number）を特定するには最初の三つのオクテット（24ビット）が用いられることを意味します。 もし 255.255.255.240 であるなら、最初の 28 ビットということになります。 24 ビットを超えるプレフィックスは、通常は DSL やケーブルを用いたインターネットサービスプロバイダー（Internet Service Provider; ISP）がよく利用しています。 上の例（PREFIX=24）では、サブネットマスクは 255.255.255.0 となります。 PREFIX 変数の値は、ネットワーク環境に応じて変更してください。

7.13.3. /etc/resolv.conf ファイルの生成

インターネットへの接続を行う場合には、ドメイン名サービス（domain name service; DNS）による名前解決を必要とします。 これによりインターネットドメイン名を IP アドレスに、あるいはその逆の変換を行います。 これを行うには ISP やネットワーク管理者が指定する DNS サーバーの割り振り IP アドレスを /etc/resolv.conf ファイルに設定します。 以下のコマンドによりこのファイルを生成します。

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

domain <Your Domain Name>
nameserver <IP address of your primary nameserver>
nameserver <IP address of your secondary nameserver>

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

<IP address of the nameserver>（ネームサーバーの IP アドレス）の部分には、 DNS が割り振る適切な IP アドレスを記述します。 IP アドレスの設定は複数行う場合もあります。（代替構成を必要とするなら二次サーバーを設けることでしょう。）一つのサーバーのみで十分な場合は、二つめの nameserver の行は削除します。 ローカルネットワークにおいてはルーターの IP アドレスを設定することになるでしょう。

第8章 LFS システムのブート設定

8.1. はじめに

ここからは LFS システムをブート可能にしていきます。 この章では fstab ファイルを作成し、LFS システムのカーネルを構築します。 また GRUB のブートローダをインストールして LFS システムの起動時にブートローダを選択できるようにします。

8.2. /etc/fstab ファイルの生成

/etc/fstab ファイルは、種々のプログラムがファイルシステムのマウント状況を確認するために利用するファイルです。 ファイルシステムがデフォルトでどこにマウントされ、それがどういう順序であるか、マウント前に（整合性エラーなどの）チェックを行うかどうか、という設定が行われます。 新しいファイルシステムに対する設定は以下のようにして生成します。

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

# file system  mount-point  type   options         dump  fsck
#                                                       order

/dev/<xxx>     /            <fff>  defaults        1     1
/dev/<yyy>     swap         swap   pri=1           0     0
proc           /proc        proc   defaults        0     0
sysfs          /sys         sysfs  defaults        0     0
devpts         /dev/pts     devpts gid=4,mode=620  0     0
tmpfs          /dev/shm     tmpfs  defaults        0     0
# End /etc/fstab
EOF
```

<xxx>、 <yyy>、 <fff> の部分はシステムに合わせて正しい記述に書き換えてください。 例えば hda2、 hda5、 ext3 といったものです。 上のファイルの6行分の記述内容の詳細は man 5 fstab により確認してください。

マウントポイント /dev/shm は tmpfs ファイルシステムを指し示すもので POSIX 共有メモリ（POSIX-shared memory）を利用するためのものです。 この共有メモリを正しく動作させるためには、これをサポートする機能をカーネルに組み入れておく必要があります。（詳しくは次節にて説明します。）POSIX 共有ライブラリを利用するソフトウェアは、今のところは非常に少ないことを覚えておいてください。 したがってマウントポイント /dev/shm は設定しなくても構いません。 詳細については、カーネルのソース内にある Documentation/filesystems/tmpfs.txt を参照してください。

MS-DOS や Windows において利用されるファイルシステム（例えば vfat、ntfs、smbfs、cifs、iso9660、udf）では、ファイル名称内に用いられた非アスキー文字を正しく認識させるために、マウントオプションとして 「iocharset」 を指定することが必要となります。 オプションに設定する値は利用するロケールとすることが必要で、カーネルが理解できる形でなければなりません。 またこれを動作させるために、対応するキャラクタセット定義（File systems ->Native Language Support にあります）をカーネルに組み入れるか、モジュールとしてビルドすることが必要です。 vfat や smbfs ファイルシステムを用いるなら、さらに 「codepage」 オプションも必要です。 このオプションには、国情報に基づいて MS-DOS にて用いられるコードページ番号をセットします。 例えば USB フラッシュドライブをマウントし ru_RU.KOI8-R をセットするユーザーであれば /etc/fstab ファイルの設定は以下のようになります。

```
noauto,user,quiet,showexec,iocharset=koi8r,codepage=866
```

ru_RU.UTF-8 をセットするなら以下のように変わります。

```
noauto,user,quiet,showexec,iocharset=utf8,codepage=866
```

注記

後者の設定では、カーネルが以下のようなメッセージを出力します。

```
FAT: utf8 is not a recommended IO charset for FAT filesystems,
    filesystem will be case sensitive!
```

否定的な設定を勧めるメッセージですが、これは無視して構いません。 「iocharset」 オプションに他の設定を行ったとしても UTF-8 ロケールでは結局はファイル名の表示を正しく処理できないためです。

ファイルシステムによっては codepage と iocharset のデフォルト値をカーネルにおいて設定することもできます。カーネルにおいて対応する設定は「Default NLS Option」(CONFIG_NLS_DEFAULT)、「Default Remote NLS Option」(CONFIG_SMB_NLS_DEFAULT)、「Default codepage for FAT」(CONFIG_FAT_DEFAULT_CODEPAGE)、「Default iocharset for FAT」(CONFIG_FAT_DEFAULT_IOCHARSET)です。 なお ntfs ファイルシステムに対しては、カーネルのコンパイル時に設定する項目はありません。

特定のハードディスクにおいて ext3 ファイルシステムでの電源供給不足時の信頼性を向上させることができます。 これは /etc/fstab での定義においてマウントオプション barrier=1 を指定します。 ハードディスクがこのオプションをサポートしているかどうかは hdparm を実行することで確認できます。 例えば以下のコマンドを実行します。

```
hdparm -I /dev/sda | grep NCQ
```

何かが出力されたら、このオプションがサポートされていることを意味します。

論理ボリュームマネージャ (Logical Volume Management; LVM) に基づいたパーティションでは barrier オプションは利用できません。

8.3. Linux-2.6.35.4

Linux パッケージは Linux カーネルを提供します。

概算ビルド時間: 1.5 - 5.0 SBU
必要ディスク容量: 450 - 500 MB

8.3.1. カーネルのインストール

カーネルの構築は、カーネルの設定、コンパイル、インストールの順に行っていきます。 本書が行っているカーネル設定の方法以外については、カーネルソースツリー内にある README ファイルを参照してください。

コンパイルするための準備として以下のコマンドを実行します。

```
make mrproper
```

これによりカーネルソースが完全にクリーンなものになります。 カーネル開発チームは、カーネルコンパイルするなら、そのたびにこれを実行することを推奨しています。 tar コマンドにより伸張しただけのソースではクリーンなものにはなりません。

メニュー形式のインターフェースによりカーネルを設定します。 カーネルの設定方法に関する一般的な情報が http://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt にあるので参照してください。 BLFS では LFS が取り扱わない各種パッケージに対して、必要となるカーネル設定項目を説明しています。 http://www.linuxfromscratch.org/blfs/view/svn/longindex.html#kernel-config-index を参照してください。

```
make LANG=<host_LANG_value> LC_ALL= menuconfig
```

make パラメータの意味：

LANG=<host_LANG_value> LC_ALL=
これはホストのロケール設定を指示するものです。 この設定は UTF-8 での表示設定がされたテキストコンソールにて、 menuconfig の ncurses による行表示を適切に行うために必要となります。
<host_LANG_value> の部分は、ホストの $LANG 変数の値に置き換えてください。 ホストにてその値が設定されていない場合は $LC_ALL あるいは $LC_CTYPE の値を設定してください。

上のコマンドではなく、状況によっては make oldconfig を実行することが適当な場合もあります。 詳細についてはカーネルソース内の README ファイルを参照してください。

カーネル設定は行わずに、ホストシステムにあるカーネル設定ファイル .config をコピーして利用することもできます。 そのファイルが存在すればの話です。 その場合は linux-2.6.35.4 ディレクトリにそのファイルをコピーしてください。 もっともこのやり方はお勧めしません。 設定項目をメニューから探し出して、カーネル設定を一から行っていくことが望ましいことです。

カーネルイメージとモジュールをコンパイルします。

```
make
```

カーネルモジュールを利用する場合 /etc/modprobe.d ディレクトリ内での設定を必要とします。 モジュールやカーネル設定に関する情報は 7.9. 「LFS システムにおけるデバイスとモジュールの扱い」 や linux-2.6.35.4/Documentation ディレクトリにあるカーネルドキュメントを参照してください。 また modprobe.conf(5) も有用です。

カーネル設定においてモジュールを利用することにした場合、モジュールをインストールします。

```
make modules_install
```

カーネルのコンパイルが終わったら、インストールの完了に向けてあと少し作業を行います。 /boot ディレクトリにいくつかのファイルをコピーします。

カーネルイメージへのパスは、利用しているプラットフォームによってさまざまです。 そのファイル名は、好みにより自由に変更して構いません。 ただし vmlinux という語は必ず含めてください。 これにより、次節で説明するブートプロセスを自動的に設定するために必要なことです。 以下のコマンドは x86 アーキテクチャの場合の例です。

```
cp -v arch/x86/boot/bzImage /boot/vmlinux-2.6.35.4-lfs-6.7
```

System.map はカーネルに対するシンボルファイルです。 このファイルはカーネル API の各関数のエントリポイントをマッピングしています。 同様に実行中のカーネルのデータ構成のアドレスを保持します。 このファイルは、カーネルに問題があった場合にその状況を調べる手段として利用できます。 マップファイルをインストールするには以下を実行します。

```
cp -v System.map /boot/System.map-2.6.35.4
```

カーネル設定ファイル .config は、上で実行した make menuconfig によって生成されます。 このファイル内には、今コンパイルしたカーネルの設定項目の情報がすべて保持されています。 将来このファイルを参照する必要が出てくるかもしれないため、このファイルを保存しておきます。

```
cp -v .config /boot/config-2.6.35.4
```

Linux カーネルのドキュメントをインストールします。

```
install -d /usr/share/doc/linux-2.6.35.4
cp -r Documentation/* /usr/share/doc/linux-2.6.35.4
```

カーネルのソースディレクトリは所有者が root ユーザーになっていません。 我々は chroot 環境内の root ユーザーとなってパッケージを展開してきましたが、展開されたファイル類はパッケージ開発者が用いていたユーザー ID、グループ ID が適用されています。 このことは普通はあまり問題になりません。 というのもパッケージをインストールした後のソースファイルは、たいていは削除するからです。 一方 Linux のソースファイルは、削除せずに保持しておくことがよく行われます。 このことがあるため開発者の用いたユーザーIDが、インストールしたマシン内の誰かの ID に割り当たった状態となりえます。 その人はカーネルソースを自由に書き換えてしまう権限を持つことになるわけです。

カーネルのソース・ファイルを保持しておくつもりなら linux-2.6.35.4 ディレクトリにおいて chown -R 0:0 を実行しておいてください。 これによりそのディレクトリの所有者は root ユーザーとなります。

警告

カーネルを説明する書の中には、カーネルのソースディレクトリに対してシンボリックリンク /usr/src/linux の生成を勧めているものがあります。 これはカーネル 2.6 系以前におけるものであり LFS システム上では生成してはなりません 。 ベースとなる LFS システムを構築し、そこに新たなパッケージを追加していこうとした際に、そのことが問題となるからです。

警告

さらに include ディレクトリにあるヘッダファイルは、必ず Glibc のコンパイルによって得られるものでなければならず、つまりは Linux カーネルの tarball によって提供されるものでなければなりません。 したがってカーネルヘッダによって上書きされてしまうのは避けなければなりません。

8.3.2. Linux モジュールのロード順の設定

USB ドライバをモジュールとして構築した場合は /etc/modprobe.d/usb.conf ファイルを生成する必要があります。 USB ドライバには ehci_hcd、ohci_hcd、uhci_hcd があります。 これらのロード順は正しく行う必要があります。 ehci_hcd は ohci_hcd や uhci_hcd よりも先にロードしなければなりません。 これを行わないとブート時に警告メッセージが出力されます。

以下のコマンドを実行して /etc/modprobe.d/usb.conf ファイルを生成します。

```
install -v -m755 -d /etc/modprobe.d
cat > /etc/modprobe.d/usb.conf << "EOF"
# Begin /etc/modprobe.d/usb.conf

install ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true
install uhci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i uhci_hcd ; true

# End /etc/modprobe.d/usb.conf
EOF
```

8.3.3. Linux の構成

インストールファイル: config-2.6.35.4, vmlinux-2.6.35.4-lfs-6.7-2.6.35.4, System.map-2.6.35.4
インストールディレクトリ: /lib/modules, /usr/share/doc/linux-2.6.35.4

概略説明

config-2.6.35.4　　カーネルの設定をすべて含みます。

vmlinux-2.6.35.4-lfs-6.7	Linux システムのエンジンです。 コンピュータを起動した際には、オペレーティングシステム内にて最初にロードされるものです。 カーネルはコンピュータのハードウェアを構成するあらゆるコンポーネントを検知して初期化します。 そしてそれらのコンポーネントをツリー階層のファイルとして、ソフトウェアが利用できるようにします。 ただひとつの CPU からマルチタスクを処理するマシンとして、あたかも多数のプログラムが同時稼動しているように仕向けます。
System.map-2.6.35.4	アドレスとシンボルのリストです。 カーネル内のすべての関数とデータ構成のエントリポイントおよびアドレスを示します。

8.4. GRUB を用いたブートプロセスの設定

8.4.1. はじめに

ブートローディングというものは複雑に入り組んでいます。 そこで注意すべき点を順に説明していきます。 ご自身が現時点で利用しているブートローダーが何であるのか、あるいはブートを必要とする他のオペレーティングシステムがハードドライブに存在しているかどうかについては、よく確認しておいてください。 またコンピュータが利用不能に(ブート不能に)なってしまうことに備えて、コンピュータを「復旧(resucue)」するブートディスクの生成も忘れないでください。

これ以降の手順にて、 GRUB に関する特別なファイル類をハードドライブ上の所定ディレクトリに書き出します。 ここではバックアップ目的で GRUB のブートフロッピーディスケットを生成しておくことを強く推奨します。 空のフロッピーディスケットを挿入して以下を実行してください。

```
cd /tmp
grub-mkrescue --output=grub-img.iso
dd if=grub-img.iso of=/dev/fd0 bs=1440 count=1
```

または、ホストシステムにある CD ライティングツールを使えば、ブート CD を作ることもできます。 その場合は grub-img.iso を空の CD に書き込みます。

GRUB ではドライブやパーティションに対して(hdn,m)といった書式の命名法を採用しています。 n はドライブ番号であり m はパーティション番号です。 いずれもゼロから数え始めます。 ただしパーティションの番号は普通は1から数え始め、拡張パーティションは5から数え始めます。 かつてのバージョンでは共にゼロから数え始めていましたが、今はそうではないので注意してください。 例えばパーティション sda1 は(hd0,1)となり、パーティション sdb3 は(hd1,3)となります。 Linux システムでの取り扱いとは違って GRUB では CD-ROM ドライブをハードドライブとしては扱いません。 例えば CD が hdb であり２番めのハードドライブが hdc であった場合、２番めのハードドライブは(hd1)と表記されます。

ディスクデバイスを GRUB がどのような名称で取り扱うかを確認する場合は以下を実行してください。

```
grub-mkdevicemap --device-map=device.map
cat device.map
```

ブートパーティションをどこにするかは各人に委ねられていて、それによって設定方法が変わります。 推奨される1つの手順としては、ブートパーティションとして独立した小さな(100MB 程度のサイズの)パーティションを設けることです。 こうしておくと、この後に LFS であろうが商用ディストリビューションであろうが、システム導入する際に同一のブートファイルを利用することが可能です。 つまりどのようなブートシステムからでもアクセスが可能となります。 この方法をとるなら、新たなパーティションをマウントした上で、現在 /boot ディレクトリにある全ファイルを(例えば前節にてビルドした Linux カーネルも)新しいパーティションに移動させる必要があります。 そしていったんパーティションをアンマウントし、再度 /boot としてマウントしなおすことになります。 これを行った後は/etc/fstab を適切に書き換えてください。

現時点での LFS パーティションでも問題なく動作します。 ただし複数システムを取り扱うための設定は、より複雑になります。

8.4.2. 設定作業

ここまでの情報に基づいて、ルートパーティションの名称を(あるいはブートパーティションを別パーティションとするならそれも含めて)決定します。 以下では例として、ルートパーティション(あるいは別立てのブートパーティション)が sda2 であるとします。

以下を実行して GRUB ファイル類を /boot/grub にインストールします。

```
grub-install --grub-setup=/bin/true /dev/sda
```

ここでは --grub-setup=/bin/true を指定して、マスタブートレコード(Master Boot Record; MBR)への書き込みを行わないようにしています。 書き込みを行ってしまってから元に戻すのは大変な作業になります。 そこでここでは事前にテストを行う方法をとります。

/boot/grub/grub.cfg ファイルを生成します。

```
grub-mkconfig -o /boot/grub/grub.cfg
```

grub-mkconfig コマンドは /etc/grub.d/ ディレクトリにあるファイル類を利用して、上のファイルの内容を決定します。 上の設定ファイルは以下のようなものです。

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by /usr/sbin/grub-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#

### BEGIN /etc/grub.d/00_header ###
set default=0
set timeout=5
### END /etc/grub.d/00_header ###

### BEGIN /etc/grub.d/10_linux ###
menuentry "GNU/Linux, Linux 2.6.35.4-lfs-6.7" {
        insmod ext2
        set root=(hd0,2)
        search --no-floppy --fs-uuid --set 915852a7-859e-45a6-9ff0-d3ebfdb5cea2
        linux   /boot/vmlinux-2.6.35.4-lfs-6.7 root=/dev/sda2 ro
}
menuentry "GNU/Linux, Linux 2.6.35.4-lfs-6.7" (recovery mode)" {
        insmod ext2
        set root=(hd0,2)
        search --no-floppy --fs-uuid --set 915852a7-859e-45a6-9ff0-d3ebfdb5cea2
        linux   /boot/vmlinux-2.6.35.4-lfs-6.7 root=/dev/sda2 ro single
}
menuentry "GNU/Linux, Linux 2.6.28-11-server" {
        insmod ext2
        set root=(hd0,2)
        search --no-floppy --fs-uuid --set 6b4c0339-5501-4a85-8351-e398e5252be8
        linux   /boot/vmlinuz-2.6.28-11-server \
                root=UUID=6b4c0339-5501-4a85-8351-e398e5252be8 ro
        initrd  /boot/initrd.img-2.6.28-11-server
}
menuentry "GNU/Linux, Linux 2.6.28-11-server (recovery mode)" {
        insmod ext2
        set root=(hd0,2)
        search --no-floppy --fs-uuid --set 6b4c0339-5501-4a85-8351-e398e5252be8
        linux   /boot/vmlinuz-2.6.28-11-server \
                root=UUID=6b4c0339-5501-4a85-8351-e398e5252be8 ro single
        initrd  /boot/initrd.img-2.6.28-11-server
}
### END /etc/grub.d/10_linux ###

### BEGIN /etc/grub.d/30_os-prober ###
### END /etc/grub.d/30_os-prober ###

### BEGIN /etc/grub.d/40_custom ###
# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.
### END /etc/grub.d/40_custom ###
```

注記

- このファイルを無闇に編集するのは避けるべきですが、 grub-mkconfig コマンドを再実行しない限りは、編集作業を行っても構いません。
- search と書かれた行は LFS システムにとっては意味がありません。 そこに示されるコマンドは GRUB の内部変数をセットし、カーネルイメージを検索するためのものです。 set root コマンドの記述があれば、同等の機能が実現され、検索のオーバーヘッドを抑えることができます。
- set root と insmod ext2 の２つのコマンドは menuentry のセクションの外に記述することもできます。そうすると本ファイル内のすべてのセクションに適用されるものとなります。 したがって個々のセクションは、例えば以下のように単純な記述とすることもできます。

```
menuentry "Linux 2.6.35.4-lfs-6.7" {
linux   /boot/vmlinux-2.6.35.4-lfs-6.7 root=/dev/sda2 ro
}
```

- カーネルに対して UUID を指定する場合は、初期 RAM ディスク（initial ram disk; initrd）を必要としますが、 LFS ではこれを構築しません。
- /boot パーティションが独立したパーティションとして設けられている場合は linux と initrd の行において /boot の記述は取り除く必要があります。
- 上のサンプル記述では /boot に Ubuntu のカーネルファイルがインストールされている例を含んでいます。

8.4.3. 設定のテスト

GRUB のコアイメージ（core image）もマルチブートカーネル（Multiboot kernel）です。 したがって GRUB Legacy を既にインストール済であるなら、それまでの古いブートローダーを用いて新たな GRUB-1.98 をロードすることが可能です。 具体的な方法としては、まず chroot 環境からいったん抜け出た上で再度入り直し、本節の残りの作業を進めます。

```
/sbin/reboot
...
grub> root (hd0,1)
grub> kernel /boot/grub/core.img
grub> boot
```

上に示しているコマンドは GRUB Legacy であるものとして説明しています。 この時点で GRUB は（GRUB Legacy と非常に似た）プロンプトを表示します。 そこではさまざまな入力を行ったり、grub.cfg ファイルに定められているシステムを起動することもできます。

8.4.4. マスタブートレコードへの書き込み

上で示したように GRUB の設定に対するテストを終えたら、再び chroot 環境に入ります。

警告

以下に示すコマンドを実行すると、現在のブートローダーを上書きします。 上書きするのが不適当であるならコマンドを実行しないでください。 例えばマスタブートレコード（Master Boot Record; MBR）を管理するサードパーティ製のブートマネージャソフトウェアを利用している場合などがこれに該当します。

以下により MBR を書き換えます。

```
grub-setup '<DEVICE>'
```

DEVICE の部分はブートディスクに応じて書き換えてください。 通常は '(hd0)' あるいは /dev/sda となるはずです。(hd0) を指定する場合は、カッコの文字をバックスラッシュによりエスケープするか、シングルクォートで囲むようにしてください。 そうしておかないと、サブシェルを表わすものとして解釈されてしまうからです。

このプログラムは以下に示すデフォルト値を用います。 ここまでの手順において本書とは異なる方法をとっている場合は、適切に修正してください。

- ブートイメージ - boot.img
- コアイメージ - core.img

- ディレクトリ - /boot/grub
- デバイスマップ - device.map
- デフォルトルート設定 - 自動推測

注記

ルート設定は grub.cfg ファイル内にて 'set root' の指定がない場合のデフォルト値です。 これは、カーネルや他の関連ファイルが検出するパーティションとなり、 'linux' の設定行内にあるパラメータ 'root=' での設定内容とは異なります。 'root=' での設定は、カーネルが '/' としてマウントしたパーティションを意味します。 上に示した grub.cfg のサンプルでは、どちらも /dev/sda2 に設定していますが、ブートパーティションを別に用意している場合は設定値が異なることとなります。

第9章 作業終了

9.1. 作業終了

できました！ LFS システムのインストール終了です。 あなたの輝かしいカスタムメイドの Linux システムが完成したことでしょう。

/etc/lfs-release というファイルをここで作成することにします。 このファイルを作っておけば、どのバージョンの LFS をインストールしたのか、すぐに判別できます。（もしあなたが質問を投げた時には、我々もすぐに判別できることになります。）以下のコマンドによりこのファイルを生成します。

```
echo 6.7 > /etc/lfs-release
```

9.2. ユーザー登録

これにより本書の作業は終了です。 LFS ユーザー登録を行ってカウンタを取得しますか？ 以下のページ http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi にて氏名と LFS バージョンを登録して下さい。

それではシステムの再起動を行ないましょう。

9.3. システムの再起動

ソフトウェアのインストールがすべて完了しました。 ここでコンピュータを再起動しますが、いくつか注意しておいて下さい。 本書を通じて構築したシステムは最小限のものです。 これ以降に様々なことを繰り広げていくには、機能が不足しているはずです。 もうしばらくは今までと同じように chroot 環境を利用して BLFS ブックからいくつかのパッケージをインストールしていきましょう。 その後のリブートにより新しい LFS システムを起動すれば、より一層、満足できる環境を得ることになるはずです。 例えば Lynx のようなテキストウェブブラウザをインストールすれば、仮想端末上で BLFS ブックを参照でき、同時にパッケージのビルドを行っていくことができます。 GPM パッケージを導入すれば、仮想端末上にてコピー・ペースト作業を行うことができます。 またネットワーク接続にあたって固定 IP アドレスが不適当である場合には Dhcpcd や PPP といったパッケージをインストールしておくのが良いでしょう。

さあよろしいですか。 新しくインストールした LFS システムの再起動を行いましょう。 まずは chroot 環境から抜けます。

```
logout
```

仮想ファイルシステムをアンマウントします。

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

LFS ファイルシステムもアンマウントします。

```
umount -v $LFS
```

複数のパーティションを生成していた場合は、以下のようにして複数パーティションをアンマウントします。 メインのパーティションのアンマウントはその後に行います。

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

以下のようにしてシステムを再起動します。

```
shutdown -r now
```

これまでの作業にて GRUB ブートローダが設定されているはずです。 そのメニューには LFS 6.7 を起動するためのメニュー項目があるはずです。

再起動が無事行われ LFS システムを使うことができます。 必要に応じてさらなるソフトウェアをインストールしていってください。

9.4. 今度は何?

本書をお読み頂き、ありがとうございます。 本書が皆さんにとって有用なものとなり、システムの構築方法について十分に学んで頂けたものと思います。

LFS システムをインストールしたら 「次は何を？」 とお考えになるかもしれません。 その質問に答えるために以下に各種の情報をまとめます。

- 保守

  あらゆるソフトウェアにおいて、バグやセキュリティの情報は日々報告されています。 LFS システムはソースコードからコンパイルしていますので、そのような報告を見逃さずにおくことは皆さんの仕事となります。 そのような報告をオンラインで提供する情報の場がありますので、いくつかを以下に示しましょう。

  - Freshmeat.net (http://freshmeat.net/)

    Freshmeat は、システムにインストールされているパッケージの新しいバージョンが提供されると、それを (電子メールで) 通知してくれます。

  - CERT (Computer Emergency Response Team)

    CERT にはメーリングリストがあり、数々のオペレーティングシステムやアプリケーションにおけるセキュリティ警告を公開しています。 購読に関する情報は http://www.us-cert.gov/cas/signup.html を参照してください。

  - バグトラック (Bugtraq)

    バグトラックは、完全公開のコンピュータセキュリティに関するメーリングリストです。 これは新たに発見されたセキュリティに関する問題を公開しています。 また時には、その問題を解消するフィックス情報も提供してくれます。 購読に関する情報は http://www.securityfocus.com/archive を参照してください。

- Beyond Linux From Scratch

  Beyond Linux From Scratch ブックは、 LFS ブックが取り扱うソフトウェアの範囲を超えて、数多くのソフトウェアをインストールする手順を示しています。 BLFS プロジェクトは以下にあります。 http://www.linuxfromscratch.org/blfs/ .

- LFS ヒント (LFS Hints)

  LFS ヒントは有用なドキュメントを集めたものです。 LFS コミュニティのボランティアによって投稿されたものです。 それらのヒントは http://www.linuxfromscratch.org/hints/list.html にて参照することができます。

- メーリングリスト

  皆さんにも参加して頂ける LFS メーリングリストがあります。 何かの助けが必要になったり、最新の開発を行いたかったり、あるいはプロジェクトに貢献したいといった場合に、参加して頂くことができます。 詳しくは 第1章 - メーリングリスト を参照してください。

- Linux ドキュメントプロジェクト (The Linux Documentation Project; TLDP)

  Linux ドキュメントプロジェクトの目指すことは Linux のドキュメントに関わる問題を共同で取り組むことです。 TLDP ではハウツー (HOWTO)、ガイド、man ページを数多く提供しています。 以下のサイトにあります。 http://www.tldp.org/

第IV部 付録

付録 A. 略語と用語

日本語訳情報

本節における日本語訳は、訳語が一般的に普及していると思われるものは、その訳語とカッコ書き内に原語を示します。 逆に訳語に適当なものがないと思われるものは、無理に訳出せず原語だけを示すことにします。この判断はあくまで訳者によるものであるため、不適切・不十分な個所についてはご指摘ください。

ABI	アプリケーション バイナリ インターフェース (Application Binary Interface)
ALFS	Automated Linux From Scratch
ALSA	Advanced Linux Sound Architecture
API	アプリケーション プログラミング インターフェース (Application Programming Interface)
ASCII	American Standard Code for Information Interchange
BIOS	ベーシック インプット/アウトプット システム; バイオス (Basic Input/Output System)
BLFS	Beyond Linux From Scratch
BSD	Berkeley Software Distribution
chroot	ルートのチェンジ (change root)
CMOS	シーモス (Complementary Metal Oxide Semiconductor)
COS	Class Of Service
CPU	中央演算処理装置 (Central Processing Unit)
CRC	巡回冗長検査 (Cyclic Redundancy Check)
CVS	Concurrent Versions System
DHCP	ダイナミック ホスト コンフィギュレーション プロトコル (Dynamic Host Configuration Protocol)
DNS	ドメインネームサービス (Domain Name Service)
EGA	Enhanced Graphics Adapter
ELF	Executable and Linkable Format
EOF	ファイルの終端 (End of File)
EQN	式 (equation)
EVMS	Enterprise Volume Management System
ext2	second extended file system
ext3	third extended file system
ext4	fourth extended file system
FAQ	よく尋ねられる質問 (Frequently Asked Questions)
FHS	ファイルシステム階層標準 (Filesystem Hierarchy Standard)
FIFO	ファーストイン、ファーストアウト (First-In, First Out)
FQDN	完全修飾ドメイン名 (Fully Qualified Domain Name)
FTP	ファイル転送プロトコル (File Transfer Protocol)
GB	ギガバイト (gigabytes)
GCC	GNU コンパイラ コレクション (GNU Compiler Collection)
GID	グループ識別子 (Group Identifier)
GMT	グリニッジ標準時 (Greenwich Mean Time)
GPG	GNU Privacy Guard
HTML	ハイパーテキスト マークアップ ランゲージ (Hypertext Markup Language)
IDE	Integrated Drive Electronics
IEEE	Institute of Electrical and Electronic Engineers
IO	入出力 (Input/Output)
IP	インターネット プロトコル (Internet Protocol)

IPC　プロセス間通信（Inter-Process Communication）
IRC　インターネット リレイ チャット（Internet Relay Chat）
ISO　国際標準化機構（International Organization for Standardization）
ISP　インターネット サービス プロバイダ（Internet Service Provider）
KB　キロバイト（kilobytes）
LED　発光ダイオード（Light Emitting Diode）
LFS　Linux From Scratch
LSB　Linux Standard Base
MB　メガバイト（megabytes）
MBR　マスタ ブート レコード（Master Boot Record）
MD5　Message Digest 5
NIC　ネットワーク インターフェース カード（Network Interface Card）
NLS　Native Language Support
NNTP　Network News Transport Protocol
NPTL　Native POSIX Threading Library
OSS　Open Sound System
PCH　プリコンパイル済みヘッダ（Pre-Compiled Headers）
PCRE　Perl Compatible Regular Expression
PID　プロセス識別子（Process Identifier）
PLFS　Pure Linux From Scratch
PTY　仮想端末（pseudo terminal）
QA　品質保証（Quality Assurance）
QOS　クオリティ オブ サービス（Quality Of Service）
RAM　ランダム アクセス メモリ（Random Access Memory）
RPC　リモート プロシージャ コール（Remote Procedure Call）
RTC　リアルタイムクロック（Real Time Clock）
SBU　標準ビルド時間（Standard Build Unit）
SCO　サンタ クルズ オペレーション社（The Santa Cruz Operation）
SGR　Select Graphic Rendition
SHA1　Secure-Hash Algorithm 1
SMP　対称型マルチプロセッサ（Symmetric Multi-Processor）
TLDP　The Linux Documentation Project
TFTP　Trivial File Transfer Protocol
TLS　スレッド ローカル ストレージ（Thread-Local Storage）
UID　ユーザー識別子（User Identifier）
umask　user file-creation mask
USB　ユニバーサル シリアル バス（Universal Serial Bus）
UTC　協定世界時（Coordinated Universal Time）
UUID　汎用一意識別子（Universally Unique Identifier）
VC　仮想コンソール（Virtual Console）
VGA　ビデオ グラフィックス アレイ（Video Graphics Array）
VT　仮想端末（Virtual Terminal）

付録 B. 謝辞

Linux From Scratch プロジェクトへ貢献して下さった以下の方々および組織団体に感謝致します。

- Gerard Beekmans <gerard@linuxfromscratch.org> - LFS 構築者、LFS プロジェクトリーダー
- Matthew Burgess <matthew@linuxfromscratch.org> - LFS プロジェクトリーダー、LFS テクニカルライター/編集者
- Bruce Dubbs <bdubbs@linuxfromscratch.org> - LFS リリース管理者、LFS テクニカルライター/編集者
- Jim Gifford <jim@linuxfromscratch.org> - CLFS プロジェクト共同リーダー
- Bryan Kadzban <bryan@linuxfromscratch.org> - LFS テクニカルライター
- Randy McMurchy <randy@linuxfromscratch.org> - BLFS プロジェクトリーダー、LFS 編集者
- DJ Lucas <dj@linuxfromscratch.org> - LFS、BLFS 編集者
- Ken Moffat <ken@linuxfromscratch.org> - LFS、CLFS 編集者
- Ryan Oliver <ryan@linuxfromscratch.org> - CLFS プロジェクト共同リーダー
- この他に数多くの方々にも協力頂きました。 皆さまには LFS や BLFS などのメーリングリストにて、 提案、ブック内容のテスト、バグ報告、作業指示、パッケージインストールの経験談などを通じて、本ブック製作にご協力頂きました。

翻訳者

- Manuel Canales Esparcia <macana@macana-es.com> - スペインの LFS 翻訳プロジェクト
- Johan Lenglet <johan@linuxfromscratch.org> - フランスの LFS 翻訳プロジェクト
- Anderson Lizardo <lizardo@linuxfromscratch.org> - ポルトガルの LFS 翻訳プロジェクト
- Thomas Reitelbach <tr@erdfunkstelle.de> - ドイツの LFS 翻訳プロジェクト

ミラー管理者

北米のミラー

- Scott Kveton <scott@osuosl.org> - lfs.oregonstate.edu ミラー
- William Astle <lost@l-w.net> - ca.linuxfromscratch.org ミラー
- Eujon Sellers <jpolen@rackspace.com> - lfs.introspeed.com ミラー
- Justin Knierim <tim@idge.net> - lfs-matrix.net ミラー

南米のミラー

- Manuel Canales Esparcia <manuel@linuxfromscratch.org> - lfsmirror.lfs-es.info ミラー
- Luis Falcon <Luis Falcon> - torredehanoi.org ミラー

ヨーロッパのミラー

- Guido Passet <guido@primerelay.net> - nl.linuxfromscratch.org ミラー
- Bastiaan Jacques <baafie@planet.nl> - lfs.pagefault.net ミラー
- Sven Cranshoff <sven.cranshoff@lineo.be> - lfs.lineo.be ミラー
- Scarlet Belgium - lfs.scarlet.be ミラー
- Sebastian Faulborn <info@aliensoft.org> - lfs.aliensoft.org ミラー
- Stuart Fox <stuart@dontuse.ms> - lfs.dontuse.ms ミラー
- Ralf Uhlemann <admin@realhost.de> - lfs.oss-mirror.org ミラー
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> - at.linuxfromscratch.org ミラー
- Fredrik Danerklint <fredan-lfs@fredan.org> - se.linuxfromscratch.org ミラー
- Franck <franck@linuxpourtous.com> - lfs.linuxpourtous.com ミラー
- Philippe Baqué <baque@cict.fr> - lfs.cict.fr ミラー
- Vitaly Chekasin <gyouja@pilgrims.ru> - lfs.pilgrims.ru ミラー

- Benjamin Heil <kontakt@wankoo.org> - lfs.wankoo.org ミラー

アジアのミラー

- Satit Phermsawang <satit@wbac.ac.th> - lfs.phayoune.org ミラー
- Shizunet Co.,Ltd. <info@shizu-net.jp> - lfs.mirror.shizu-net.jp ミラー
- Init World <http://www.initworld.com/> - lfs.initworld.com ミラー

オーストラリアのミラー

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

以前のプロジェクトチームメンバー

- Christine Barczak <theladyskye@linuxfromscratch.org> - LFS ブック編集者
- Archaic <archaic@linuxfromscratch.org> - LFS テクニカルライター/編集者、 HLFS プロジェクトリーダー、 BLFS 編集者、 ヒントプロジェクトとパッチプロジェクトの管理者
- Nathan Coulson <nathan@linuxfromscratch.org> - LFS-ブートスクリプトの管理者
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- Jeroen Coumans <jeroen@linuxfromscratch.org> - ウェブサイト開発者、 FAQ 管理者
- Manuel Canales Esparcia <manuel@linuxfromscratch.org> - LFS/BLFS/HLFS の XML と XSL の管理者
- Alex Groenewoud - LFS テクニカルライター
- Marc Heerdink
- Jeremy Huntwork <jhuntwork@linuxfromscratch.org> - LFS テクニカルライター、LFS LiveCD 管理者
- Mark Hymers
- Seth W. Klein - FAQ 管理者
- Nicholas Leippe <nicholas@linuxfromscratch.org> - Wiki 管理者
- Anderson Lizardo <lizardo@linuxfromscratch.org> - ウェブサイトのバックエンドスクリプトの管理者
- Dan Nicholson <dnicholson@linuxfromscratch.org> - LFS/BLFS 編集者
- Alexander E. Patrakov <alexander@linuxfromscratch.org> - LFS テクニカルライター、 LFS 国際化に関する編集者、 LFS Live CD 管理者
- Simon Perreault
- Scot Mc Pherson <scot@linuxfromscratch.org> - LFS NNTP ゲートウェイ管理者
- Greg Schafer <gschafer@zip.com.au> - LFS テクニカルライター、 次世代 64 ビット機での構築手法の開発者
- Jesse Tie-Ten-Quee - LFS テクニカルライター
- James Robertson <jwrober@linuxfromscratch.org> - Bugzilla 管理者
- Tushar Teredesai <tushar@linuxfromscratch.org> - BLFS ブック編集者、 ヒントプロジェクト・パッチプロジェクトのリーダー
- Jeremy Utley <jeremy@linuxfromscratch.org> - LFS テクニカルライター、 Bugzilla 管理者、 LFS-ブートスクリプト管理者
- Zack Winkles <zwinkles@gmail.com> - LFS テクニカルライター

付録 C. パッケージの依存関係

LFS にて構築するパッケージはすべて、他のいくつかのパッケージに依存していて、それらがあって初めて適切にインストールができます。 パッケージの中には互いに依存し合っているものもあります。 つまり一つめのパッケージが二つめのパッケージに依存しており、二つめが実は一つめのパッケージにも依存しているような例です。 こういった依存関係があることから LFS においてパッケージを構築する順番は非常に重要なものとなります。 本節は LFS にて構築する各パッケージの依存関係を示すものです。

ビルドするパッケージの個々には、3種類あるいは4種類の依存関係を示しています。 1つめは対象パッケージをコンパイルしてビルドするために必要となるパッケージです。 2つめは一つめのものに加えて、テストスイートを実行するために必要となるパッケージです。 3つめは対象パッケージをビルドし、最終的にインストールするために必要となるパッケージです。 たいていの場合、それらのパッケージに含まれているスクリプトが、実行モジュールへのパスを固定的に取り扱っています。 所定の順番どおりにパッケージのビルドを行わないと、最終的にインストールされるシステムにおいて、スクリプトの中に /tools/bin/[実行モジュール] といったパスが含まれてしまうことになりかねません。これは明らかに不適切なことです。

依存関係として4つめに示すのは任意のパッケージであり LFS では説明していないものです。 しかし皆さんにとっては有用なパッケージであるはずです。 それらのパッケージは、さらに別のパッケージを必要としていたり、互いに依存し合っていることがあります。 そういった依存関係があるため、それらをインストールする場合には、LFS をすべて仕上げた後に再度 LFS 内のパッケージを再構築する方法をお勧めします。 再インストールに関しては、たいていは BLFS にて説明しています。

Autoconf

インストール依存パッケージ:	Bash, Coreutils, Grep, M4, Make, Perl, Sed, Texinfo
テストスイート依存パッケージ:	Automake, Diffutils, Findutils, GCC, Libtool
事前インストールパッケージ:	Automake
任意依存パッケージ:	Emacs

Automake

インストール依存パッケージ:	Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed, Texinfo
テストスイート依存パッケージ:	Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool, Tar.
事前インストールパッケージ:	なし
任意依存パッケージ:	なし

Bash

インストール依存パッケージ:	Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed, Texinfo
テストスイート依存パッケージ:	なし
事前インストールパッケージ:	なし
任意依存パッケージ:	Xorg

Binutils

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, File, Gawk, GCC, Glibc, Grep, Make, Perl, Sed, Texinfo, Zlib
テストスイート依存パッケージ:	DejaGNU, Expect
事前インストールパッケージ:	なし
任意依存パッケージ:	なし

Bison

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Sed
テストスイート依存パッケージ: Diffutils, Findutils
事前インストールパッケージ: Flex, Kbd, Tar
任意依存パッケージ: Doxygen（テストスイート用）

Bzip2

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, Patch
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

Coreutils

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, Patch, Perl, Sed, Texinfo
テストスイート依存パッケージ: Diffutils, E2fsprogs, Findutils, Util-linux-ng
事前インストールパッケージ: Bash, Diffutils, Findutils, Man-DB, Udev
任意依存パッケージ: Perl Expect と IO:Tty モジュール（テストスイート用）

DejaGNU

インストール依存パッケージ: Bash, Coreutils, Diffutils, GCC, Grep, Make, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Diffutils

インストール依存パッケージ: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Diffutils, Perl
事前インストールパッケージ: なし
任意依存パッケージ: なし

Expect

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed, Tcl
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

E2fsprogs

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Pkg-config, Sed, Texinfo, Util-linux-ng

テストスイート依存パッケージ: なし

事前インストールパッケージ: なし

任意依存パッケージ: なし

File

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Zlib

テストスイート依存パッケージ: テストスイートはありません

事前インストールパッケージ: なし

任意依存パッケージ: なし

Findutils

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo

テストスイート依存パッケージ: DejaGNU, Diffutils, Expect

事前インストールパッケージ: なし

任意依存パッケージ: なし

Flex

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed, Texinfo

テストスイート依存パッケージ: Bison, Gawk

事前インストールパッケージ: IPRoute2, Kbd, Man-DB

任意依存パッケージ: なし

Gawk

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, Texinfo

テストスイート依存パッケージ: Diffutils

事前インストールパッケージ: なし

任意依存パッケージ: なし

Gcc

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, GMP, Grep, M4, Make, MPC, MPFR, Patch, Perl, Sed, Tar, Texinfo

テストスイート依存パッケージ: DejaGNU, Expect

事前インストールパッケージ: なし

任意依存パッケージ: CLooG-PPL, GNAT, PPL

GDBM

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, Sed
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

Gettext

インストール依存パッケージ: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Diffutils, Perl, Tcl
事前インストールパッケージ: Automake
任意依存パッケージ: なし

Glibc

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API ヘッダ, Make, Perl, Sed, Texinfo
テストスイート依存パッケージ: File
事前インストールパッケージ: なし
任意依存パッケージ: なし

GMP

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, M4, Make, Sed, Texinfo
テストスイート依存パッケージ: なし
事前インストールパッケージ: MPFR, GCC
任意依存パッケージ: なし

Grep

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, Texinfo
テストスイート依存パッケージ: Gawk
事前インストールパッケージ: Man-DB
任意依存パッケージ: Pcre, Xorg, CUPS

Groff

インストール依存パッケージ: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed, Texinfo
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: Man-DB, Perl
任意依存パッケージ: GPL Ghostscript

GRUB

インストール依存パッケージ: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Texinfo
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

Gzip

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Diffutils
事前インストールパッケージ: Man-DB
任意依存パッケージ: なし

Iana-Etc

インストール依存パッケージ: Coreutils, Gawk, Make
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: Perl
任意依存パッケージ: なし

Inetutils

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo, Zlib
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: Tar
任意依存パッケージ: なし

IProute2

インストール依存パッケージ: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, Linux API ヘッダ
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Kbd

インストール依存パッケージ: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Less

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: Pcre

Libtool

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Findutils
事前インストールパッケージ: なし
任意依存パッケージ: なし

Linux Kernel

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Module-Init-Tools, Ncurses, Perl, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

M4

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Diffutils
事前インストールパッケージ: Autoconf, Bison
任意依存パッケージ: libsigsegv

Make

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Perl, Procps
事前インストールパッケージ: なし
任意依存パッケージ: なし

Man-DB

インストール依存パッケージ: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, GDBM, Gettext, Glibc, Grep, Groff, Gzip, Less, Make, Sed
テストスイート依存パッケージ: 動かすためには Man-DB テストスイートパッケージが必要
事前インストールパッケージ: なし
任意依存パッケージ: なし

Man-Pages

インストール依存パッケージ: Bash, Coreutils, Make
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Module-Init-Tools

インストール依存パッケージ: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Make, Patch, Sed, Zlib
テストスイート依存パッケージ: Diffutils, File, Gawk, Gzip
事前インストールパッケージ: なし
任意依存パッケージ: なし

MPC

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, MPFR, Sed, Texinfo
テストスイート依存パッケージ: なし
事前インストールパッケージ: GCC
任意依存パッケージ: なし

MPFR

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, Sed, Texinfo
テストスイート依存パッケージ: なし
事前インストールパッケージ: GCC
任意依存パッケージ: なし

Ncurses

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: Bash, GRUB, Inetutils, Less, Procps, Psmisc, Readline, Texinfo, Util-linux-ng, Vim
任意依存パッケージ: なし

Patch

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: Ed

Perl

インストール依存パッケージ: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Groff, Make, Sed, Zlib
テストスイート依存パッケージ: Iana-Etc, Procps
事前インストールパッケージ: Autoconf
任意依存パッケージ: なし

Pkg-config

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

Procps

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Psmisc

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Readline

インストール依存パッケージ: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: Bash
任意依存パッケージ: なし

Sed

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo
テストスイート依存パッケージ: Diffutils, Gawk
事前インストールパッケージ: E2fsprogs, File, Libtool, Shadow
任意依存パッケージ: Cracklib

Shadow

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Sysklogd

インストール依存パッケージ: Binutils, Coreutils, GCC, Glibc, Make, Patch
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Sysvinit

インストール依存パッケージ: Binutils, Coreutils, GCC, Glibc, Make, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Tar

インストール依存パッケージ: Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, Texinfo
テストスイート依存パッケージ: Diffutils, Findutils, Gawk, Gzip
事前インストールパッケージ: なし
任意依存パッケージ: なし

Tcl

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

Texinfo

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, Sed
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: なし

Udev

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Util-linux-ng

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib
テストスイート依存パッケージ: テストスイートはありません
事前インストールパッケージ: なし
任意依存パッケージ: なし

Vim

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed
テストスイート依存パッケージ: なし
事前インストールパッケージ: なし
任意依存パッケージ: Xorg, GTK+2, LessTif, Python, Tcl, Ruby, GPM

Zlib

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Make, Sed
テストスイート依存パッケージ: なし
事前インストールパッケージ: File, Module-Init-Tools, Perl, Util-linux-ng
任意依存パッケージ: なし

付録 D. ブートスクリプトと sysconfig スクリプト version-20100627

本付録に示すスクリプトは、それらが収容されているディレクトリごとに列記します。 /etc/rc.d/init.d、 /etc/sysconfig、 /etc/sysconfig/network-devices、 /etc/sysconfig/network-devices/services の順です。 各ディレクトリにおいてのスクリプトは呼び出し順に説明します。

D.1. /etc/rc.d/init.d/rc

rc スクリプトは init によって呼び出される最初のスクリプトであり、ブート処理を初期化します。

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/rc
#
# Description : Main Run Level Control Script
#
# Authors     : Gerard Beekmans  - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

# This sets a few default terminal options.
stty sane

# These 3 signals will not cause our script to exit
trap "" INT QUIT TSTP

[ "${1}" != "" ] && runlevel=${1}

if [ "${runlevel}" = "" ]; then
    echo "Usage: ${0} <runlevel>" >&2
    exit 1
fi

previous=${PREVLEVEL}
[ "${previous}" = "" ] && previous=N

if [ ! -d ${rc_base}/rc${runlevel}.d ]; then
    boot_mesg "${rc_base}/rc${runlevel}.d does not exist." ${WARNING}
    boot_mesg_flush
    exit 1
fi

# Attempt to stop all service started by previous runlevel,
# and killed in this runlevel
if [ "${previous}" != "N" ]; then
    for i in $(ls -v ${rc_base}/rc${runlevel}.d/K* 2> /dev/null)
    do
        check_script_status

        suffix=${i#$rc_base/rc$runlevel.d/K[0-9][0-9]}
        prev_start=$rc_base/rc$previous.d/S[0-9][0-9]$suffix
```

```
        sysinit_start=$rc_base/rcsysinit.d/S[0-9][0-9]$suffix

        if [ "${runlevel}" != "0" ] && [ "${runlevel}" != "6" ]; then
            if [ ! -f ${prev_start} ] && [ ! -f ${sysinit_start} ]; then
                boot_mesg -n "WARNING:\n\n${i} can't be" ${WARNING}
                boot_mesg -n " executed because it was not"
                boot_mesg -n " not started in the previous"
                boot_mesg -n " runlevel (${previous})."
                boot_mesg "" ${NORMAL}
                boot_mesg_flush
                continue
            fi
        fi
        ${i} stop
        error_value=${?}

        if [ "${error_value}" != "0" ]; then
            print_error_msg
        fi
    done
fi

#Start all functions in this runlevel
for i in $( ls -v ${rc_base}/rc${runlevel}.d/S* 2> /dev/null)
do
    if [ "${previous}" != "N" ]; then
        suffix=${i#$rc_base/rc$runlevel.d/S[0-9][0-9]}
        stop=$rc_base/rc$runlevel.d/K[0-9][0-9]$suffix
        prev_start=$rc_base/rc$previous.d/S[0-9][0-9]$suffix

        [ -f ${prev_start} ] && [ ! -f ${stop} ] && continue
    fi

    check_script_status

    case ${runlevel} in
        0|6)
            ${i} stop
            ;;
        *)
            ${i} start
            ;;
    esac
    error_value=${?}

    if [ "${error_value}" != "0" ]; then
        print_error_msg
    fi
done

# End $rc_base/init.d/rc
```

D.2. /etc/rc.d/init.d/functions

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/functions
#
# Description : Run Level Control Functions
#
```

```
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       : With code based on Matthias Benkmann's simpleinit-msb
#        http://winterdrache.de/linux/newboot/index.html
#
########################################################################

## Environmental setup
# Setup default values for environment
umask 022
export PATH="/bin:/usr/bin:/sbin:/usr/sbin"

# Signal sent to running processes to refresh their configuration
RELOADSIG="HUP"

# Number of seconds between STOPSIG and FALLBACK when stopping processes
KILLDELAY="3"

## Screen Dimensions
# Find current screen size
if [ -z "${COLUMNS}" ]; then
    COLUMNS=$(stty size)
    COLUMNS=${COLUMNS##* }
fi

# When using remote connections, such as a serial port, stty size returns 0
if [ "${COLUMNS}" = "0" ]; then
    COLUMNS=80
fi

## Measurements for positioning result messages
COL=$((${COLUMNS} - 8))
WCOL=$((${COL} - 2))

## Provide an echo that supports -e and -n
# If formatting is needed, $ECHO should be used
case "`echo -e -n test`" in
    -[en]*)
        ECHO=/bin/echo
        ;;
    *)
        ECHO=echo
        ;;
esac

## Set Cursor Position Commands, used via $ECHO
SET_COL="\\033[${COL}G"      # at the $COL char
SET_WCOL="\\033[${WCOL}G"    # at the $WCOL char
CURS_UP="\\033[1A\\033[0G"   # Up one line, at the 0'th char

## Set color commands, used via $ECHO
# Please consult 'man console_codes for more information
# under the "ECMA-48 Set Graphics Rendition" section
#
# Warning: when switching from a 8bit to a 9bit font,
# the linux console will reinterpret the bold (1;) to
# the top 256 glyphs of the 9bit font.  This does
# not affect framebuffer consoles
NORMAL="\\033[0;39m"         # Standard console grey
```

```
SUCCESS="\\033[1;32m"        # Success is green
WARNING="\\033[1;33m"        # Warnings are yellow
FAILURE="\\033[1;31m"        # Failures are red
INFO="\\033[1;36m"           # Information is light cyan
BRACKET="\\033[1;34m"        # Brackets are blue

STRING_LENGTH="0"   # the length of the current message

#*****************************************************************************
# Function - boot_mesg()
#
# Purpose:      Sending information from bootup scripts to the console
#
# Inputs:       $1 is the message
#               $2 is the colorcode for the console
#
# Outputs:      Standard Output
#
# Dependencies: - sed for parsing strings.
#            - grep for counting string length.
#
# Todo:
#*****************************************************************************
boot_mesg()
{
    local ECHOPARM=""

    while true
    do
        case "${1}" in
            -n)
                ECHOPARM=" -n "
                shift 1
                ;;
            -*)
                echo "Unknown Option: ${1}"
                return 1
                ;;
            *)
                break
                ;;
        esac
    done

    ## Figure out the length of what is to be printed to be used
    ## for warning messages.
    STRING_LENGTH=$((${#1} + 1))

    # Print the message to the screen
    ${ECHO} ${ECHOPARM} -e "${2}${1}"

}

boot_mesg_flush()
{
    # Reset STRING_LENGTH for next message
    STRING_LENGTH="0"
}

boot_log()
{
```

```
    # Left in for backwards compatibility
    :
}

echo_ok()
{
    ${ECHO} -n -e "${CURS_UP}${SET_COL}${BRACKET}[${SUCCESS}  OK  ${BRACKET}]"
    ${ECHO} -e "${NORMAL}"
        boot_mesg_flush
}

echo_failure()
{
    ${ECHO} -n -e "${CURS_UP}${SET_COL}${BRACKET}[${FAILURE} FAIL ${BRACKET}]"
    ${ECHO} -e "${NORMAL}"
        boot_mesg_flush
}

echo_warning()
{
    ${ECHO} -n -e "${CURS_UP}${SET_COL}${BRACKET}[${WARNING} WARN ${BRACKET}]"
    ${ECHO} -e "${NORMAL}"
        boot_mesg_flush
}

print_error_msg()
{
    echo_failure
    # $i is inherited by the rc script
    boot_mesg -n "FAILURE:\n\nYou should not be reading this error message.\n\n" ${FAILURE}
    boot_mesg -n " It means that an unforeseen error took"
    boot_mesg -n " place in ${i}, which exited with a return value of"
    boot_mesg " ${error_value}.\n"
    boot_mesg_flush
    boot_mesg -n "If you're able to track this"
    boot_mesg -n " error down to a bug in one of the files provided by"
    boot_mesg -n " the LFS book, please be so kind to inform us at"
    boot_mesg " lfs-dev@linuxfromscratch.org.\n"
    boot_mesg_flush
    boot_mesg -n "Press Enter to continue..." ${INFO}
    boot_mesg "" ${NORMAL}
    read ENTER
}

check_script_status()
{
    # $i is inherited by the rc script
    if [ ! -f ${i} ]; then
        boot_mesg "${i} is not a valid symlink." ${WARNING}
        echo_warning
        continue
    fi

    if [ ! -x ${i} ]; then
        boot_mesg "${i} is not executable, skipping." ${WARNING}
        echo_warning
        continue
    fi
}

evaluate_retval()
```

```
{
    error_value="${?}"

    if [ ${error_value} = 0 ]; then
        echo_ok
    else
        echo_failure
    fi

    # This prevents the 'An Unexpected Error Has Occurred' from trivial
    # errors.
    return 0
}

print_status()
{
    if [ "${#}" = "0" ]; then
        echo "Usage: ${0} {success|warning|failure}"
        return 1
    fi

    case "${1}" in

        success)
            echo_ok
            ;;

        warning)
            # Leave this extra case in because old scripts
            # may call it this way.
            case "${2}" in
                running)
                    ${ECHO} -e -n "${CURS_UP}"
                    ${ECHO} -e -n "\\033[${STRING_LENGTH}G   "
                    boot_mesg "Already running." ${WARNING}
                    echo_warning
                    ;;
                not_running)
                    ${ECHO} -e -n "${CURS_UP}"
                    ${ECHO} -e -n "\\033[${STRING_LENGTH}G   "
                    boot_mesg "Not running." ${WARNING}
                    echo_warning
                    ;;
                not_available)
                    ${ECHO} -e -n "${CURS_UP}"
                    ${ECHO} -e -n "\\033[${STRING_LENGTH}G   "
                    boot_mesg "Not available." ${WARNING}
                    echo_warning
                    ;;
                *)
                    # This is how it is supposed to
                    # be called
                    echo_warning
                    ;;
            esac
        ;;

        failure)
            echo_failure
        ;;
```

```
    esac

}

reloadproc()
{
    local pidfile=""
    local failure=0

    while true
    do
        case "${1}" in
            -p)
                pidfile="${2}"
                shift 2
                ;;
            -*)
                log_failure_msg "Unknown Option: ${1}"
                return 2
                ;;
            *)
                break
                ;;
        esac
    done

    if [ "${#}" -lt "1" ]; then
        log_failure_msg "Usage: reloadproc [-p pidfile] pathname"
        return 2
    fi

    # This will ensure compatibility with previous LFS Bootscripts
    if [ -n "${PIDFILE}" ];    then
        pidfile="${PIDFILE}"
    fi

    # Is the process running?
    if [ -z "${pidfile}" ];    then
        pidofproc -s "${1}"
    else
        pidofproc -s -p "${pidfile}" "${1}"
    fi

    # Warn about stale pid file
    if [ "$?" = 1 ]; then
        boot_mesg -n "Removing stale pid file: ${pidfile}. " ${WARNING}
        rm -f "${pidfile}"
    fi

    if [ -n "${pidlist}" ];    then
        for pid in ${pidlist}
        do
            kill -"${RELOADSIG}" "${pid}" || failure="1"
        done

        (exit ${failure})
        evaluate_retval

    else
        boot_mesg "Process ${1} not running." ${WARNING}
        echo_warning
```

```
    fi
}

statusproc()
{
    local pidfile=""
    local base=""
    local ret=""

    while true
    do
        case "${1}" in
            -p)
                pidfile="${2}"
                shift 2
                ;;
            -*)
                log_failure_msg "Unknown Option: ${1}"
                return 2
                ;;
            *)
                break
                ;;
        esac
    done

    if [ "${#}" != "1" ]; then
        shift 1
        log_failure_msg "Usage: statusproc [-p pidfile] pathname"
        return 2
    fi

    # Get the process basename
    base="${1##*/}"

    # This will ensure compatibility with previous LFS Bootscripts
    if [ -n "${PIDFILE}" ];    then
        pidfile="${PIDFILE}"
    fi

    # Is the process running?
    if [ -z "${pidfile}" ];    then
        pidofproc -s "${1}"
    else
        pidofproc -s -p "${pidfile}" "${1}"
    fi

    # Store the return status
    ret=$?

    if [ -n "${pidlist}" ];    then
        ${ECHO} -e "${INFO}${base} is running with Process"\
            "ID(s) ${pidlist}.${NORMAL}"
    else
        if [ -n "${base}" -a -e "/var/run/${base}.pid" ]; then
            ${ECHO} -e "${WARNING}${1} is not running but"\
                "/var/run/${base}.pid exists.${NORMAL}"
        else
            if [ -n "${pidfile}" -a -e "${pidfile}" ]; then
                ${ECHO} -e "${WARNING}${1} is not running"\
                    "but ${pidfile} exists.${NORMAL}"
```

```
            else
                ${ECHO} -e "${INFO}${1} is not running.${NORMAL}"
            fi
        fi
    fi

    # Return the status from pidofproc
    return $ret
}

# The below functions are documented in the LSB-generic 2.1.0

#****************************************************************************
# Function - pidofproc [-s] [-p pidfile] pathname
#
# Purpose: This function returns one or more pid(s) for a particular daemon
#
# Inputs: -p pidfile, use the specified pidfile instead of pidof
#         pathname, path to the specified program
#
# Outputs: return 0 - Success, pid's in stdout
#          return 1 - Program is dead, pidfile exists
#          return 2 - Invalid or excessive number of arguments,
#                     warning in stdout
#          return 3 - Program is not running
#
# Dependencies: pidof, echo, head
#
# Todo: Remove dependency on head
#       This replaces getpids
#       Test changes to pidof
#
#****************************************************************************
pidofproc()
{
    local pidfile=""
    local lpids=""
    local silent=""
    pidlist=""
    while true
    do
        case "${1}" in
            -p)
                pidfile="${2}"
                shift 2
                ;;

            -s)
                # Added for legacy opperation of getpids
                # eliminates several '> /dev/null'
                silent="1"
                shift 1
                ;;
            -*)
                log_failure_msg "Unknown Option: ${1}"
                return 2
                ;;
            *)
                break
                ;;
        esac
```

```
    done

    if [ "${#}" != "1" ]; then
        shift 1
        log_failure_msg "Usage: pidofproc [-s] [-p pidfile] pathname"
        return 2
    fi

    if [ -n "${pidfile}" ]; then
        if [ ! -r "${pidfile}" ]; then
            return 3 # Program is not running
        fi

        lpids='head -n 1 ${pidfile}'
        for pid in ${lpids}
        do
            if [ "${pid}" -ne "$$" -a "${pid}" -ne "${PPID}" ]; then
                kill -0 "${pid}" 2>/dev/null &&
                pidlist="${pidlist} ${pid}"
            fi

            if [ "${silent}" != "1" ]; then
                echo "${pidlist}"
            fi

            test -z "${pidlist}" &&
            # Program is dead, pidfile exists
            return 1
            # else
            return 0
        done

    else
        pidlist='pidof -o $$ -o $PPID -x "$1"'
        if [ "${silent}" != "1" ]; then
            echo "${pidlist}"
        fi

        # Get provide correct running status
        if [ -n "${pidlist}" ]; then
            return 0
        else
            return 3
        fi

    fi

    if [ "$?" != "0" ]; then
        return 3 # Program is not running
    fi
}

#****************************************************************************
# Function - loadproc [-f] [-n nicelevel] [-p pidfile] pathname [args]
#
# Purpose: This runs the specified program as a daemon
#
# Inputs: -f, run the program even if it is already running
#         -n nicelevel, specifies a nice level. See nice(1).
#         -p pidfile, uses the specified pidfile
#         pathname, pathname to the specified program
```

```
#         args, arguments to pass to specified program
#
# Outputs: return 0 - Success
#          return 2 - Invalid of excessive number of arguments,
#                     warning in stdout
#          return 4 - Program or service status is unknown
#
# Dependencies: nice, rm
#
# Todo: LSB says this should be called start_daemon
#       LSB does not say that it should call evaluate_retval
#       It checks for PIDFILE, which is deprecated.
#         Will be removed after BLFS 6.0
#       loadproc returns 0 if program is already running, not LSB compliant
#
#*******************************************************************************
loadproc()
{
    local pidfile=""
    local forcestart=""
    local nicelevel="10"

# This will ensure compatibility with previous LFS Bootscripts
    if [ -n "${PIDFILE}" ];    then
        pidfile="${PIDFILE}"
    fi

  while true
    do
        case "${1}" in
            -f)
                forcestart="1"
                shift 1
                ;;
            -n)
                nicelevel="${2}"
                shift 2
                ;;
            -p)
                pidfile="${2}"
                shift 2
                ;;
            -*)
                log_failure_msg "Unknown Option: ${1}"
                return 2 #invalid or excess argument(s)
                ;;
            *)
                break
                ;;
        esac
    done

    if [ "${#}" = "0" ]; then
        log_failure_msg "Usage: loadproc [-f] [-n nicelevel] [-p pidfile] pathname [args]"
        return 2 #invalid or excess argument(s)
    fi

    if [ -z "${forcestart}" ]; then
        if [ -z "${pidfile}" ];    then
            pidofproc -s "${1}"
        else
```

```
            pidofproc -s -p "${pidfile}" "${1}"
        fi

        case "${?}" in
            0)
                log_warning_msg "Unable to continue: ${1} is running"
                return 0 # 4
                ;;
            1)
                boot_mesg "Removing stale pid file: ${pidfile}" ${WARNING}
                rm -f "${pidfile}"
                ;;
            3)
                ;;
            *)
                log_failure_msg "Unknown error code from pidofproc: ${?}"
                return 4
                ;;
        esac
    fi

    nice -n "${nicelevel}" "${@}"
    evaluate_retval # This is "Probably" not LSB compliant,
#                       but required to be compatible with older bootscripts
    return 0
}

#****************************************************************************
# Function - killproc  [-p pidfile] pathname [signal]
#
# Purpose:
#
# Inputs: -p pidfile, uses the specified pidfile
#         pathname, pathname to the specified program
#         signal, send this signal to pathname
#
# Outputs: return 0 - Success
#          return 2 - Invalid of excessive number of arguments,
#                     warning in stdout
#          return 4 - Unknown Status
#
# Dependencies: kill, rm
#
# Todo: LSB does not say that it should call evaluate_retval
#       It checks for PIDFILE, which is deprecated.
#         Will be removed after BLFS 6.0
#
#****************************************************************************
killproc()
{
    local pidfile=""
    local killsig=TERM # default signal is SIGTERM
    pidlist=""

    # This will ensure compatibility with previous LFS Bootscripts
    if [ -n "${PIDFILE}" ];    then
        pidfile="${PIDFILE}"
    fi

    while true
    do
```

```
        case "${1}" in
            -p)
                pidfile="${2}"
                shift 2
                ;;
            -*)
                log_failure_msg "Unknown Option: ${1}"
                return 2
                ;;
            *)
                 break
                ;;
        esac
    done

    if [ "${#}" = "2" ]; then
        killsig="${2}"
    elif [ "${#}" != "1" ];    then
        shift 2
        log_failure_msg "Usage: killproc  [-p pidfile] pathname [signal]"
        return 2
    fi

    # Is the process running?
    if [ -z "${pidfile}" ];    then
        pidofproc -s "${1}"
    else
        pidofproc -s -p "${pidfile}" "${1}"
    fi

    # Remove stale pidfile
    if [ "$?" = 1 ]; then
        boot_mesg "Removing stale pid file: ${pidfile}." ${WARNING}
        rm -f "${pidfile}"
    fi

    # If running, send the signal
    if [ -n "${pidlist}" ]; then
    for pid in ${pidlist}
    do
        kill -${killsig} ${pid} 2>/dev/null

        # Wait up to 3 seconds, for ${pid} to terminate
        case "${killsig}" in
        TERM|SIGTERM|KILL|SIGKILL)
            # sleep in 1/10ths of seconds and
            # multiply KILLDELAY by 10
            local dtime="${KILLDELAY}0"
            while [ "${dtime}" != "0" ]
            do
                kill -0 ${pid} 2>/dev/null || break
                sleep 0.1
                dtime=$(( ${dtime} - 1))
            done
            # If ${pid} is still running, kill it
            kill -0 ${pid} 2>/dev/null && kill -KILL ${pid} 2>/dev/null
            ;;
        esac
    done

    # Check if the process is still running if we tried to stop it
```

```
    case "${killsig}" in
    TERM|SIGTERM|KILL|SIGKILL)
        if [ -z "${pidfile}" ];    then
            pidofproc -s "${1}"
        else
            pidofproc -s -p "${pidfile}" "${1}"
        fi

        # Program was terminated
        if [ "$?" != "0" ]; then
            # Remove the pidfile if necessary
            if [ -f "${pidfile}" ];    then
                rm -f "${pidfile}"
            fi
            echo_ok
            return 0
        else # Program is still running
            echo_failure
            return 4 # Unknown Status
        fi
        ;;
    *)
        # Just see if the kill returned successfully
        evaluate_retval
        ;;
    esac
    else # process not running
    print_status warning not_running
    fi
}


#****************************************************************************
# Function - log_success_msg "message"
#
# Purpose: Print a success message
#
# Inputs: $@ - Message
#
# Outputs: Text output to screen
#
# Dependencies: echo
#
# Todo: logging
#
#****************************************************************************
log_success_msg()
{
    ${ECHO} -n -e "${BOOTMESG_PREFIX}${@}"
    ${ECHO} -e "${SET_COL}""${BRACKET}""["${SUCCESS}""  OK  ""${BRACKET}""]""${NORMAL}"
    return 0
}

#****************************************************************************
# Function - log_failure_msg "message"
#
# Purpose: Print a failure message
#
# Inputs: $@ - Message
#
# Outputs: Text output to screen
```

```
#
# Dependencies: echo
#
# Todo: logging
#
#****************************************************************************
log_failure_msg() {
    ${ECHO} -n -e "${BOOTMESG_PREFIX}${@}"
    ${ECHO} -e "${SET_COL}""${BRACKET}""["${FAILURE}"" FAIL ""${BRACKET}""]""${NORMAL}"
    return 0
}

#****************************************************************************
# Function - log_warning_msg "message"
#
# Purpose: print a warning message
#
# Inputs: $@ - Message
#
# Outputs: Text output to screen
#
# Dependencies: echo
#
# Todo: logging
#
#****************************************************************************
log_warning_msg() {
    ${ECHO} -n -e "${BOOTMESG_PREFIX}${@}"
    ${ECHO} -e "${SET_COL}""${BRACKET}""["${WARNING}"" WARN ""${BRACKET}""]""${NORMAL}"
    return 0
}

# End $rc_base/init.d/functions
```

D.3. /etc/rc.d/init.d/mountkernfs

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/mountkernfs
#
# Description : Mount proc and sysfs
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        boot_mesg -n "Mounting kernel-based file systems:" ${INFO}

        if ! mountpoint /proc >/dev/null; then
            boot_mesg -n " /proc" ${NORMAL}
            mount -n /proc || failed=1
```

```
        fi

        if ! mountpoint /sys >/dev/null; then
            boot_mesg -n " /sys" ${NORMAL}
            mount -n /sys || failed=1
        fi

        boot_mesg "" ${NORMAL}

        (exit ${failed})
        evaluate_retval
        ;;

    *)
        echo "Usage: ${0} {start}"
        exit 1
        ;;
esac

# End $rc_base/init.d/mountkernfs
```

D.4. /etc/rc.d/init.d/consolelog

```
#!/bin/sh
# Begin $rc_base/init.d/consolelog

########################################################################
#
# Description : Set the kernel log level for the console
#
# Authors     : Dan Nicholson - dnicholson@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       : /proc must be mounted before this can run
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

# set the default loglevel
LOGLEVEL=7
if [ -r /etc/sysconfig/console ]; then
    . /etc/sysconfig/console
fi

case "${1}" in
    start)
        case "$LOGLEVEL" in
        [1-8])
            boot_mesg "Setting the console log level to ${LOGLEVEL}..."
            dmesg -n $LOGLEVEL
            evaluate_retval
            ;;
        *)
            boot_mesg "Console log level '${LOGLEVEL}' is invalid" ${FAILURE}
            echo_failure
            ;;
        esac
```

```
        ;;
    status)
        # Read the current value if possible
        if [ -r /proc/sys/kernel/printk ]; then
            read level line < /proc/sys/kernel/printk
        else
            boot_mesg "Can't read the current console log level" ${FAILURE}
            echo_failure
        fi

        # Print the value
        if [ -n "$level" ]; then
            ${ECHO} -e "${INFO}The current console log level" \
                "is ${level}${NORMAL}"
        fi
        ;;

    *)
        echo "Usage: ${0} {start|status}"
        exit 1
        ;;
esac

# End $rc_base/init.d/consolelog
```

D.5. /etc/rc.d/init.d/modules

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/modules
#
# Description : Module auto-loading script
#
# Authors     : Zack Winkles
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

# Assure that the kernel has module support.
[ -e /proc/ksyms -o -e /proc/modules ] || exit 0

case "${1}" in
    start)

        # Exit if there's no modules file or there are no
        # valid entries
        [ -r /etc/sysconfig/modules ] &&
            egrep -qv '^($|#)' /etc/sysconfig/modules ||
            exit 0

        boot_mesg -n "Loading modules:" ${INFO}

        # Only try to load modules if the user has actually given us
        # some modules to load.
```

```
        while read module args; do

            # Ignore comments and blank lines.
            case "$module" in
                ""|"#"*) continue ;;
            esac

            # Attempt to load the module, making
            # sure to pass any arguments provided.
            modprobe ${module} ${args} >/dev/null

            # Print the module name if successful,
            # otherwise take note.
            if [ $? -eq 0 ]; then
                boot_mesg -n " ${module}" ${NORMAL}
            else
                failedmod="${failedmod} ${module}"
            fi
        done < /etc/sysconfig/modules

        boot_mesg "" ${NORMAL}
        # Print a message about successfully loaded
        # modules on the correct line.
        echo_ok

        # Print a failure message with a list of any
        # modules that may have failed to load.
        if [ -n "${failedmod}" ]; then
            boot_mesg "Failed to load modules:${failedmod}" ${FAILURE}
            echo_failure
        fi
        ;;
    *)
        echo "Usage: ${0} {start}"
        exit 1
        ;;
esac

# End $rc_base/init.d/modules
```

D.6. /etc/rc.d/init.d/udev

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/udev
#
# Description : Udev cold-plugging script
#
# Authors     : Zack Winkles, Alexander E. Patrakov
#
# Version     : 00.02
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
```

```
    start)
        boot_mesg "Populating /dev with device nodes..."
        if ! grep -q '[[:space:]]sysfs' /proc/mounts; then
            echo_failure
            boot_mesg -n "FAILURE:\n\nUnable to create" ${FAILURE}
            boot_mesg -n " devices without a SysFS filesystem"
            boot_mesg -n "\n\nAfter you press Enter, this system"
            boot_mesg -n " will be halted and powered off."
            boot_mesg -n "\n\nPress Enter to continue..." ${INFO}
            boot_mesg "" ${NORMAL}
            read ENTER
            /etc/rc.d/init.d/halt stop
        fi

        # Mount a temporary file system over /dev, so that any devices
        # made or removed during this boot don't affect the next one.
        # The reason we don't write to mtab is because we don't ever
        # want /dev to be unavailable (such as by 'umount -a').
        if ! mountpoint /dev > /dev/null; then
            mount -n -t tmpfs tmpfs /dev -o mode=755
        fi
        if [ ${?} != 0 ]; then
            echo_failure
            boot_mesg -n "FAILURE:\n\nCannot mount a tmpfs" ${FAILURE}
            boot_mesg -n " onto /dev, this system will be halted."
            boot_mesg -n "\n\nAfter you press Enter, this system"
            boot_mesg -n " will be halted and powered off."
            boot_mesg -n "\n\nPress Enter to continue..." ${INFO}
            boot_mesg "" ${NORMAL}
            read ENTER
            /etc/rc.d/init.d/halt stop
        fi

        # Udev handles uevents itself, so we don't need to have
        # the kernel call out to any binary in response to them
        echo > /proc/sys/kernel/hotplug

        # Copy the only static device node that Udev >= 155 doesn't
        # handle to /dev
        cp -a /lib/udev/devices/null /dev

        # Start the udev daemon to continually watch for, and act on,
        # uevents
        /sbin/udevd --daemon

        # Now traverse /sys in order to "coldplug" devices that have
        # already been discovered
        /sbin/udevadm trigger --action=add

        # Now wait for udevd to process the uevents we triggered
        /sbin/udevadm settle
        evaluate_retval

        ;;

    *)
        echo "Usage ${0} {start}"
        exit 1
        ;;
esac
```

```
# End $rc_base/init.d/udev
```

D.7. /etc/rc.d/init.d/swap

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/swap
#
# Description : Swap Control Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        boot_mesg "Activating all swap files/partitions..."
        swapon -a
        evaluate_retval
        ;;

    stop)
        boot_mesg "Deactivating all swap files/partitions..."
        swapoff -a
        evaluate_retval
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;

    status)
        boot_mesg "Retrieving swap status." ${INFO}
        echo_ok
        echo
        swapon -s
        ;;

    *)
        echo "Usage: ${0} {start|stop|restart|status}"
        exit 1
        ;;
esac

# End $rc_base/init.d/swap
```

D.8. /etc/rc.d/init.d/setclock

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/setclock
```

```
#
# Description : Setting Linux Clock
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
###############################################################################

. /etc/sysconfig/rc
. ${rc_functions}
. /etc/sysconfig/clock

case "${UTC}" in
    yes|true|1)
        CLOCKPARAMS="${CLOCKPARAMS} --utc"
        ;;

    no|false|0)
        CLOCKPARAMS="${CLOCKPARAMS} --localtime"
        ;;

esac

case ${1} in
    start)
        boot_mesg "Setting system clock..."
        hwclock --hctosys ${CLOCKPARAMS} >/dev/null
        evaluate_retval
        ;;

    stop)
        boot_mesg "Setting hardware clock..."
        hwclock --systohc ${CLOCKPARAMS} >/dev/null
        evaluate_retval
        ;;

    *)
        echo "Usage: ${0} {start|stop}"
        ;;

esac
```

D.9. /etc/rc.d/init.d/checkfs

```
#!/bin/sh
###############################################################################
# Begin $rc_base/init.d/checkfs
#
# Description : File System Check
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               A. Luebke - luebke@users.sourceforge.net
#
# Version     : 00.00
#
# Notes       :
#
```

```
# Based on checkfs script from LFS-3.1 and earlier.
#
# From man fsck
# 0    - No errors
# 1    - File system errors corrected
# 2    - System should be rebooted
# 4    - File system errors left uncorrected
# 8    - Operational error
# 16   - Usage or syntax error
# 32   - Fsck canceled by user request
# 128  - Shared library error
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        if [ -f /fastboot ]; then
            boot_mesg -n "/fastboot found, will not perform" ${INFO}
            boot_mesg " file system checks as requested."
            echo_ok
            exit 0
        fi

        boot_mesg "Mounting root file system in read-only mode..."
        mount -n -o remount,ro / >/dev/null
        evaluate_retval

        if [ ${?} != 0 ]; then
            echo_failure
            boot_mesg -n "FAILURE:\n\nCannot check root" ${FAILURE}
            boot_mesg -n " filesystem because it could not be mounted"
            boot_mesg -n " in read-only mode.\n\nAfter you"
            boot_mesg -n " press Enter, this system will be"
            boot_mesg -n " halted and powered off."
            boot_mesg -n "\n\nPress enter to continue..." ${INFO}
            boot_mesg "" ${NORMAL}
            read ENTER
            ${rc_base}/init.d/halt stop
        fi

        if [ -f /forcefsck ]; then
            boot_mesg -n "/forcefsck found, forcing file" ${INFO}
            boot_mesg " system checks as requested."
            echo_ok
            options="-f"
        else
            options=""
        fi

        boot_mesg "Checking file systems..."
        # Note: -a option used to be -p; but this fails e.g.
        # on fsck.minix
        fsck ${options} -a -A -C -T
        error_value=${?}

        if [ "${error_value}" = 0 ]; then
            echo_ok
        fi
```

```
        if [ "${error_value}" = 1 ]; then
            echo_warning
            boot_mesg -n "WARNING:\n\nFile system errors" ${WARNING}
            boot_mesg -n " were found and have been corrected."
            boot_mesg -n "  You may want to double-check that"
            boot_mesg -n " everything was fixed properly."
            boot_mesg "" ${NORMAL}
        fi

        if [ "${error_value}" = 2 -o "${error_value}" = 3 ]; then
            echo_warning
            boot_mesg -n "WARNING:\n\nFile system errors" ${WARNING}
            boot_mesg -n " were found and have been been"
            boot_mesg -n " corrected, but the nature of the"
            boot_mesg -n " errors require this system to be"
            boot_mesg -n " rebooted.\n\nAfter you press enter,"
            boot_mesg -n " this system will be rebooted"
            boot_mesg -n "\n\nPress Enter to continue..." ${INFO}
            boot_mesg "" ${NORMAL}
            read ENTER
            reboot -f
        fi

        if [ "${error_value}" -gt 3 -a "${error_value}" -lt 16 ]; then
            echo_failure
            boot_mesg -n "FAILURE:\n\nFile system errors" ${FAILURE}
            boot_mesg -n " were encountered that could not be"
            boot_mesg -n " fixed automatically.  This system"
            boot_mesg -n " cannot continue to boot and will"
            boot_mesg -n " therefore be halted until those"
            boot_mesg -n " errors are fixed manually by a"
            boot_mesg -n " System Administrator.\n\nAfter you"
            boot_mesg -n " press Enter, this system will be"
            boot_mesg -n " halted and powered off."
            boot_mesg -n "\n\nPress Enter to continue..." ${INFO}
            boot_mesg "" ${NORMAL}
            read ENTER
          ${rc_base}/init.d/halt stop
        fi

        if [ "${error_value}" -ge 16 ]; then
            echo_failure
            boot_mesg -n "FAILURE:\n\nUnexpected Failure" ${FAILURE}
            boot_mesg -n " running fsck.  Exited with error"
            boot_mesg -n " code: ${error_value}."
            boot_mesg "" ${NORMAL}
            exit ${error_value}
        fi
        ;;
    *)
        echo "Usage: ${0} {start}"
        exit 1
        ;;
esac

# End $rc_base/init.d/checkfs
```

D.10. /etc/rc.d/init.d/mountfs

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/mountfs
#
# Description : File System Mount Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        boot_mesg "Remounting root file system in read-write mode..."
        mount -n -o remount,rw / >/dev/null
        evaluate_retval

        # Remove fsck-related file system watermarks.
        rm -f /fastboot /forcefsck

        boot_mesg "Recording existing mounts in /etc/mtab..."
        > /etc/mtab
        mount -f / || failed=1
        mount -f /proc || failed=1
        mount -f /sys || failed=1
        (exit ${failed})
        evaluate_retval

        # This will mount all filesystems that do not have _netdev in
        # their option list.  _netdev denotes a network filesystem.
        boot_mesg "Mounting remaining file systems..."
        mount -a -O no_netdev >/dev/null
        evaluate_retval
        ;;

    stop)
        boot_mesg "Unmounting all other currently mounted file systems..."
        umount -a -d -r >/dev/null
        evaluate_retval
        ;;

    *)
        echo "Usage: ${0} {start|stop}"
        exit 1
        ;;
esac

# End $rc_base/init.d/mountfs
```

D.11. /etc/rc.d/init.d/udev_retry

```
#!/bin/sh
```

```
########################################################################
# Begin $rc_base/init.d/udev_retry
#
# Description : Udev cold-plugging script (retry)
#
# Authors     : Alexander E. Patrakov
#
# Version     : 00.02
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        boot_mesg "Retrying failed uevents, if any..."

        # From Debian: "copy the rules generated before / was mounted
        # read-write":
        for file in /dev/.udev/tmp-rules--*; do
            dest=${file##*tmp-rules--}
            [ "$dest" = '*' ] && break
            cat $file >> /etc/udev/rules.d/$dest
            rm -f $file
        done

        # Re-trigger the failed uevents in hope they will succeed now
        /sbin/udevadm trigger --type=failed --action=add

        # Now wait for udevd to process the uevents we triggered
        /sbin/udevadm settle
        evaluate_retval
        ;;

    *)
        echo "Usage ${0} {start}"
        exit 1
        ;;
esac

# End $rc_base/init.d/udev_retry
```

D.12. /etc/rc.d/init.d/cleanfs

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/cleanfs
#
# Description : Clean file system
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################
```

```
. /etc/sysconfig/rc
. ${rc_functions}

# Function to create files/directory on boot.
create_files() {
    # Read in the configuration file.
    exec 9>&0 < /etc/sysconfig/createfiles
        while read name type perm usr grp dtype maj min junk
        do

            # Ignore comments and blank lines.
                case "${name}" in
                ""|\#*) continue ;;
            esac

            # Ignore existing files.
            if [ ! -e "${name}" ]; then
                # Create stuff based on its type.
                case "${type}" in
                    dir)
                        mkdir "${name}"
                        ;;
                    file)
                        :> "${name}"
                        ;;
                    dev)
                        case "${dtype}" in
                            char)
                                mknod "${name}" c ${maj} ${min}
                                ;;
                            block)
                                mknod "${name}" b ${maj} ${min}
                                ;;
                            pipe)
                                mknod "${name}" p
                                ;;
                            *)
                                boot_mesg -n "\nUnknown device type: ${dtype}" ${WARNING}
                                boot_mesg "" ${NORMAL}
                                ;;
                        esac
                        ;;
                    *)
                        boot_mesg -n "\nUnknown type: ${type}" ${WARNING}
                        boot_mesg "" ${NORMAL}
                        continue
                        ;;
                esac

                # Set up the permissions, too.
                chown ${usr}:${grp} "${name}"
                chmod ${perm} "${name}"
            fi
        done
    exec 0>&9 9>&-
}

case "${1}" in
    start)
        boot_mesg -n "Cleaning file systems:" ${INFO}
```

```
        boot_mesg -n " /tmp" ${NORMAL}
        cd /tmp &&
        find . -xdev -mindepth 1 ! -name lost+found \
            -delete || failed=1

        boot_mesg -n " /var/lock" ${NORMAL}
        cd /var/lock &&
        find . -type f -exec rm -f {} \; || failed=1

        boot_mesg " /var/run" ${NORMAL}
        cd /var/run &&
        find . ! -type d ! -name utmp \
            -exec rm -f {} \; || failed=1
        > /var/run/utmp
        if grep -q '^utmp:' /etc/group ; then
            chmod 664 /var/run/utmp
            chgrp utmp /var/run/utmp
        fi

        (exit ${failed})
        evaluate_retval

        if egrep -qv '^(#|$)' /etc/sysconfig/createfiles 2>/dev/null; then
            boot_mesg "Creating files and directories..."
            create_files
            evaluate_retval
        fi
        ;;
    *)
        echo "Usage: ${0} {start}"
        exit 1
        ;;
esac

# End $rc_base/init.d/cleanfs
```

D.13. /etc/rc.d/init.d/console

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/console
#
# Description : Sets keymap and screen font
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               Alexander E. Patrakov
#
# Version     : 00.03
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

# Native English speakers probably don't have /etc/sysconfig/console at all
if [ -f /etc/sysconfig/console ]
then
```

```
    . /etc/sysconfig/console
else
        exit 0
fi

is_true() {
    [ "$1" = "1" ] || [ "$1" = "yes" ] || [ "$1" = "true" ]
}

failed=0

case "${1}" in
    start)
        boot_mesg "Setting up Linux console..."
        # There should be no bogus failures below this line!

        # Figure out if a framebuffer console is used
        [ -d /sys/class/graphics/fb0 ] && USE_FB=1 || USE_FB=0

        # Figure out the command to set the console into the
        # desired mode
        is_true "${UNICODE}" &&
            MODE_COMMAND="${ECHO} -en '\033%G' && kbd_mode -u" ||
            MODE_COMMAND="${ECHO} -en '\033%@\033(K' && kbd_mode -a"

        # On framebuffer consoles, font has to be set for each vt in
        # UTF-8 mode. This doesn't hurt in non-UTF-8 mode also.

        ! is_true "${USE_FB}" || [ -z "${FONT}" ] ||
            MODE_COMMAND="${MODE_COMMAND} && setfont ${FONT}"

        # Apply that command to all consoles mentioned in
        # /etc/inittab. Important: in the UTF-8 mode this should
        # happen before setfont, otherwise a kernel bug will
        # show up and the unicode map of the font will not be
        # used.
        # FIXME: Fedora Core also initializes two spare consoles
        # - do we want that?

        for TTY in 'grep '^[^#].*respawn:/sbin/agetty' /etc/inittab |
            grep -o '\btty[[:digit:]]*\b''
        do
            openvt -f -w -c ${TTY#tty} -- \
                /bin/sh -c "${MODE_COMMAND}" || failed=1
        done

        # Set the font (if not already set above) and the keymap
        is_true "${USE_FB}" || [ -z "${FONT}" ] ||
            setfont $FONT ||
            failed=1
        [ -z "${KEYMAP}" ] ||
            loadkeys ${KEYMAP} >/dev/null 2>&1 ||
            failed=1
        [ -z "${KEYMAP_CORRECTIONS}" ] ||
            loadkeys ${KEYMAP_CORRECTIONS} >/dev/null 2>&1 ||
            failed=1

        # Convert the keymap from $LEGACY_CHARSET to UTF-8
        [ -z "$LEGACY_CHARSET" ] ||
            dumpkeys -c "$LEGACY_CHARSET" |
            loadkeys -u >/dev/null 2>&1 ||
```

```
            failed=1

        # If any of the commands above failed, the trap at the
        # top would set $failed to 1
        ( exit $failed )
        evaluate_retval
        ;;
    *)
        echo $"Usage:" "${0} {start}"
        exit 1
        ;;
esac

# End $rc_base/init.d/console
```

D.14. /etc/rc.d/init.d/localnet

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/localnet
#
# Description : Loopback device
#
# Authors     : Gerard Beekmans  - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}
. /etc/sysconfig/network

case "${1}" in
    start)
        boot_mesg "Bringing up the loopback interface..."
        ip addr add 127.0.0.1/8 label lo dev lo
        ip link set lo up
        evaluate_retval

        boot_mesg "Setting hostname to ${HOSTNAME}..."
        hostname ${HOSTNAME}
        evaluate_retval
        ;;

    stop)
        boot_mesg "Bringing down the loopback interface..."
        ip link set lo down
        evaluate_retval
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;

    status)
```

```
        echo "Hostname is: $(hostname)"
        ip link show lo
        ;;

    *)
        echo "Usage: ${0} {start|stop|restart|status}"
        exit 1
        ;;
esac

# End $rc_base/init.d/localnet
```

D.15. /etc/rc.d/init.d/sysctl

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/sysctl
#
# Description : File uses /etc/sysctl.conf to set kernel runtime
#               parameters
#
# Authors     : Nathan Coulson (nathan@linuxfromscratch.org)
#               Matthew Burgress (matthew@linuxfromscratch.org)
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        if [ -f "/etc/sysctl.conf" ]; then
            boot_mesg "Setting kernel runtime parameters..."
            sysctl -q -p
            evaluate_retval
        fi
        ;;

    status)
        sysctl -a
        ;;

    *)
        echo "Usage: ${0} {start|status}"
        exit 1
        ;;
esac

# End $rc_base/init.d/sysctl
```

D.16. /etc/rc.d/init.d/sysklogd

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/sysklogd
#
```

```
# Description : Sysklogd loader
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
###############################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        boot_mesg "Starting system log daemon..."
        loadproc syslogd -m 0

        boot_mesg "Starting kernel log daemon..."
        loadproc klogd
        ;;

    stop)
        boot_mesg "Stopping kernel log daemon..."
        killproc klogd

        boot_mesg "Stopping system log daemon..."
        killproc syslogd
        ;;

    reload)
        boot_mesg "Reloading system log daemon config file..."
        reloadproc syslogd
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;

    status)
        statusproc syslogd
        statusproc klogd
        ;;

    *)
        echo "Usage: ${0} {start|stop|reload|restart|status}"
        exit 1
        ;;
esac

# End $rc_base/init.d/sysklogd
```

D.17. /etc/rc.d/init.d/network

```
#!/bin/sh
###############################################################################
# Begin $rc_base/init.d/network
#
```

```
# Description : Network Control Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}
. /etc/sysconfig/network

case "${1}" in
    start)
        # Start all network interfaces
        for file in ${network_devices}/ifconfig.*
        do
            interface=${file##*/ifconfig.}

            # skip if $file is * (because nothing was found)
            if [ "${interface}" = "*" ]
            then
                continue
            fi

            IN_BOOT=1 ${network_devices}/ifup ${interface}
        done
        ;;

    stop)
        # Reverse list
        FILES=""
        for file in ${network_devices}/ifconfig.*
        do
            FILES="${file} ${FILES}"
        done

        # Stop all network interfaces
        for file in ${FILES}
        do
            interface=${file##*/ifconfig.}

            # skip if $file is * (because nothing was found)
            if [ "${interface}" = "*" ]
            then
                continue
            fi

            IN_BOOT=1 ${network_devices}/ifdown ${interface}
        done
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;
```

```
    *)
        echo "Usage: ${0} {start|stop|restart}"
        exit 1
        ;;
esac

# End /etc/rc.d/init.d/network
```

D.18. /etc/rc.d/init.d/sendsignals

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/sendsignals
#
# Description : Sendsignals Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    stop)
        boot_mesg "Sending all processes the TERM signal..."
        killall5 -15
        error_value=${?}

        sleep ${KILLDELAY}

        if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
            echo_ok
        else
            echo_failure
        fi

        boot_mesg "Sending all processes the KILL signal..."
        killall5 -9
        error_value=${?}

        sleep ${KILLDELAY}

        if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
            echo_ok
        else
            echo_failure
        fi
        ;;

    *)
        echo "Usage: ${0} {stop}"
        exit 1
        ;;
```

```
esac

# End $rc_base/init.d/sendsignals
```

D.19. /etc/rc.d/init.d/reboot

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/reboot
#
# Description : Reboot Scripts
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    stop)
        boot_mesg "Restarting system..."
        reboot -d -f -i
        ;;

    *)
        echo "Usage: ${0} {stop}"
        exit 1
        ;;

esac

# End $rc_base/init.d/reboot
```

D.20. /etc/rc.d/init.d/halt

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/halt
#
# Description : Halt Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    stop)
        halt -d -f -i -p
```

```
        ;;
    *)
        echo "Usage: {stop}"
        exit 1
        ;;
esac

# End $rc_base/init.d/halt
```

D.21. /etc/rc.d/init.d/template

```
#!/bin/sh
########################################################################
# Begin $rc_base/init.d/
#
# Description :
#
# Authors     :
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

case "${1}" in
    start)
        boot_mesg "Starting..."
        loadproc
        ;;

    stop)
        boot_mesg "Stopping..."
        killproc
        ;;

    reload)
        boot_mesg "Reloading..."
        reloadproc
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;

    status)
        statusproc
        ;;

    *)
        echo "Usage: ${0} {start|stop|reload|restart|status}"
        exit 1
        ;;
esac
```

```
# End $rc_base/init.d/
```

D.22. /etc/sysconfig/rc

```
################################################################################
# Begin /etc/sysconfig/rc
#
# Description : rc script configuration
#
# Authors     :
#
# Version     : 00.00
#
# Notes       :
#
################################################################################

rc_base=/etc/rc.d
rc_functions=${rc_base}/init.d/functions
network_devices=/etc/sysconfig/network-devices

# End /etc/sysconfig/rc
```

D.23. /etc/sysconfig/modules

```
################################################################################
# Begin /etc/sysconfig/modules
#
# Description : Module auto-loading configuration
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : The syntax of this file is as follows:
#          <module> [<arg1> <arg2> ...]
#
# Each module should be on it's own line, and any options that you want
# passed to the module should follow it.  The line deliminator is either
# a space or a tab.
################################################################################

# End /etc/sysconfig/modules
```

D.24. /etc/sysconfig/createfiles

```
################################################################################
# Begin /etc/sysconfig/createfiles
#
# Description : Createfiles script config file
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : The syntax of this file is as follows:
#         if type is equal to "file" or "dir"
#          <filename> <type> <permissions> <user> <group>
#         if type is equal to "dev"
```

```
#          <filename> <type> <permissions> <user> <group> <devtype> <major> <minor>
#
#        <filename> is the name of the file which is to be created
#        <type> is either file, dir, or dev.
#              file creates a new file
#              dir creates a new directory
#              dev creates a new device
#        <devtype> is either block, char or pipe
#              block creates a block device
#              char creates a character deivce
#              pipe creates a pipe, this will ignore the <major> and <minor> fields
#        <major> and <minor> are the major and minor numbers used for the device.
#########################################################################

# End /etc/sysconfig/createfiles
```

D.25. /etc/sysconfig/network-devices/ifup

```
#!/bin/sh
#########################################################################
# Begin $network_devices/ifup
#
# Description : Interface Up
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       : the IFCONFIG variable is passed to the scripts found
#               in the services directory, to indicate what file the
#               service should source to get environmental variables.
#
#########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

# Collect a list of configuration files for our interface
if [ -n "${2}" ]; then
    for file in ${@#$1} # All parameters except $1
  do
        FILES="${FILES} ${network_devices}/ifconfig.${1}/${file}"
    done
elif [ -d "${network_devices}/ifconfig.${1}" ]; then
    FILES='echo ${network_devices}/ifconfig.${1}/*'
else
    FILES="${network_devices}/ifconfig.${1}"
fi

boot_mesg "Bringing up the ${1} interface..."
boot_mesg_flush

# Process each configruation file
for file in ${FILES}; do
    # skip backup files
    if [ "${file}" != "${file%""~""}" ]; then
        continue
    fi
```

```
    if [ ! -f "${file}" ]; then
        boot_mesg "${file} is not a network configuration file or directory." ${WARNING}
        echo_warning
        continue
    fi

    (
        . ${file}

        # Will not process this service if started by boot, and ONBOOT
        # is not set to yes
        if [ "${IN_BOOT}" = "1" -a "${ONBOOT}" != "yes" ]; then
            continue
        fi
        # Will not process this service if started by hotplug, and
        # ONHOTPLUG is not set to yes
        if [ "${IN_HOTPLUG}" = "1" -a "${ONHOTPLUG}" != "yes" \
                    -a "${HOSTNAME}" != "(none)" ]; then continue
        fi

        if [ -n "${SERVICE}" -a -x "${network_devices}/services/${SERVICE}" ]; then
            if [ -z "${CHECK_LINK}" -o "${CHECK_LINK}" = "y" \
                            -o "${CHECK_LINK}" = "yes" -o "${CHECK_LINK}" = "1" ]; then
                if ip link show ${1} > /dev/null 2>&1; then
                    link_status='ip link show ${1}'
                    if [ -n "${link_status}" ]; then
                        if ! echo "${link_status}" | grep -q UP; then
                            ip link set ${1} up
                        fi
                    fi
                else
                    boot_mesg "Interface ${1} doesn't exist." ${WARNING}
                    echo_warning
                    continue
                fi
            fi
            IFCONFIG=${file} ${network_devices}/services/${SERVICE} ${1} up
        else
            boot_mesg "Unable to process ${file}.  Either" ${FAILURE}
            boot_mesg " the SERVICE variable was not set,"
            boot_mesg " or the specified service cannot be executed."
            echo_failure
            continue
        fi
    )
done

# End $network_devices/ifup
```

D.26. /etc/sysconfig/network-devices/ifdown

```
#!/bin/sh
########################################################################
# Begin $network_devices/ifdown
#
# Description : Interface Down
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
#
```

```
# Version     : 00.01
#
# Notes       : the IFCONFIG variable is passed to the scripts found
#               in the services directory, to indicate what file the
#               service should source to get environmental variables.
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}

# Collect a list of configuration files for our interface
if [ -n "${2}" ]; then
    for file in ${@#$1}; do # All parameters except $1
        FILES="${FILES} ${network_devices}/ifconfig.${1}/${file}"
    done
elif [ -d "${network_devices}/ifconfig.${1}" ]; then
    FILES='echo ${network_devices}/ifconfig.${1}/*'
else
    FILES="${network_devices}/ifconfig.${1}"
fi

# Reverse the order configuration files are processed in
for file in ${FILES}; do
    FILES2="${file} ${FILES2}"
done
FILES=${FILES2}

# Process each configuration file
for file in ${FILES}; do
    # skip backup files
    if [ "${file}" != "${file%""~""}" ]; then
        continue
    fi

    if [ ! -f "${file}" ]; then
        boot_mesg "${file} is not a network configuration file or directory." ${WARNING}
        echo_warning
        continue
    fi
    (
        . ${file}

        # Will not process this service if started by boot, and ONBOOT
        # is not set to yes
        if [ "${IN_BOOT}" = "1" -a "${ONBOOT}" != "yes" ]; then
            continue
        fi

        # Will not process this service if started by hotplug, and
        # ONHOTPLUG is not set to yes
        if [ "${IN_HOTPLUG}" = "1" -a "${ONHOTPLUG}" != "yes" ]; then
            continue
        fi

        # This will run the service script, if SERVICE is set
        if [ -n "${SERVICE}" -a -x "${network_devices}/services/${SERVICE}" ]; then
            if ip link show ${1} > /dev/null 2>&1
            then
                IFCONFIG=${file} ${network_devices}/services/${SERVICE} ${1} down
            else
```

```
                boot_mesg "Interface ${1} doesn't exist." ${WARNING}
                echo_warning
            fi
        else
            boot_mesg -n "Unable to process ${file}.  Either" ${FAILURE}
            boot_mesg -n " the SERVICE variable was not set,"
            boot_mesg " or the specified service cannot be executed."
            echo_failure
            continue
        fi
    )
done

if [ -z "${2}" ]; then
    link_status='ip link show $1 2>/dev/null'
    if [ -n "${link_status}" ]; then
        if echo "${link_status}" | grep -q UP; then
            boot_mesg "Bringing down the ${1} interface..."
            ip link set ${1} down
            evaluate_retval
        fi
    fi
fi

# End $network_devices/ifdown
```

D.27. /etc/sysconfig/network-devices/services/ipv4-static

```
#!/bin/sh
########################################################################
# Begin $network_devices/services/ipv4-static
#
# Description : IPV4 Static Boot Script
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#        Kevin P. Fleming - kpfleming@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}
. ${IFCONFIG}

if [ -z "${IP}" ]; then
    boot_mesg "IP variable missing from ${IFCONFIG}, cannot continue." ${FAILURE}
    echo_failure
    exit 1
fi

if [ -z "${PREFIX}" -a -z "${PEER}" ]; then
    boot_mesg -n "PREFIX variable missing from ${IFCONFIG}," ${WARNING}
    boot_mesg " assuming 24."
    echo_warning
    PREFIX=24
    args="${args} ${IP}/${PREFIX}"
elif [ -n "${PREFIX}" -a -n "${PEER}" ]; then
```

```
    boot_mesg "PREFIX and PEER both specified in ${IFCONFIG}, cannot continue." ${FAILURE}
    echo_failure
    exit 1
elif [ -n "${PREFIX}" ]; then
    args="${args} ${IP}/${PREFIX}"
elif [ -n "${PEER}" ]; then
    args="${args} ${IP} peer ${PEER}"
fi

if [ -n "${BROADCAST}" ]; then
    args="${args} broadcast ${BROADCAST}"
fi

case "${2}" in
    up)
        boot_mesg "Adding IPv4 address ${IP} to the ${1} interface..."
        ip addr add ${args} dev ${1}
        evaluate_retval

        if [ -n "${GATEWAY}" ]; then
            if ip route | grep -q default; then
                boot_mesg "Gateway already setup; skipping." ${WARNING}
                echo_warning
            else
                boot_mesg "Setting up default gateway..."
                ip route add default via ${GATEWAY} dev ${1}
                evaluate_retval
             fi
        fi
    ;;

    down)
        if [ -n "${GATEWAY}" ];    then
            boot_mesg "Removing default gateway..."
            ip route del default
            evaluate_retval
        fi

        boot_mesg "Removing IPv4 address ${IP} from the ${1} interface..."
        ip addr del ${args} dev ${1}
        evaluate_retval
    ;;

    *)
        echo "Usage: ${0} [interface] {up|down}"
        exit 1
    ;;
esac

# End $network_devices/services/ipv4-static
```

D.28. /etc/sysconfig/network-devices/services/ipv4-static-route

```
#!/bin/sh
########################################################################
# Begin $network_devices/services/ipv4-static-route
#
# Description : IPV4 Static Route Script
#
```

```
# Authors     : Kevin P. Fleming - kpfleming@linuxfromscratch.org
#
# Version     : 00.00
#
# Notes       :
#
########################################################################

. /etc/sysconfig/rc
. ${rc_functions}
. ${IFCONFIG}

case "${TYPE}" in
    ("" | "network")
        need_ip=1
        need_gateway=1
    ;;

    ("default")
        need_gateway=1
        args="${args} default"
        desc="default"
    ;;

    ("host")
        need_ip=1
    ;;

    ("unreachable")
        need_ip=1
        args="${args} unreachable"
        desc="unreachable "
    ;;

    (*)
        boot_mesg "Unknown route type (${TYPE}) in ${IFCONFIG}, cannot continue." ${FAILURE}
        echo_failure
        exit 1
    ;;
esac

if [ -n "${need_ip}" ]; then
    if [ -z "${IP}" ]; then
        boot_mesg "IP variable missing from ${IFCONFIG}, cannot continue." ${FAILURE}
        echo_failure
        exit 1
    fi

    if [ -z "${PREFIX}" ]; then
        boot_mesg "PREFIX variable missing from ${IFCONFIG}, cannot continue." ${FAILURE}
        echo_failure
        exit 1
    fi

    args="${args} ${IP}/${PREFIX}"
    desc="${desc}${IP}/${PREFIX}"
fi

if [ -n "${need_gateway}" ]; then
    if [ -z "${GATEWAY}" ]; then
        boot_mesg "GATEWAY variable missing from ${IFCONFIG}, cannot continue." ${FAILURE}
```

```
        echo_failure
        exit 1
    fi
    args="${args} via ${GATEWAY}"
fi

if [ -n "${SOURCE}" ]; then
        args="${args} src ${SOURCE}"
fi

case "${2}" in
    up)
        boot_mesg "Adding '${desc}' route to the ${1} interface..."
        ip route add ${args} dev ${1}
        evaluate_retval
    ;;

    down)
        boot_mesg "Removing '${desc}' route from the ${1} interface..."
        ip route del ${args} dev ${1}
        evaluate_retval
    ;;

    *)
        echo "Usage: ${0} [interface] {up|down}"
        exit 1
    ;;
esac

# End $network_devices/services/ipv4-static-route
```

付録 E. Udev 設定ルール

本付録にて udev-config-20100128.tar.bz2 に含まれるルールを列記します。 インストール手順は 6.59. 「Udev-161」 を参照してください。

E.1. 55-lfs.rules

```
# /etc/udev/rules.d/55-lfs.rules: Rule definitions for LFS.

# Core kernel devices

# This causes the system clock to be set as soon as /dev/rtc becomes available.
SUBSYSTEM=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"
KERNEL=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"

# Comms devices

KERNEL=="ippp[0-9]*",       GROUP="dialout"
KERNEL=="isdn[0-9]*",       GROUP="dialout"
KERNEL=="isdnctrl[0-9]*",   GROUP="dialout"
KERNEL=="dcbri[0-9]*",      GROUP="dialout"
```

付録 F. LFS ライセンス

本ブックはクリエイティブコモンズ（Creative Commons）の 表示-非営利-継承（Attribution-NonCommercial-ShareAlike）2.0 ライセンスに従います。

本書のインストール手順のコマンドを抜き出したものは MIT ライセンスに従ってください。

F.1. クリエイティブコモンズライセンス

日本語訳情報

以下は日本語へ訳出することなく、原文のライセンス条項をそのまま示します。

Creative Commons Legal Code

Attribution-NonCommercial-ShareAlike 2.0

重要項目

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions
   a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
   b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
   c. "Licensor" means the individual or entity that offers the Work under the terms of this License.
   d. "Original Author" means the individual or entity who created the Work.
   e. "Work" means the copyrightable work of authorship offered under the terms of this License.
   f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
   g. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.
2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

b. to create and reproduce Derivative Works;

c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(e) and 4(f).

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

b. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-NonCommercial-ShareAlike 2.0 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

d. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner;

provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

e. For the avoidance of doubt, where the Work is a musical composition:

   i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

   ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation. 6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

重要項目

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at http://creativecommons.org/ .

F.2. MIT ライセンス（The MIT License）

日本語訳情報

以下は日本語へ訳出することなく、原文のライセンス条項をそのまま示します。

Copyright © 1999-2010 Gerard Beekmans

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

項目別もくじ

パッケージ

Autoconf: 126
Automake: 127
Bash: 117
 ツール: 48
Binutils: 85
 ツール, 1回め: 31
 ツール, 2回め: 39
Bison: 112
Bootscripts: 172
 利用方法: 174
Bzip2: 128
 ツール: 49
Coreutils: 106
 ツール: 50
DejaGNU: 46
Diffutils: 130
 ツール: 51
E2fsprogs: 103
Expect: 44
File: 132
 ツール: 52
Findutils: 133
 ツール: 53
Flex: 134
Gawk: 131
 ツール: 54
GCC: 91
 ツール, 1回め: 33
 ツール, 2回め: 40
GDBM: 120
Gettext: 136
 ツール: 55
Glibc: 76
 ツール: 36
GMP: 87
Grep: 114
 ツール: 56
Groff: 138
GRUB: 140
Gzip: 141
 ツール: 57
Iana-Etc: 110
Inetutils: 121
IPRoute2: 142
Kbd: 144
Less: 146
Libtool: 119
Linux: 191
 API ヘッダ: 74
 ツール, API ヘッダ: 35
M4: 111
 ツール: 58
Make: 147
 ツール: 59
Man-DB: 148
Man-pages: 75
Module-Init-Tools: 151
MPC: 90
MPFR: 89
Ncurses: 97
 ツール: 47
Patch: 152
 ツール: 60
Perl: 123
 ツール: 61
Pkg-config: 96
Procps: 113
Psmisc: 153
Readline: 115
Sed: 95
 ツール: 62
Shadow: 154
 設定: 154
Sysklogd: 157
 設定: 157
Sysvinit: 158
 設定: 159
Tar: 161
 ツール: 63
Tcl: 43
Texinfo: 162
 ツール: 64
Udev: 164
 利用方法: 181
Util-linux-ng: 99
Vim: 166
Zlib: 84

プログラム

a2p: 123, 124
accessdb: 148, 149
acinstall: 127, 127
aclocal: 127, 127
aclocal-1.11.1: 127, 127
addftinfo: 138, 138
addpart: 99, 99
addr2line: 85, 86
afmtodit: 138, 138
agetty: 99, 99
apropos: 148, 149
ar: 85, 86
arch: 99, 99
as: 85, 86
ata_id: 164, 165
autoconf: 126, 126
autoheader: 126, 126
autom4te: 126, 126
automake: 127, 127
automake-1.11.1: 127, 127
autopoint: 136, 136
autoreconf: 126, 126
autoscan: 126, 126
autoupdate: 126, 126
awk: 131, 131
badblocks: 103, 104

base64: 106, 107
basename: 106, 107
bash: 117, 117
bashbug: 117, 118
bigram: 133, 133
bison: 112, 112
blkid: 99, 99
blockdev: 99, 100
bootlogd: 158, 159
bunzip2: 128, 128
bzcat: 128, 128
bzcmp: 128, 128
bzdiff: 128, 129
bzegrep: 128, 129
bzfgrep: 128, 129
bzgrep: 128, 129
bzip2: 128, 129
bzip2recover: 128, 129
bzless: 128, 129
bzmore: 128, 129
c++: 91, 94
c++filt: 85, 86
c2ph: 123, 124
cal: 99, 100
captoinfo: 97, 98
cat: 106, 107
catchsegv: 76, 79
catman: 148, 149
cc: 91, 94
cdrom_id: 164, 165
cfdisk: 99, 100
chage: 154, 155
chattr: 103, 104
chcon: 106, 107
chem: 138, 138
chfn: 154, 155
chgpasswd: 154, 155
chgrp: 106, 107
chkdupexe: 99, 100
chmod: 106, 107
chown: 106, 107
chpasswd: 154, 155
chroot: 106, 107
chrt: 99, 100
chsh: 154, 155
chvt: 144, 144
cksum: 106, 107
clear: 97, 98
cmp: 130, 130
code: 133, 133
col: 99, 100
colcrt: 99, 100
collect: 164, 165
colrm: 99, 100
column: 99, 100
comm: 106, 107
compile: 127, 127
compile_et: 103, 104
config.charset: 136, 136
config.guess: 127, 127
config.rpath: 136, 136
config.sub: 127, 127
config_data: 123, 124
corelist: 123, 124
cp: 106, 107
cpan: 123, 124
cpan2dist: 123, 124
cpanp: 123, 124
cpanp-run-perl: 123, 124
cpp: 91, 94
create_floppy_devices: 164, 165
csplit: 106, 107
ctrlaltdel: 99, 100
ctstat: 142, 142
cut: 106, 107
cytune: 99, 100
date: 106, 107
dd: 106, 107
ddate: 99, 100
deallocvt: 144, 145
debugfs: 103, 104
delpart: 99, 100
depcomp: 127, 127
depmod: 151, 151
df: 106, 107
diff: 130, 130
diff3: 130, 130
dir: 106, 107
dircolors: 106, 108
dirname: 106, 108
dmesg: 99, 100
dprofpp: 123, 124
du: 106, 108
dumpe2fs: 103, 104
dumpkeys: 144, 145
e2freefrag: 103, 104
e2fsck: 103, 104
e2image: 103, 104
e2initrd_helper: 103, 104
e2label: 103, 104
e2undo: 103, 104
echo: 106, 108
edd_id: 164, 165
egrep: 114, 114
elisp-comp: 127, 127
enc2xs: 123, 124
env: 106, 108
envsubst: 136, 136
eqn: 138, 138
eqn2graph: 138, 138
ex: 166, 167
expand: 106, 108
expect: 44, 45
expiry: 154, 155
expr: 106, 108
factor: 106, 108
faillog: 154, 155
fallocate: 99, 100
false: 106, 108
fdformat: 99, 100
fdisk: 99, 100
fgconsole: 144, 145

fgrep: 114, 114
file: 132, 132
filefrag: 103, 104
find: 133, 133
find2perl: 123, 124
findfs: 99, 100
findmnt: 99, 100
firmware.sh: 164, 165
flex: 134, 134
flock: 99, 100
fmt: 106, 108
fold: 106, 108
frcode: 133, 133
free: 113, 113
fsck: 99, 100
fsck.cramfs: 99, 100
fsck.ext2: 103, 104
fsck.ext3: 103, 104
fsck.ext4: 103, 104
fsck.ext4dev: 103, 104
fsck.minix: 99, 100
fsfreeze: 99, 100
fstab-decode: 158, 159
fstab_import: 164, 165
ftp: 121, 121
fuser: 153, 153
g++: 91, 94
gawk: 131, 131
gawk-3.1.8: 131, 131
gcc: 91, 94
gccbug: 91, 94
gcov: 91, 94
gdiffmk: 138, 138
gencat: 76, 79
genl: 142, 142
geqn: 138, 138
getconf: 76, 79
getent: 76, 80
getkeycodes: 144, 145
getopt: 99, 100
gettext: 136, 136
gettext.sh: 136, 136
gettextize: 136, 136
gpasswd: 154, 155
gprof: 85, 86
grap2graph: 138, 138
grcat: 131, 131
grep: 114, 114
grn: 138, 138
grodvi: 138, 138
groff: 138, 138
groffer: 138, 138
grog: 138, 139
grolbp: 138, 139
grolj4: 138, 139
grops: 138, 139
grotty: 138, 139
groupadd: 154, 155
groupdel: 154, 156
groupmems: 154, 156
groupmod: 154, 156
groups: 106, 108
grpck: 154, 156
grpconv: 154, 156
grpunconv: 154, 156
grub-bin2h: 140, 140
grub-editenv: 140, 140
grub-install: 140, 140
grub-mkconfig: 140, 140
grub-mkdevicemap: 140, 140
grub-mkelfimage: 140, 140
grub-mkimage: 140, 140
grub-mkisofs: 140, 140
grub-mkpasswd-pbkdf2: 140, 140
grub-mkrelpath: 140, 140
grub-mkrescue: 140, 140
grub-probe: 140, 140
grub-reboot: 140, 140
grub-script-check: 140, 140
grub-set-default: 140, 140
grub-setup: 140, 140
gtbl: 138, 139
gunzip: 141, 141
gzexe: 141, 141
gzip: 141, 141
h2ph: 123, 124
h2xs: 123, 124
halt: 158, 159
head: 106, 108
hexdump: 99, 100
hostid: 106, 108
hostname: 121, 121
hostname: 136, 136
hpftodit: 138, 139
hwclock: 99, 100
i386: 99, 100
iconv: 76, 80
iconvconfig: 76, 80
id: 106, 108
ifcfg: 142, 142
ifnames: 126, 126
ifstat: 142, 142
igawk: 131, 131
indxbib: 138, 139
info: 162, 162
infocmp: 97, 98
infokey: 162, 162
infotocap: 97, 98
init: 158, 159
insmod: 151, 151
insmod.static: 151, 151
install: 106, 108
install-info: 162, 162
install-sh: 127, 127
instmodsh: 123, 124
ionice: 99, 100
ip: 142, 142
ipcmk: 99, 100
ipcrm: 99, 100
ipcs: 99, 100
isosize: 99, 100
join: 106, 108

kbdrate: 144, 145
kbd_mode: 144, 145
kill: 113, 113
killall: 153, 153
killall5: 158, 159
klogd: 157, 157
last: 158, 159
lastb: 158, 159
lastlog: 154, 156
ld: 85, 86
ldattach: 99, 100
ldconfig: 76, 80
ldd: 76, 80
lddlibc4: 76, 80
less: 146, 146
lessecho: 146, 146
lesskey: 146, 146
lex: 134, 135
lexgrog: 148, 149
lfskernel-2.6.35.4: 191, 193
libnetcfg: 123, 124
libtool: 119, 119
libtoolize: 119, 119
line: 99, 100
link: 106, 108
linux32: 99, 100
linux64: 99, 100
lkbib: 138, 139
ln: 106, 108
lnstat: 142, 143
loadkeys: 144, 145
loadunimap: 144, 145
locale: 76, 80
localedef: 76, 80
locate: 133, 133
logger: 99, 100
login: 154, 156
logname: 106, 108
logoutd: 154, 156
logsave: 103, 104
look: 99, 100
lookbib: 138, 139
losetup: 99, 100
ls: 106, 108
lsattr: 103, 104
lscpu: 99, 101
lsmod: 151, 151
m4: 111, 111
make: 147, 147
makeinfo: 162, 162
man: 148, 150
mandb: 148, 150
manpath: 148, 150
mapscrn: 144, 145
mcookie: 99, 101
md5sum: 106, 108
mdate-sh: 127, 127
mesg: 158, 159
missing: 127, 127
mkdir: 106, 108
mke2fs: 103, 104
mkfifo: 106, 108
mkfs: 99, 101
mkfs.bfs: 99, 101
mkfs.cramfs: 99, 101
mkfs.ext2: 103, 104
mkfs.ext3: 103, 104
mkfs.ext4: 103, 105
mkfs.ext4dev: 103, 105
mkfs.minix: 99, 101
mkinstalldirs: 127, 127
mklost+found: 103, 105
mknod: 106, 108
mkswap: 99, 101
mktemp: 106, 108
mk_cmds: 103, 104
mmroff: 138, 139
modinfo: 151, 151
modprobe: 151, 151
more: 99, 101
mount: 99, 101
mountpoint: 158, 159
msgattrib: 136, 136
msgcat: 136, 136
msgcmp: 136, 136
msgcomm: 136, 136
msgconv: 136, 136
msgen: 136, 136
msgexec: 136, 136
msgfilter: 136, 137
msgfmt: 136, 137
msggrep: 136, 137
msginit: 136, 137
msgmerge: 136, 137
msgunfmt: 136, 137
msguniq: 136, 137
mtrace: 76, 80
mv: 106, 108
namei: 99, 101
ncursesw5-config: 97, 98
neqn: 138, 139
newgrp: 154, 156
newusers: 154, 156
ngettext: 136, 137
nice: 106, 108
nl: 106, 108
nm: 85, 86
nohup: 106, 108
nologin: 154, 156
nproc: 106, 108
nroff: 138, 139
nscd: 76, 80
nstat: 142, 143
objcopy: 85, 86
objdump: 85, 86
od: 106, 108
oldfind: 133, 133
openvt: 144, 145
partx: 99, 101
passwd: 154, 156
paste: 106, 108
patch: 152, 152

pathchk: 106, 108
path_id: 164, 165
pcprofiledump: 76, 80
pdfroff: 138, 139
pdftexi2dvi: 162, 162
peekfd: 153, 153
perl: 123, 124
perl5.12.1: 123, 124
perlbug: 123, 124
perldoc: 123, 124
perlivp: 123, 124
perlthanks: 123, 124
pfbtops: 138, 139
pg: 99, 101
pgawk: 131, 131
pgawk-3.1.8: 131, 131
pgrep: 113, 113
pic: 138, 139
pic2graph: 138, 139
piconv: 123, 124
pidof: 158, 159
ping: 121, 121
ping6: 121, 122
pinky: 106, 108
pivot_root: 99, 101
pkg-config: 96, 96
pkill: 113, 113
pl2pm: 123, 124
pmap: 113, 113
pod2html: 123, 124
pod2latex: 123, 124
pod2man: 123, 124
pod2text: 123, 124
pod2usage: 123, 124
podchecker: 123, 124
podselect: 123, 124
post-grohtml: 138, 139
poweroff: 158, 159
pr: 106, 108
pre-grohtml: 138, 139
preconv: 138, 139
printenv: 106, 108
printf: 106, 108
prove: 123, 124
prtstat: 153, 153
ps: 113, 113
psed: 123, 125
psfaddtable: 144, 145
psfgettable: 144, 145
psfstriptable: 144, 145
psfxtable: 144, 145
pstree: 153, 153
pstree.x11: 153, 153
pstruct: 123, 125
ptar: 123, 125
ptardiff: 123, 125
ptx: 106, 108
pt_chown: 76, 80
pwcat: 131, 131
pwck: 154, 156
pwconv: 154, 156
pwd: 106, 108
pwdx: 113, 113
pwunconv: 154, 156
py-compile: 127, 127
ranlib: 85, 86
rcp: 121, 122
readelf: 85, 86
readlink: 106, 108
readprofile: 99, 101
reboot: 158, 160
recode-sr-latin: 136, 137
refer: 138, 139
rename: 99, 101
renice: 99, 101
reset: 97, 98
resize2fs: 103, 105
resizecons: 144, 145
rev: 99, 101
rexec: 121, 122
rlogin: 121, 122
rm: 106, 109
rmdir: 106, 109
rmmod: 151, 151
rmt: 161, 161
roff2dvi: 138, 139
roff2html: 138, 139
roff2pdf: 138, 139
roff2ps: 138, 139
roff2text: 138, 139
roff2x: 138, 139
routef: 142, 143
routel: 142, 143
rpcgen: 76, 80
rpcinfo: 76, 80
rsh: 121, 122
rtacct: 142, 143
rtcwake: 99, 101
rtmon: 142, 143
rtpr: 142, 143
rtstat: 142, 143
runcon: 106, 109
runlevel: 158, 160
runtest: 46, 46
rview: 166, 167
rvim: 166, 167
s2p: 123, 125
script: 99, 101
scriptreplay: 99, 101
scsi_id: 164, 165
sdiff: 130, 130
sed: 95, 95
seq: 106, 109
setarch: 99, 101
setfont: 144, 145
setkeycodes: 144, 145
setleds: 144, 145
setmetamode: 144, 145
setsid: 99, 101
setterm: 99, 101
sfdisk: 99, 101
sg: 154, 156

sh: 117, 118
sha1sum: 106, 109
sha224sum: 106, 109
sha256sum: 106, 109
sha384sum: 106, 109
sha512sum: 106, 109
shasum: 123, 125
showconsolefont: 144, 145
showkey: 144, 145
shred: 106, 109
shuf: 106, 109
shutdown: 158, 160
size: 85, 86
skill: 113, 113
slabtop: 113, 113
sleep: 106, 109
sln: 76, 80
snice: 113, 113
soelim: 138, 139
sort: 106, 109
splain: 123, 125
split: 106, 109
sprof: 76, 80
ss: 142, 143
stat: 106, 109
stdbuf: 106, 109
strings: 85, 86
strip: 85, 86
stty: 106, 109
su: 154, 156
sulogin: 158, 160
sum: 106, 109
swaplabel: 99, 101
swapoff: 99, 101
swapon: 99, 101
switch_root: 99, 101
symlink-tree: 127, 127
sync: 106, 109
sysctl: 113, 113
syslogd: 157, 157
tac: 106, 109
tail: 106, 109
tailf: 99, 101
talk: 121, 122
tar: 161, 161
taskset: 99, 101
tbl: 138, 139
tc: 142, 143
tclsh: 43, 43
tclsh8.5: 43, 43
tee: 106, 109
telinit: 158, 160
telnet: 121, 122
test: 106, 109
texi2dvi: 162, 163
texi2pdf: 162, 163
texindex: 162, 163
tfmtodit: 138, 139
tftp: 121, 122
tic: 97, 98
timeout: 106, 109
tload: 113, 113
toe: 97, 98
top: 113, 113
touch: 106, 109
tput: 97, 98
tr: 106, 109
traceroute: 121, 122
troff: 138, 139
true: 106, 109
truncate: 106, 109
tset: 97, 98
tsort: 106, 109
tty: 106, 109
tune2fs: 103, 105
tunelp: 99, 101
tzselect: 76, 80
udevadm: 164, 165
udevd: 164, 165
ul: 99, 101
umount: 99, 101
uname: 106, 109
uncompress: 141, 141
unexpand: 106, 109
unicode_start: 144, 145
unicode_stop: 144, 145
uniq: 106, 109
unlink: 106, 109
unshare: 99, 101
updatedb: 133, 133
uptime: 113, 113
usb_id: 164, 165
useradd: 154, 156
userdel: 154, 156
usermod: 154, 156
users: 106, 109
utmpdump: 158, 160
uuidd: 99, 101
uuidgen: 99, 101
vdir: 106, 109
vi: 166, 167
view: 166, 167
vigr: 154, 156
vim: 166, 168
vimdiff: 166, 168
vimtutor: 166, 168
vipw: 154, 156
vmstat: 113, 113
w: 113, 113
wall: 99, 101
watch: 113, 113
wc: 106, 109
whatis: 148, 150
whereis: 99, 101
who: 106, 109
whoami: 106, 109
wipefs: 99, 101
write: 99, 102
write_cd_rules: 164, 165
write_net_rules: 164, 165
xargs: 133, 133
xgettext: 136, 137

xsubpp: 123, 125
xtrace: 76, 80
xxd: 166, 168
yacc: 112, 112
yes: 106, 109
ylwrap: 127, 127
zcat: 141, 141
zcmp: 141, 141
zdiff: 141, 141
zdump: 76, 80
zegrep: 141, 141
zfgrep: 141, 141
zforce: 141, 141
zgrep: 141, 141
zic: 76, 80
zless: 141, 141
zmore: 141, 141
znew: 141, 141
zsoelim: 148, 150

ライブラリ

ld.so: 76, 80
libanl: 76, 80
libasprintf: 136, 137
libbfd: 85, 86
libblkid: 99, 102
libBrokenLocale: 76, 80
libbsd-compat: 76, 80
libbz2*: 128, 129
libc: 76, 80
libcidn: 76, 80
libcom_err: 103, 105
libcrypt: 76, 80
libcurses: 97, 98
libdl: 76, 80
libe2p: 103, 105
libexpect-5.44: 44, 45
libext2fs: 103, 105
libfl.a: 134, 135
libform: 97, 98
libg: 76, 80
libgcc*: 91, 94
libgcov: 91, 94
libgdbm: 120, 120
libgettextlib: 136, 137
libgettextpo: 136, 137
libgettextsrc: 136, 137
libgmp: 87, 87
libgmpxx: 87, 88
libgomp: 91, 94
libhistory: 115, 115
libiberty: 85, 86
libieee: 76, 80
libltdl: 119, 119
libm: 76, 80
libmagic: 132, 132
libmcheck: 76, 80
libmemusage: 76, 80
libmenu: 97, 98
libmp: 87, 88
libmpc: 90, 90
libmpfr: 89, 89
libmudflap*: 91, 94
libncurses: 97, 98
libnsl: 76, 80
libnss: 76, 80
libopcodes: 85, 86
libpanel: 97, 98
libpcprofile: 76, 80
libproc: 113, 113
libpthread: 76, 81
libreadline: 115, 116
libresolv: 76, 81
librpcsvc: 76, 81
librt: 76, 81
libSegFault: 76, 80
libss: 103, 105
libssp*: 91, 94
libstdbuf: 106, 109
libstdc++: 91, 94
libsupc++: 91, 94
libtcl8.5.so: 43, 43
libtclstub8.5.a: 43, 43
libthread_db: 76, 81
libudev: 164, 165
libutil: 76, 81
libuuid: 99, 102
liby.a: 112, 112
libz: 84, 84
preloadable_libintl: 136, 137

スクリプト

checkfs: 172, 172
cleanfs: 172, 172
console: 172, 172
  設定: 175
consolelog: 172, 172
  設定: 175
functions: 172, 172
halt: 172, 172
ifdown: 172, 172
ifup: 172, 172
localnet: 172, 172
  /etc/hosts: 186
  設定: 185
modules: 172, 172
mountfs: 172, 172
mountkernfs: 172, 172
network: 172, 172
  /etc/hosts: 186
  設定: 186
rc: 172, 172
reboot: 172, 172
sendsignals: 172, 172
setclock: 172, 172
  設定: 175
static: 172, 172
swap: 172, 172
sysctl: 172, 172
sysklogd: 172, 172

設定: 178
template: 172, 173
udev: 172, 173
udev_retry: 172, 173

その他

/boot/config-2.6.35.4: 191, 192
/boot/System.map-2.6.35.4: 191, 193
/dev/*: 67
/etc/fstab: 189
/etc/group: 72
/etc/hosts: 186
/etc/inittab: 159
/etc/inputrc: 178
/etc/ld.so.conf: 79
/etc/lfs-release: 198
/etc/localtime: 78
/etc/modprobe.d/usb.conf: 192
/etc/nsswitch.conf: 78
/etc/passwd: 72
/etc/profile: 179
/etc/protocols: 110
/etc/resolv.conf: 188
/etc/services: 110
/etc/syslog.conf: 157
/etc/udev: 164, 165
/etc/vimrc: 167
/usr/include/asm-generic/*.h: 74, 74
/usr/include/asm/*.h: 74, 74
/usr/include/drm/*.h: 74, 74
/usr/include/linux/*.h: 74, 74
/usr/include/mtd/*.h: 74, 74
/usr/include/rdma/*.h: 74, 74
/usr/include/scsi/*.h: 74, 74
/usr/include/sound/*.h: 74, 74
/usr/include/video/*.h: 74, 74
/usr/include/xen/*.h: 74, 74
/var/log/btmp: 72
/var/log/lastlog: 72
/var/log/wtmp: 72
/var/run/utmp: 72
man pages: 75, 75