

Linux From Scratch

Version 11.2

2022/09/01 公開

製作：Gerard Beekmans
編集総括：Bruce Dubbs
日本語訳：松山 道夫

Linux From Scratch: Version 11.2: 2022/09/01 公開

: 製作: Gerard Beekmans, 編集総括: Bruce Dubbs, 、日本語訳: 松山 道夫
製作著作 © 1999-2022 Gerard Beekmans

Copyright © 1999-2022, 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 アーキテクチャー	viii
iv. 必要な知識	viii
v. LFS と各種標準	ix
vi. 各パッケージを用いる理由	x
vii. 本書の表記	xv
viii. 本書の構成	xvi
ix. 正誤情報とセキュリティアドバイス	xvi
x. 日本語訳について	xvi
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	3
1.4. 変更履歴 (日本語版)	6
1.5. 情報源	8
1.6. ヘルプ	8
II. ビルド作業のための準備	10
2. ホストシステムの準備	11
2.1. はじめに	11
2.2. ホストシステム要件	11
2.3. 作業段階ごとの LFS 構築	13
2.4. 新しいパーティションの生成	14
2.5. ファイルシステムの生成	15
2.6. 変数 \$LFS の設定	16
2.7. 新しいパーティションのマウント	17
3. パッケージとパッチ	18
3.1. はじめに	18
3.2. 全パッケージ	20
3.3. 必要なパッチ	26
4. 準備作業の仕上げ	28
4.1. はじめに	28
4.2. LFS ファイルシステムの限定的なディレクトリレイアウトの生成	28
4.3. LFS ユーザーの追加	28
4.4. 環境設定	29
4.5. SBU 値について	31
4.6. テストスイートについて	31
III. LFS クロスチェーンと一時的ツールの構築	32
重要な準備事項	xxxiii
i. はじめに	xxxiii
ii. ツールチェーンの技術的情報	xxxiii
iii. 全般的なコンパイル手順	xxxvii
5. クロスツールチェーンの構築	39
5.1. はじめに	39
5.2. Binutils-2.39 - 1回め	40
5.3. GCC-12.2.0 - 1回め	42
5.4. Linux-5.19.2 API ヘッダー	44
5.5. Glibc-2.36	45
5.6. GCC-12.2.0 から取り出した libstdc++	48
6. クロスコンパイルによる一時的ツール	49
6.1. はじめに	49
6.2. M4-1.4.19	50
6.3. Ncurses-6.3	51
6.4. Bash-5.1.16	53
6.5. Coreutils-9.1	54
6.6. Diffutils-3.8	55

6.7.	File-5.42	56
6.8.	Findutils-4.9.0	57
6.9.	Gawk-5.1.1	58
6.10.	Grep-3.7	59
6.11.	Gzip-1.12	60
6.12.	Make-4.3	61
6.13.	Patch-2.7.6	62
6.14.	Sed-4.8	63
6.15.	Tar-1.34	64
6.16.	Xz-5.2.6	65
6.17.	Binutils-2.39 - 2回め	66
6.18.	GCC-12.2.0 - 2回め	67
7.	chroot への移行と一時的ツールの追加ビルド	69
7.1.	はじめに	69
7.2.	所有者の変更	69
7.3.	仮想カーネルファイルシステムの準備	69
7.4.	Chroot 環境への移行	70
7.5.	ディレクトリの生成	70
7.6.	重要なファイルとシンボリックリンクの生成	71
7.7.	Gettext-0.21	74
7.8.	Bison-3.8.2	75
7.9.	Perl-5.36.0	76
7.10.	Python-3.10.6	77
7.11.	Texinfo-6.8	78
7.12.	Util-linux-2.38.1	79
7.13.	一時的システムのクリーンアップと保存	80
IV.	LFSシステムの構築	82
8.	基本的なソフトウェアのインストール	83
8.1.	はじめに	83
8.2.	パッケージ管理	83
8.3.	Man-pages-5.13	87
8.4.	Iana-Etc-20220812	88
8.5.	Glibc-2.36	89
8.6.	Zlib-1.2.12	96
8.7.	Bzip2-1.0.8	97
8.8.	Xz-5.2.6	99
8.9.	Zstd-1.5.2	101
8.10.	File-5.42	102
8.11.	Readline-8.1.2	103
8.12.	M4-1.4.19	104
8.13.	Bc-6.0.1	105
8.14.	Flex-2.6.4	106
8.15.	Tcl-8.6.12	107
8.16.	Expect-5.45.4	109
8.17.	DejaGNU-1.6.3	110
8.18.	Binutils-2.39	111
8.19.	GMP-6.2.1	114
8.20.	MPFR-4.1.0	116
8.21.	MPC-1.2.1	117
8.22.	Attr-2.5.1	118
8.23.	Acl-2.3.1	119
8.24.	Libcap-2.65	120
8.25.	Shadow-4.12.2	121
8.26.	GCC-12.2.0	125
8.27.	Pkg-config-0.29.2	129
8.28.	Ncurses-6.3	130
8.29.	Sed-4.8	133
8.30.	Psmisc-23.5	134
8.31.	Gettext-0.21	135
8.32.	Bison-3.8.2	137
8.33.	Grep-3.7	138

8.34.	Bash-5.1.16	139
8.35.	Libtool-2.4.7	140
8.36.	GDBM-1.23	141
8.37.	Gperf-3.1	142
8.38.	Expat-2.4.8	143
8.39.	Inetutils-2.3	144
8.40.	Less-590	146
8.41.	Perl-5.36.0	147
8.42.	XML::Parser-2.46	149
8.43.	Intltool-0.51.0	150
8.44.	Autoconf-2.71	151
8.45.	Automake-1.16.5	152
8.46.	OpenSSL-3.0.5	153
8.47.	Kmod-30	155
8.48.	Elfutils-0.187 から取り出した libelf	157
8.49.	Libffi-3.4.2	158
8.50.	Python-3.10.6	159
8.51.	Wheel-0.37.1	161
8.52.	Ninja-1.11.0	162
8.53.	Meson-0.63.1	163
8.54.	Coreutils-9.1	164
8.55.	Check-0.15.2	168
8.56.	Diffutils-3.8	169
8.57.	Gawk-5.1.1	170
8.58.	Findutils-4.9.0	171
8.59.	Groff-1.22.4	172
8.60.	GRUB-2.06	174
8.61.	Gzip-1.12	176
8.62.	IPRoute2-5.19.0	177
8.63.	Kbd-2.5.1	179
8.64.	Libpipeline-1.5.6	181
8.65.	Make-4.3	182
8.66.	Patch-2.7.6	183
8.67.	Tar-1.34	184
8.68.	Texinfo-6.8	185
8.69.	Vim-9.0.0228	187
8.70.	Eudev-3.2.11	190
8.71.	Man-DB-2.10.2	191
8.72.	Procps-ng-4.0.0	194
8.73.	Util-linux-2.38.1	196
8.74.	E2fsprogs-1.46.5	201
8.75.	Sysklogd-1.5.1	204
8.76.	Sysvinit-3.04	205
8.77.	デバッグシンボルについて	206
8.78.	ストリップ	206
8.79.	仕切り直し	208
9.	システム設定	209
9.1.	はじめに	209
9.2.	LFS-ブートスクリプト-20220723	210
9.3.	デバイスとモジュールの扱いについて	212
9.4.	デバイスの管理	215
9.5.	全般的なネットワークの設定	217
9.6.	System V ブートスクリプトの利用と設定	219
9.7.	Bash シェルの初期起動ファイル	227
9.8.	/etc/inputrc ファイルの生成	228
9.9.	/etc/shells ファイルの生成	229
10.	LFS システムのブート設定	231
10.1.	はじめに	231
10.2.	/etc/fstab ファイルの生成	231
10.3.	Linux-5.19.2	233
10.4.	GRUB を用いたブートプロセスの設定	238

11. 作業終了	241
11.1. 作業終了	241
11.2. ユーザー登録	241
11.3. システムの再起動	241
11.4. 今度は何?	242
V. 付録	244
A. 略語と用語	245
B. 謝辞	247
C. パッケージの依存関係	249
D. ブートスクリプトと sysconfig スクリプト version-20220723	261
D.1. /etc/rc.d/init.d/rc	261
D.2. /lib/lsb/init-functions	265
D.3. /etc/rc.d/init.d/mountvirtfs	278
D.4. /etc/rc.d/init.d/modules	279
D.5. /etc/rc.d/init.d/udev	281
D.6. /etc/rc.d/init.d/swap	282
D.7. /etc/rc.d/init.d/setclock	283
D.8. /etc/rc.d/init.d/checkfs	284
D.9. /etc/rc.d/init.d/mountfs	287
D.10. /etc/rc.d/init.d/udev_retry	288
D.11. /etc/rc.d/init.d/cleanfs	289
D.12. /etc/rc.d/init.d/console	291
D.13. /etc/rc.d/init.d/localnet	293
D.14. /etc/rc.d/init.d/sysctl	294
D.15. /etc/rc.d/init.d/sysklogd	295
D.16. /etc/rc.d/init.d/network	297
D.17. /etc/rc.d/init.d/sendsignals	298
D.18. /etc/rc.d/init.d/reboot	299
D.19. /etc/rc.d/init.d/halt	300
D.20. /etc/rc.d/init.d/template	301
D.21. /etc/sysconfig/modules	302
D.22. /etc/sysconfig/createfiles	303
D.23. /etc/sysconfig/udev_retry	303
D.24. /sbin/ifup	304
D.25. /sbin/ifdown	306
D.26. /lib/services/ipv4-static	308
D.27. /lib/services/ipv4-static-route	309
E. Udev 設定ルール	312
E.1. 55-lfs.rules	312
F. LFS ライセンス	313
F.1. クリエイティブコモンズライセンス	313
F.2. MIT ライセンス (The MIT License)	316
項目別もくじ	317

序文

はしがき

私が Linux について学び始め理解するようになったのは 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りました。

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、さまざま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はたくさんあるものです。したがって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ました。何かが気に入らなかつたら、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となったのです。

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はありませんでした。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ます。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。そして私は自分自身の Linux を作り出して、自分の好みを満足させるものを目指しました。

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。さらに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か、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。我々は皆さんが旅に挑戦することを応援します。

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

本書を読む理由はさまざま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、さまざま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が。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事は、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 を構築する理由はさまざまですから、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 アーキテクチャー

LFS が対象としている CPU アーキテクチャーは AMD/インテル x86 CPU (32ビット) と x86_64 CPU (64ビット) です。Power PC や ARM については、本書の手順を多少修正することで動作することが確認されています。これらの CPU を利用したシステムをビルドする場合は、この後に示す諸条件を満たす必要がありますが、まずはそのアーキテクチャーをターゲットとする、LFS システムそのものや Ubuntu、Red Hat/Fedora、SuSE などの Linux システムが必要です。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。

LFS の構築において 64 ビットシステムを用いることは 32 ビットシステムを用いた場合に比べて大きな効果はありません。たとえば Core i7-4790 CPU 上において、4 コアを使って試しに LFS-9.1 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。

アーキテクチャー	ビルド時間	ビルドサイズ
32 ビット	239.9 分	3.6 GB
64 ビット	233.2 分	4.4 GB

ご存知かと思いますが、同一ハードウェア上にて 64 ビットによりビルドを行っても、32 ビットのときのビルドに比べて 3% 早くなるだけで 22% は大きなものになります。仮に LFS を使って LAMP サーバーやファイアーウォールを実現しようとする場合、32 ビット CPU を用いても十分機能するかもしれませんが。一方 BLFS にあるパッケージの中には、ビルド時や実行時に 4GB 以上の RAM を必要としているものもあります。このため LFS をデスクトップ環境に利用するなら、64 ビットシステムにおいてビルドすることをお勧めします。

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。本書ではこの点を直接サポートしていません。この理由は、素直な Linux ベースシステムを構築するという LFS の教育的な目的とは合致しないからです。LFS/BLFS 編集者の中に、マルチライブラリを行う LFS フォークを構築している方もいます。これは <https://www.linuxfromscratch.org/~thomas/multilib/index.html> からアクセスすることができます。ただしこれは応用的なトピックです。

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、ひよ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。

- ソースコードからのインストール入門ガイド (Beginner's Guide to Installing from Source) <http://moi.vonos.net/linux/beginners-installing-from-source/>

このガイドは、ソフトウェアをソースコードからビルドするために必要な基本的スキルや技術をほど良くまとめています。

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。

- POSIX.1-2008
- Filesystem Hierarchy Standard (FHS) Version 3.0
- Linux Standard Base (LSB) Version 5.0 (2015)

LSB はさらに以下の4つの標準から構成されます。 コア (Core)、デスクトップ (Desktop)、ランタイム言語 (Runtime Languages)、画像処理 (Imaging) です。 また一般的な要求事項に加えて、アーキテクチャーに固有の要求事項もあります。 Gtk3 やグラフィックスという二項目に関しての試用も含んでいます。 LFS では前節にて示したように、各アーキテクチャー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、おおむね 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, Bc, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib
LSB デスクトップ:	なし
LSB ランタイム言語:	Perl, Python
LSB 画像処理:	なし
LSB Gtk3、LSB グラフィックス (試用):	なし

BLFS 提供のパッケージで LSB 要求に従うもの

LSB コア:	At, Batch (At の一部), Cpio, Ed, Fcfrontab, LSB-Tools, NSPR, NSS, PAM, Pax, Sendmail (または Postfix または Exim), time
LSB デスクトップ:	Alsa, ATK, Cairo, Desktop-file-utils, Freetype, Fontconfig, Gdk-pixbuf, Glib2, GTK+2, Icon-naming-utils, Libjpeg-turbo, Libpng, Libtiff, Libxml2, MesaLib, Pango, Qt4, Xdg-utils, Xorg
LSB ランタイム言語:	Libxml2, Libxslt
LSB 画像処理:	CUPS, Cups-filters, Ghostscript, SANE
LSB Gtk3、LSB グラフィックス (試用):	GTK+3

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:	なし
---------	----

LSB デスクトップ:	Qt4 (Qt5 が提供されている)
LSB ランタイム言語:	なし
LSB 画像処理:	なし
LSB Gtk3、LSB グラフィックス (試用):	なし

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。

- **Acl**

このパッケージはアクセス制御リスト (Access Control Lists) を管理するツールを提供します。これはファイルやディレクトリに対して、きめ細かく様々なアクセス権限を定義するために利用されます。
- **Attr**

このパッケージはファイルシステムオブジェクト上の拡張属性を管理するプログラムを提供します。
- **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ける拡張性が高いからです。
- **Bc**

このパッケージは、任意精度 (arbitrary precision) の演算処理言語を提供します。Linux カーネルの構築に必要となります。
- **Binutils**

このパッケージは、リンカー、アセンブラーのような、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。
- **Check**

このパッケージは、他のプログラムに対するテストハーネス (test harness) を提供します。
- **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。
- **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。
- **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。
- **Eudev**
このパッケージはデバイスマネージャーです。 /dev ディレクトリに登録されたデバイスの所有者、パーミッション、名称、シンボリックリンクを動的に制御します。これによりデバイスは、システムへの追加または削除が行われず。
- **Expat**
このパッケージは比較的小規模の XML 解析ライブラリを提供します。XML-Parser Perl モジュールがこれを必要とします。
- **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。
- **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。他のパッケージのビルドスクリプト内にて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ー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。
- **Gperf**
このパッケージは、キーセットから完全なハッシュ関数を生成するプログラムを提供します。Eudev がこれを必要としています。

- 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。
- Intltool

本パッケージはソースファイルから翻訳対象となる文字列を抽出するツールを提供します。
- 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。
- Kmod

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

このパッケージは Linux カーネルにて利用される POSIX 1003.1e 機能へのユーザー空間からのインターフェースを実装します。
- Libelf

elfutils プロジェクトでは、ELF ファイルや DWARF データに対するライブラリやツールを提供しています。他のパッケージに対して各種ユーティリティは有用なものですが、ライブラリは Linux カーネルのビルドに必要であり、デフォルトの（最も効果的な）カーネル設定にて利用されます。
- Libffi

このパッケージは、さまざまな呼出規約 (calling conventions) に対しての、移植性に優れた高レベルプログラミングインターフェースを提供します。プログラムをコンパイルするその時点においては、関数に対してどのような引数が与えられるかが分からない場合があります。例えばインタープリターの場合、特定の関数を呼び出す際の引数の数や型は、実行時に指定されます。libffi はそういうプログラムであっても、インタープリタープログラムからコンパイルコードへのブリッジを提供します。
- Libpipeline

Libpipeline パッケージは、サブプロセスのパイプラインを柔軟にかつ容易に操作するライブラリを提供します。これは Man-DB パッケージが必要としています。
- 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せん。

- 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。

- Meson

このパッケージは、ソフトウェアを自動的にビルドするソフトウェアツールを提供します。Meson が目指すのは、ソフトウェア開発者がビルドシステムの設定にかかる時間を、できるだけ減らすことにあります。これは Systemd のビルドに必要であり、また BLFS における多くのパッケージにも必要です。

- 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。

- Ninja

このパッケージは、処理速度を重視した軽量なビルドシステムを提供します。高レベルなビルドシステムが生成したファイルを入力として、ビルド実行をできるだけ高速に行うように設計されています。このパッケージは Meson が必要としています。

- Ncurses

このパッケージは、端末に依存せず文字キャラクター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。

- Openssl

このパッケージは暗号化に関する管理ツールやライブラリを提供します。Linux カーネルや他のパッケージに対して、暗号化機能を提供するものとして有用です。

- 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ー言語 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、既にインストールされたライブラリやパッケージのメタデータを取得するプログラムを提供します。

- Procps-NG

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。

- Python 3

このパッケージは、ソースコードの可読性の向上を意図して開発されたインタープリター言語を提供します。

- 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、テキストエディター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。

- 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。

- Util-linux

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。

- Wheel

このパッケージは Python wheel パッケージング標準に基づいた標準実装の Python モジュールを提供します。

- Vim

このパッケージはテキストエディターを提供します。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。テキストエディターは個人により好みはさまざまですから、もし別のエディターを利用したいなら、そちらを用いても構いません。

- XML::Parser

このパッケージは Expat とのインターフェースを実現する Perl モジュールです。

- XZ Utils

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。一般的に用いられるものの中では高い圧縮率を実現するものであり、特に XZ フォーマットや LZMA フォーマットの伸張 (解凍) に利用されます。

- 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。

- Zstd

このパッケージは、一定のプログラムが利用している圧縮、伸張 (解凍) ルーチンを提供します。高圧縮率に加えて、圧縮、処理速度間のトレードオフを広範囲に提供します。

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。

注記

ブラウザの設定において、固定幅テキストに対しては適切なモノスペースフォントを用いるようにしてください。これを設定していれば、`l11` や `o0` のグリフを適切に識別できます。

Emphasis

上の表記はさまざま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。

<https://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` が表示されることとなります。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 クロスチェーンと一時的ツールの構築

第 III 部では、最終的な LFS システム構築のために必要となるツールのビルド説明を行います。

第 IV 部 - LFS システムの構築

第 IV 部では LFS システムの構築作業を順に説明していきます。こ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。

第 V 部 - 付録

第 V 部では、本書における略語や用語、謝辞、パッケージの依存関係、LFS ブートスクリプトの一覧、本書配布のライセンス、パッケージ、プログラム、ライブラリ、スクリプトのさくいんを示します。

正誤情報とセキュリティアドバイス

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 <https://www.linuxfromscratch.org/lfs/errata/11.2/> 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。

上に加えて Linux From Scratch 編集者は、本ブックのリリース後に発見されたセキュリティぜい弱性のリストを管理しています。ビルド作業に入る前には <https://www.linuxfromscratch.org/lfs/advisories/> にアクセスして、重大なセキュリティぜい弱性がないかどうかを事前に確認してください。LFS のビルド作業を進めていく上では、必ずセキュリティアドバイスを確認し、セキュリティぜい弱性を解消する修正手順があれば、それに従ってください。

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 11.2 の日本語版 20220901 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 OSDN.jp 内に開発の場を設け <http://lfsbookja.osdn.jp/> 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 <https://www.linuxfromscratch.org/> において、Stable 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、処理に必要な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ます、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きます。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ひます。

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, OpenMandriva, Fedora, openSUSE 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 構築のためにさまざま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、リンカー、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。

コンピューター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、他に提供されている LiveCD を利用することもできます。

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 構築に必要な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 章では初期のツールチェーン (binutils, gcc, glibc) を、クロスコンパイルによりインストールします。これによりこの新たなツールをホストシステムから切り離します。

第 6 章では、上で作ったクロスツールチェーンを利用して、基本的ユーティリティのクロスコンパイル方法を示します。

第 7 章では "chroot" 環境に入ります。そして今作り上げたビルドツールを使って、最終的なシステムをビルドしテストするために必要となる残りのツールをビルドします。

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が。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 8 章において完全な LFS システムが出来上がります。chroot を使うもう一つのメリットは、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 9 章にてベースシステムの設定を行い、第 10 章にてカーネルとブートローダーを設定します。第 11 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戦して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:

-
- Bc 6.0.1
- Binutils-2.39
- Coreutils-9.1
- E2fsprogs-1.46.5
- Expat-2.4.8
- File-5.42
- GCC-12.2.0
- Glibc-2.36
- Gzip-1.12
- IANA-Etc-20220812
- Inetutils-2.3

- IPRoute2-5.19.0
- Kbd-2.5.1
- Kmod-30
- Libcap-2.65
- Libelf-0.187 (from elfutils)
- Libpipeline-1.5.6
- Libtool-2.4.7
- Linux-5.19.2
- Man-DB-2.10.2
- Meson-0.63.1
- Ninja-1.11.0
- Openssl-3.0.5
- Perl-5.36.0
- Procps-ng-4.0.0
- Psmisc-23.5
- Python-3.10.6
- Shadow-4.12.2
- SysVinit-3.04
- Tzdata-2022c
- Util-Linux-2.38.1
- Vim-9.0.0228
- XZ-Uutils-5.2.6
- Zlib-1.2.12

追加:

-
- Wheel-0.37.1
- zstd-1.5.2-upstream_fixes-1.patch

削除:

-
- perl-5.34.0-upstream_fixes-1.patch
- systemd-250-kernel_5.17_fixes-1.patch
- systemd-250-upstream_fixes-1.patch

1.3. 変更履歴

本書は Linux From Scratch ブック、バージョン 11.2、2022/09/01 公開です。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。以下のミラーサイトを確認してください。 <https://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

- 2022-09-01
 - [bdubbs] - LFS-11.2 リリース
- 2022-08-20
 - [bdubbs] - vim-9.0.0228 へのアップデート。 #4500 にて言及。
 - [bdubbs] - iana-etc-20220812 へのアップデート。 #5006 にて言及。
 - [bdubbs] - gcc-12.2.0 へのアップデート。 #5098 を Fix に。
 - [bdubbs] - linux-5.19.2 (セキュリティフィックス) へのアップデート。 #5097 を Fix に。

- [bdubbs] - tzdata-2022c へのアップデート。 #5096 を Fix に。
- [bdubbs] - shadow-4.12.2 (セキュリティフィックス) へのアップデート。 #5095 を Fix に。
- [bdubbs] - meson-0.63.1 へのアップデート。 #5094 を Fix に。
- [bdubbs] - xz-5.2.6 へのアップデート。 #5093 を Fix に。
- 2022-08-18
 - [xryl11] - 5 章 6 章において libtool アーカイブファイル (.la) を削除。 クロスコンパイルにとっては邪魔になるため。
- 2022-08-11
 - [bdubbs] - vim-9.0.0192 へのアップデート。 #4500 において言及。
 - [bdubbs] - iana-etc-20220803 へのアップデート。 #5006 において言及。
 - [bdubbs] - tzdata-2022b へのアップデート。 #5092 を Fix に。
 - [bdubbs] - iproute2-5.19.0 へのアップデート。 #5086 を Fix に。
 - [bdubbs] - linux-5.19.1 へのアップデート。 #5085 を Fix に。
 - [bdubbs] - bc-6.0.1 へのアップデート。 #5090 を Fix に。
- 2022-08-06
 - [bdubbs] - binutils-2.39 へのアップデート。 #5089 を Fix に。
 - [bdubbs] - util-linux-2.38.1 へのアップデート。 #5088 を Fix に。
 - [bdubbs] - Python3-3.10.6 へのアップデート。 #5087 を Fix に。
 - [bdubbs] - glibc-2.36 へのアップデート。 #5084 を Fix に。
- 2022-07-24
 - [bdubbs] - linux-5.18.14 へのアップデート。 (RETbleed ぜい弱性の影響があるマシン向けのセキュリティフィックス) #5082 を Fix に。
 - [bdubbs] - libcap-2.65 へのアップデート。 #5083 を Fix に。
- 2022-07-15
 - [bdubbs] - meson-0.63.0 へのアップデート。 #5079 を Fix に。
 - [bdubbs] - kmod-30 へのアップデート。 #5077 を Fix に。
 - [bdubbs] - inetutils-2.3 へのアップデート。 #5081 を Fix に。
 - [bdubbs] - linux-5.18.11 へのアップデート。 #5078 を Fix に。
 - [bdubbs] - OpenSSL-3.0.5 へのアップデート。 #5080 を Fix に。
- 2022-07-01
 - [bdubbs] - vim-9.0.0006 へのアップデート。 #4500 において言及。
 - [bdubbs] - iana-etc-20220610 へのアップデート。 #5006 において言及。
 - [bdubbs] - OpenSSL-3.0.4 へのアップデート。 #5076 を Fix に。
 - [bdubbs] - kbd-2.5.1 へのアップデート。 #5075 を Fix に。
 - [bdubbs] - linux-5.18.8 へのアップデート。 #5074 を Fix に。
 - [bdubbs] - bc-5.3.3 へのアップデート。 #5073 を Fix に。
- 2022-06-29
 - [pierre] - ncurses における C++ バインディングに関して、スタティックなバインディングを生成してから削除するのではなく、共有バインディングを生成することに。
- 2022-06-14
 - [bdubbs] - python3-3.10.5 へのアップデート。 #5070 を Fix に。
 - [bdubbs] - meson-0.62.2 へのアップデート。 #5068 を Fix に。
 - [bdubbs] - kbd-2.5.0 へのアップデート。 #5067 を Fix に。
 - [bdubbs] - file-5.42 へのアップデート。 #5072 を Fix に。
 - [bdubbs] - linux-5.18.3 へのアップデート。 #5069 を Fix に。
 - [bdubbs] - bc-5.3.1 へのアップデート。 #5071 を Fix に。
- 2022-05-29

- [bdubbs] - iana-etc-20220524 へのアップデート。 #5006 において言及。
- [bdubbs] - iproute2-5.18.0 へのアップデート。 #5065 を Fix に。
- [bdubbs] - linux-5.18.1 へのアップデート。 #5061 を Fix に。
- [bdubbs] - perl-5.36.0 へのアップデート。 #5066 を Fix に。
- 2022-05-29
 - [xryl11] - vim-8.2.5014 へのアップデート (セキュリティフィックス)。 #4500 において言及。
 - [xryl11] - ninja-1.11.0 へのアップデート。 #5062 を Fix に。
 - [xryl11] - systemd-251 へのアップデート。 #5064 を Fix に。
- 2022-05-16
 - [bdubbs] - linux-5.17.7 へのアップデート。 #5059 を Fix に。
 - [bdubbs] - psmisc-23.5 へのアップデート。 #5060 を Fix に。
- 2022-05-01
 - [bdubbs] - openssl-3.0.3 へのアップデート。 #5057 を Fix に。
- 2022-05-01
 - [bdubbs] - nobody/nogroup の uid/gid を 65534 に変更。
 - [bdubbs] - meson-0.62.1 へのアップデート。 #5052 を Fix に。
 - [bdubbs] - libpipeline-1.5.6 へのアップデート。 #5053 を Fix に。
 - [bdubbs] - elfutils-0.187 へのアップデート。 #5054 を Fix に。
 - [bdubbs] - vim-8.2.4814 へのアップデート。 #4500 において言及。
 - [bdubbs] - sysvinit-3.03 へのアップデート。 #5046 を Fix に。
 - [bdubbs] - linux-5.17.5 へのアップデート。 #5050 を Fix に。
 - [bdubbs] - gcc-11.3.0 へのアップデート。 #5051 を Fix に。
 - [bdubbs] - coreutils-9.1 へのアップデート。 #5048 を Fix に。
 - [bdubbs] - bc-5.2.4 へのアップデート。 #5049 を Fix に。
- 2022-04-15
 - [bdubbs] - wheel-0.37.1 (Python モジュール) 追加。
- 2022-04-15
 - [ken] - Xz のセキュリティフィックスパッチ。 #5047 を Fix に。
 - [bdubbs] - libcap-2.64 へのアップデート。 #5044 を Fix に。
 - [bdubbs] - linux-5.17.3 へのアップデート。 #5043 を Fix に。
 - [bdubbs] - gzip-1.12 へのアップデート。 #5042 を Fix に。
- 2022-03-31
 - [bdubbs] - sysvinit-3.02 へのアップデート。 #5041 を Fix に。
 - [bdubbs] - zlib-1.2.12 へのアップデート (セキュリティアップデート)。 #5040 を Fix に。
 - [bdubbs] - expat-2.4.8 へのアップデート。 #5039 を Fix に。
 - [bdubbs] - Python-3.10.4 へのアップデート。 #5037 を Fix に。
 - [bdubbs] - procps-ng-4.0.0 へのアップデート。 #5036 を Fix に。
 - [bdubbs] - iproute2-5.17.0 へのアップデート。 #5035 を Fix に。
 - [bdubbs] - meson-0.62.0 へのアップデート。 #5034 を Fix に。
 - [bdubbs] - linux-5.17.1 へのアップデート (セキュリティアップデート)。 #5033 を Fix に。
 - [bdubbs] - util-linux-2.38 へのアップデート。 #4997 を Fix に。
- 2022-03-25
 - [pierre] - bootscripts を 20220324 にアップデート。 #5027 を Fix に。
- 2022-03-20
 - [xryl11] - Python-3.10.3 へのアップデート。 #5028 を Fix に。
 - [xryl11] - libtool-2.4.7 へのアップデート。 #5029 を Fix に。

- [xry111] - linux-5.16.16 へのアップデート。 #5030 を Fix に。
- [xry111] - tzdata-2022a へのアップデート。 #5031 を Fix に。
- [xry111] - man-db-2.10.2 へのアップデート。 #5032 を Fix に。
- 2022-03-16
 - [xry111] - MarkupSafe-2.1.1 へのアップデート。 #5025 を Fix に。
- 2022-03-15
 - [bdubbs] - openssl-3.0.2 へのアップデート。 #5024 を Fix に。
 - [bdubbs] - meson-0.61.3 へのアップデート。 #5023 を Fix に。
 - [xry111] - expat-2.4.7 へのアップデート。 #5019 を Fix に。
 - [xry111] - bc-5.2.3 へのアップデート。 #5020 を Fix に。
 - [xry111] - linux-5.16.14 (セキュリティフィックス) へのアップデート。 #5021 を Fix に。
 - [xry111] - perl-5.34.1 へのアップデート。 #5022 を Fix に。
 - [xry111] - vim-8.2.4567 (セキュリティフィックス) へのアップデート。 #4500 において言及。
- 2022-03-05
 - [xry111] - \$LFS/source の所有者を lfs に変更しないこととする。 #5018 を Fix に。
 - [xry111] - zstd-1.5.2 アップストリームの修正パッチを追加。
- 2022-03-02
 - [xry111] - meson-0.61.2 へのアップデート。 #5013 を Fix に。
 - [xry111] - linux-5.16.12 へのアップデート。 #5014 を Fix に。
 - [xry111] - MarkupSafe-2.1.0 へのアップデート。 #5015 を Fix に。
 - [xry111] - vim-8.2.4489 (セキュリティフィックス) へのアップデート。 #4500 において言及。
 - [xry111] - GCC 2 回めにおいて libstdc++ をビルドすることとし、個別の libstdc++ 2 回めを削除。
 - [xry111] - tcl における不要な `--enable-64bit` を削除。
- 2022-03-01
 - [bdubbs] - LFS-11.1 リリース。

1.4. 変更履歴（日本語版）

ここに示すのは LFS ブック 11.2 日本語版（バージョン20220901）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。

「r11.1-XXX」という表記は、オリジナル LFS ブック GIT 管理ソースの連番番号を意味します。また 6851fc8b2 などのリンクは、オリジナル XML ソースファイルの Git 管理下でのコミットハッシュ値（その参照ページ）を意味します。

変更履歴

- 2022-09-01
 - [matsuand] - LFS 11.2 リリース対応。 r11.1-189 (51b7349a9) までの対応。
- 2022-08-27
 - [matsuand] - r11.1-186 (bf6f9e75e) までの対応。
- 2022-08-26
 - [matsuand] - r11.1-182 (b3f157c68) までの対応。
- 2022-08-22
 - [matsuand] - r11.1-175 (fd6f71bd3) までの対応。
- 2022-08-20
 - [matsuand] - r11.1-171 (983647975) までの対応。

- [matsuand] - r11.1-168 (6099a4714) までの対応。
- 2022-08-12
- [matsuand] - r11.1-166 (c25dd3fc1) までの対応。
- 2022-08-10
- [matsuand] - r11.1-163 (960a230b4) までの対応。
- 2022-08-07
- [matsuand] - r11.1-158 (e06065fc2) までの対応。
- 2022-07-17
- [matsuand] - r11.1-151 (d060b3354) までの対応。
- 2022-06-25
- [matsuand] - r11.1-142 (bfc649552) までの対応。
- 2022-06-16
- [matsuand] - r11.1-135 (c7b29be1c) までの対応。
- 2022-06-02
- [matsuand] - r11.1-133 (9bc47a117) までの対応。
- 2022-05-17
- [matsuand] - r11.1-119 (07b9641ca) までの対応。
- 2022-05-12
- [matsuand] - r11.1-117 (59d5489ff) までの対応。
- 2022-04-22
- [matsuand] - r11.1-92 (93db1e614) までの対応。
- 2022-04-19
- [matsuand] - r11.1-84 (18e99c88b) までの対応。
- 2022-04-16
- [matsuand] - r11.1-76 (1d694184b) までの対応。
- 2022-04-13
- [matsuand] - r11.1-73 (b861051f1) までの対応。
- 2022-03-31
- [matsuand] - r11.1-69 (676f0fdce) までの対応。
- 2022-03-27
- [matsuand] - r11.1-65 (ab7af9e6f) までの対応。
- 2022-03-25
- [matsuand] - r11.1-50 (9b463a215) までの対応。
- 2022-03-22
- [matsuand] - r11.1-44 (53b26d62b) までの対応。
- 2022-03-17
- [matsuand] - r11.1-37 (43149b904) までの対応。
- 2022-03-15
- [matsuand] - r11.1-28 (f7ac150c8) までの対応。
- 2022-03-05
- [matsuand] - r11.1-20 (cb39502e1) までの対応。
- 2022-03-03
- [matsuand] - r11.1-14 (cc2c231b5) までの対応。
- 2022-03-01
- [matsuand] - LFS 11.1 リリース対応。 r11.0-203 (fe09af0cf) まで。

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 <https://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](https://www.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。 <https://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 <https://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.libera.chat](https://www.linuxfromscratch.org/irc/) において、チャンネル名 #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。 <https://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 <https://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が、そこに回答が示されていないのなら、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。 <https://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 <https://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「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 11.2)
- 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。
- ホストシステム要件 におけるスクリプトの出力結果。
- 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」ことは分かっていても「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 ホストシステムの準備

2.1. はじめに

この章では LFS システムの構築に必要なホストツールを確認し、必要に応じてインストールします。そして 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. 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ば、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ー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-2.13.1 (2.39 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-2.7 (/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。)
- Coreutils-6.9
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-4.0.1 (/usr/bin/awk が gawk へのリンクである必要があります。)
- GCC-4.8 と C++ コンパイラーである g++ (12.2.0 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。)

ホストされたプログラムを C++ コンパイラーがビルドできるように、C および C++ の標準ライブラリ (ヘッダーを含む) が存在しなければなりません。
- Grep-2.5.1a
- Gzip-1.3.12
- Linux Kernel-3.2

カーネルのバージョンを指定しているのは、第 5 章 と 第 8 章 において、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。これは udev においても必要になります。

ホストシステムのカーネルバージョンが 3.2 より古い場合は、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 3.2 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。もしそれが無い場合や、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) ブートローダーの設定方法については 第 10 章 を参照してください。

- M4-1.4.10
- Make-4.0
- Patch-2.5.4
- Perl-5.8.8
- Python-3.4
- Sed-4.1.5
- Tar-1.22
- Texinfo-4.7
- Xz-5.0.0



重要

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が、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
# Simple script to list version numbers of critical development tools
export LC_ALL=C
bash --version | head -n1 | cut -d" " -f2-4
MYSH=$(readlink -f /bin/sh)
echo "/bin/sh -> $MYSH"
echo $MYSH | grep -q bash || echo "ERROR: /bin/sh does not point to bash"
unset MYSH

echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-
bison --version | head -n1

if [ -h /usr/bin/yacc ]; then
    echo "/usr/bin/yacc -> `readlink -f /usr/bin/yacc`";
elif [ -x /usr/bin/yacc ]; then
    echo yacc is `/usr/bin/yacc --version | head -n1`
else
    echo "yacc not found"
fi

echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1

if [ -h /usr/bin/awk ]; then
    echo "/usr/bin/awk -> `readlink -f /usr/bin/awk`";
elif [ -x /usr/bin/awk ]; then
    echo awk is `/usr/bin/awk --version | head -n1`
else
    echo "awk not found"
fi

```

```

gcc --version | head -n1
g++ --version | head -n1
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`
python3 --version
sed --version | head -n1
tar --version | head -n1
makeinfo --version | head -n1 # texinfo version
xz --version | head -n1

echo 'int main(){}' > dummy.c && g++ -o dummy dummy.c
if [ -x dummy ]
  then echo "g++ compilation OK";
  else echo "g++ compilation failed"; fi
rm -f dummy.c dummy
EOF

bash version-check.sh

```

2.3. 作業段階ごとの LFS 構築

LFS は一度にすべてを構築するものとして説明を行っています。つまり作業途中でシステムをシャットダウンすることは想定していません。ただこれは、システム構築を立ち止まることなくやり続けろと言っているわけではありません。

LFS 構築を途中から再開する場合には、どの段階からなのかに応じて、特定の作業を再度行うことが必要となります。

2.3.1. 第 1 章～第 4 章

これらの章ではホストシステム上で作業を行います。作業を再開する際には以下に注意します。

- 2.4 節以降において root ユーザーにより実行する作業では LFS 環境変数の設定が必要です。さらにそれは root ユーザーにおいて設定されていなければなりません。

2.3.2. 第 5 章～第 6 章

- /mnt/lfs パーティションがマウントされていることが必要です。
- この 2 つの章における処理はすべて、ユーザー lfs により実施してください。処理の実施前には su - lfs を行ないます。これをやり忘れた場合、パッケージインストールをホストに対して行ってしまい、利用不能になってしまうリスクがあります。
- 一般的なコンパイル手順に示す内容は極めて重要です。パッケージのインストール作業に少しでも疑わしい点があったならば、展開作業を行った tarball やその展開ディレクトリをいったん消去し、再度展開し作業をやり直してください。

2.3.3. 第 7 章～第 10 章

- /mnt/lfs パーティションがマウントされていることが必要です。
- 「所有者の変更」から「Chroot 環境への移行」までの操作は、root ユーザーで行います。LFS 環境変数が root ユーザーにおいて設定されている必要があります。
- chroot 環境に入った際には、環境変数 LFS が root ユーザーにおいて設定されている必要があります。これ以降 LFS 変数は使いません。
- 仮想ファイルシステムがマウントされている必要があります。これは chroot 環境への移行前後において、ホストの仮想端末を変更することで実現します。root ユーザーとなって「/dev のマウントと有効化」と「仮想カーネルファイルシステムのマウント」を実行する必要があります。

2.4. 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 10 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、さらなる容量が必要となります。30 GB ほどのパーティションがあれば、増量していくことを考えても十分な容量でしょう。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あり、また LFS の完成後に機能追加していくためのものです。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が、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。例えばプライマリーディスクであれば `/dev/sda` 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。



注記

上級者の方であれば別のパーティション設定も可能です。最新の LFS システムは、ソフトウェア RAID アレーや、LVM 論理ボリュームを利用することができます。ただしこれらを実現するには `initramfs` が必要であり、高度なトピックです。こういったパーティション設定は、LFS 初心者にはお勧めしません。

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 `sda5` 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。

2.4.1. 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。

2.4.1.1. 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 20 GB もあれば、どんなシステムであつ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、実験的に複数パーティションを設けるとしても、これだけのサイズで十分です。

2.4.1.2. 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。

Linux のハイパーネーション (ディスクへの退避状態) 機能を利用する場合、マシンが停止する前に RAM の内容がスワップパーティションに書き出されます。この場合、スワップパーティションの容量は、システムの RAM 容量と最低でも同程度である必要があります。

スワップは好ましいことではありません。物理的なハードドライブの場合、スワップが発生しているかどうかは、単純にディスク音を聞いたり、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ればわかります。SSD ドライブの場合、スワップ時の音は聞こえてきません。その場合は `top` や `free` プログラムを使ってスワップ使用量を確認するこ

とができます。SSD ドライブにスワップパーティションを割り当てることは極力避けるべきです。最初は 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してみてください。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。

2.4.1.3. Grub バイオスパーティション

GUID パーティションテーブル (GUID Partition Table; GPT) を利用して ブートディスク をパーティショニングした場合、普通は 1 MB 程度の小さなパーティションをさらに用意しておく必要があります。このパーティションのフォーマットは不要であり、ブートローダーをインストールする際に GRUB が利用できるものでなければなりません。通常このパーティションは fdisk を用いた場合は 'BIOS Boot' と名付けられます。また gdisk を用いた場合は EF02 というコード名が与えられます。



注記

Grub バイオスパーティションは、BIOS がシステムブート時に用いるドライブ上になければなりません。これは LFS ルートパーティションがあるドライブと同一にする必要はありません。システム上にあるドライブは、同一のパーティションテーブルタイプを利用していないことがあります。つまりこの Grub バイオスパーティションに必要なのは、ブートディスクのパーティションテーブルタイプに合わせるだけです。

2.4.1.4. 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 200MB とすればそれで十分です。
- /boot/efi - EFI システムパーティションであり、UEFI を使ってシステム起動する場合に必要です。詳しくは BLFS ページ を参照してください。
- /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 - LFS においては /bin, /lib, /sbin の各ディレクトリは、/usr 配下からのシンボリックリンクとしています。したがって /usr には、システムを動作させるために必要となる実行モジュールがすべて置かれます。LFS において /usr を別パーティションとすることは、普通は不要です。それでもこれが必要となる場合、システム内のプログラムやライブラリすべてが収容できるように、そのパーティション容量を十分に確保することが必要です。root パーティションは、このような設定とするなら、極端に小さなサイズ (1 ギガバイト程度) でも十分です。これはシンクライアントやディスクなしワークステーションに適しています。(そういった環境では /usr がリモートサーバーにマウントされます。) ただし (LFS では対応していない) initramfs を利用する際には、これがブートする際に /usr が別パーティションになっていることが必要であるため、注意してください。
- /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「/etc/fstab ファイルの生成」で説明しています。

2.5. 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。LFS では Linux カーネルが識別できるならどのようなファイルシステムを用いるのでも構いません。ただ最も標準的なものとして ext3 と ext4 があります。ファイルシステムをどのようにするかは単純な話ではなく、収容するファイルの性質やパーティションサイズにも依存します。例えば以下のとおりです。

ext2

比較的小容量のパーティションで、/boot のようにあまり更新されないパーティションに対して適しています。

ext3

ext2 の拡張でありジャーナルを含みます。このジャーナルとは、不測のシャットダウン時などに、パーティション状態の復元に用いられます。汎用的なファイルシステムとして用いることができます。

ext4

パーティションタイプとして用いられる ext 系の最新バージョンです。新たな機能として、ナノ秒単位のタイムスタンプの提供、大容量ファイル (16 TB) の生成利用、処理性能の改善が加えられています。

この他のファイルシステムとして、FAT32, NTFS, ReiserFS, JFS, XFS などがあり、それぞれに特定の目的に応じて活用されています。ファイルシステムの詳細については http://en.wikipedia.org/wiki/Comparison_of_file_systems を参照してください。

LFS ではルートファイルシステム (/) として ext4 を用いるものとします。LFS 用のパーティションに対して ext4 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。

```
mkfs -v -t ext4 /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。

既存の swap パーティションを利用している場合は、初期化を行う必要はありません。新しく swap パーティションを生成した場合には、以下のコマンドにより初期化を行ってください。

```
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。

2.6. 変数 \$LFS の設定

本書の中では環境変数 LFS を何度も用います。LFS システムのビルド作業時には常に定義しておくことを忘れないでください。この変数は 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。例えば /mnt/lfs というものです。他のものとしても構いません。LFS を別のパーティションにビルドする場合、このマウントポイントはそのパーティションを示すようにしてください。ディレクトリを取り決めたら、変数を以下のコマンドにより設定します。

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

上のよう変数を定義しておくこと、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。



注意

\$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。

```
echo $LFS
```

上の出力結果が LFS システムのビルドディレクトリ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が、出力が正しくない場合は、冒頭に示したコマンド実行により \$LFS 変数に正しいディレクトリを設定してください。



注記

LFS 変数を確実に設定しておくために、ローカルな .bash_profile および /root/.bash_profile に上記変数を export するコマンドを記述しておく方法もあります。なお /etc/passwd ファイルにて LFS 変数を必要とするユーザーは、シェルとして bash を利用するようにしてください。/root/.bash_profile ファイルはログインプロセスの一部として機能するためです。

もう一つ気にかけることとして、ホストシステム上にログ出力を行う方法に関してです。グラフィカルディスプレイマネージャーを通じてログ出力を行うと、仮想端末が起動する際に、ユーザー独自の .bash_profile は普通は用いられません。この場合は、各ユーザー用と root 用の .bashrc に export コマンドを追加してください。ここでディストリビューションの中には、非対話形式での bash の実行時には .bashrc を実行しないように求めているものがあります。その場合は、非対話形式の利用をテストする前に export コマンドを追加してください。

2.7. 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。本書では前に示したように、環境変数 `LFS` に指定されたディレクトリに対してファイルシステムがマウントされるものとします。

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 ext4 /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 /home 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 ext4 /dev/<xxx> $LFS
mkdir -v $LFS/home
mount -v -t ext4 /dev/<yyy> $LFS/home
```

<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` 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` 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。



警告

上で説明した内容は、LFS 構築作業においてコンピューターを再起動しない場合の話です。コンピューターを一度シャットダウンした場合は、LFS 構築作業の再開のたびに LFS パーティションを再マウントする必要があります。あるいはブート時に自動マウントをしたいのであれば、ホストシステムの `/etc/fstab` ファイルを書き換えておく必要があります。書き換えは例えば以下ようになります。

```
/dev/<xxx> /mnt/lfs ext4 defaults 1 1
```

追加のパーティションを利用している場合は、それらを書き加えることも忘れないでください。

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。LFS errata やセキュリティアドバイザリーに示されていれば別ですが、ここに示すバージョンとは異なる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、違うバージョンで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。

パッケージによっては、リリース tarball に加えて (Git や SVN の) リポジトリスナップショット tarball があって、両者を同じファイル名で提供している場合があります。リリース tarball には、該当するリポジトリスナップショットの内容に加えて生成済みファイル (たとえば autoconf が作り出す configure スクリプト) が含まれます。本書では可能な限りリリース tarball を用いることにします。本書が指定するリリース tarball ではなく、リポジトリスナップショットを利用すると、問題が発生するかもしれません。

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。たいいていのパッケージを見つけて出すことが出来るはずですが、それでも見つけられなかったら <https://www.linuxfromscratch.org/lfs/packages.html#packages> から入手してください。

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
```

LFS のビルドに必要なパッケージやパッチを得る方法は、いろいろとあります。

- 各ファイルは次の 2 節に示されているので、個々に入手することができます。
- 本書の安定版であれば、それに対して必要となるファイルを集めた tarball が、<https://www.linuxfromscratch.org/mirrors.html#files> に示す LFS ミラーサイトからダウンロードできます。
- `wget` と以下に示す `wget-list` ファイルを利用すれば、すべてのファイルをダウンロードすることができます。

パッケージとパッチのダウンロードを行うため `wget-list` を利用することにします。これは以下のように `wget` コマンドの入力引数に指定します。

```
wget --input-file=wget-list-sysv --continue --directory-prefix=$LFS/sources
```



日本語訳情報

オリジナルの LFS ブックでは、wget-list 内に含まれる、各種パッケージの入手 URL が主に米国サイトとなっています。一方、日本国内にて作業する方であれば、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。また同じ方法で Linux カーネル、Perl、Vim の入手先も変更します。

```
cat > wl.sed << "EOF"
s|ftp\.gnu\.org/gnu/|ftp.riken.jp/GNU/|g
s|https://www\.kernel\.org/pub/linux/|http://ftp.riken.jp/Linux/kernel.org/linux/|g
s|www\.cpan\.org|ftp.riken.jp/lang/CPAN|g
s|ftp\.vim\.org|ftp.jp.vim.org|g
EOF
sed -f wl.sed -i.orig wget-list-sysv
rm wl.sed
```

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。サイトを変えた場合は、パッケージの URL が異なることが多々あるため、適宜 sed 置換内容を書き換えてください。

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。その場合にはパッケージ取得に失敗してしまいます。そこで wget-list と wget-list.orig を順に利用し、かつ wget コマンドにて -N オプションを使って（取得済のものはスキップするようにして）以下のコマンドを実行すれば、確実にすべてのパッケージを入手することができます。

```
wget -N --input-file=wget-list-sysv --continue --directory-prefix=$LFS/sources
wget -N --input-file=wget-list-sysv.orig --continue --directory-prefix=$LFS/sources
```

さらに LFS-7.0 からは md5sums というファイルを用意しています。このファイルは、入手した各種パッケージのファイルが正しいことを確認するために用いることができます。このファイルを \$LFS/sources に配置して以下を実行してください。

```
pushd $LFS/sources
md5sum -c md5sums
popd
```

必要なファイルを入手した方法が前述のどの方法であっても、この md5sum チェックを実施することができます。

3.2. 全パッケージ



注記

パッケージをダウンロードする前には セキュリティアドバイス (security advisories) を読んでください。セキュリティぜい弱性を回避するためにパッケージの最新バージョンがないかどうかを確認してください。

アップストリームでは、古いリリースを削除していることがあります。特にそのリリースにセキュリティぜい弱性を含んでいた場合です。以下に示す URL が無効になっていたら、まず初めにセキュリティアドバイスを読んでください。そして新たなバージョンが（ぜい弱性を解消して）入手できるかどうかを確認してください。それでもパッケージが削除されてしまっている場合は、ミラーサイトからのダウンロードを試してみてください。ぜい弱性が原因で削除されていた古いバージョンのパッケージがダウンロードできたとしても、ぜい弱性に問題があるのであれば、システムビルドに用いることはお勧めしません。

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。

- Acl (2.3.1) - 348 KB:
 ホームページ: <https://savannah.nongnu.org/projects/acl>
 ダウンロード: <https://download.savannah.gnu.org/releases/acl/acl-2.3.1.tar.xz>
 MD5 sum: 95ce715fe09acca7c12d3306d0f076b2
- Attr (2.5.1) - 456 KB:
 ホームページ: <https://savannah.nongnu.org/projects/attr>
 ダウンロード: <https://download.savannah.gnu.org/releases/attr/attr-2.5.1.tar.gz>
 MD5 sum: ac1c5a7a084f0f83b8cace34211f64d8
- Autoconf (2.71) - 1,263 KB:
 ホームページ: <https://www.gnu.org/software/autoconf/>
 ダウンロード: <https://ftp.gnu.org/gnu/autoconf/autoconf-2.71.tar.xz>
 MD5 sum: 12cfa1687ffa2606337efe1a64416106
- Automake (1.16.5) - 1,565 KB:
 ホームページ: <https://www.gnu.org/software/automake/>
 ダウンロード: <https://ftp.gnu.org/gnu/automake/automake-1.16.5.tar.xz>
 MD5 sum: 4017e96f89fca45ca946f1c5db6be714
 SHA256 sum: 80facc09885a57e6d49d06972c0a0e1089c5fa8f4d4c7cfe5baea58e5085f136d
- Bash (5.1.16) - 10,277 KB:
 ホームページ: <https://www.gnu.org/software/bash/>
 ダウンロード: <https://ftp.gnu.org/gnu/bash/bash-5.1.16.tar.gz>
 MD5 sum: c17b20a09fc38d67fb303aeb6c130b4e

- Bc (6.0.1) - 441 KB:
 ホームページ: <https://git.yzena.com/gavin/bc>
 ダウンロード: <https://github.com/gavinhoward/bc/releases/download/6.0.1/bc-6.0.1.tar.xz>
 MD5 sum: 4c8b8d51eb52ee66f5bcf6a6a1ca576e
- Binutils (2.39) - 24,578 KB:
 ホームページ: <https://www.gnu.org/software/binutils/>
 ダウンロード: <https://ftp.gnu.org/gnu/binutils/binutils-2.39.tar.xz>
 MD5 sum: f7e986ae9ff06405cafb2e585ee36d27
- Bison (3.8.2) - 2,752 KB:
 ホームページ: <https://www.gnu.org/software/bison/>
 ダウンロード: <https://ftp.gnu.org/gnu/bison/bison-3.8.2.tar.xz>
 MD5 sum: c28f119f405a2304ff0a7ccdcc629713
- Bzip2 (1.0.8) - 792 KB:
 ダウンロード: <https://www.sourceware.org/pub/bzip2/bzip2-1.0.8.tar.gz>
 MD5 sum: 67e051268d0c475ea773822f7500d0e5
- Check (0.15.2) - 760 KB:
 ホームページ: <https://libcheck.github.io/check>
 ダウンロード: <https://github.com/libcheck/check/releases/download/0.15.2/check-0.15.2.tar.gz>
 MD5 sum: 50fcacfcecd5a380415b12e9c574e0b2
- Coreutils (9.1) - 5,570 KB:
 ホームページ: <https://www.gnu.org/software/coreutils/>
 ダウンロード: <https://ftp.gnu.org/gnu/coreutils/coreutils-9.1.tar.xz>
 MD5 sum: 8b1ca4e018a7dce9bb937faec6618671
- DejaGNU (1.6.3) - 608 KB:
 ホームページ: <https://www.gnu.org/software/dejagnu/>
 ダウンロード: <https://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.3.tar.gz>
 MD5 sum: 68c5208c58236eba447d7d6d1326b821
- Diffutils (3.8) - 1,548 KB:
 ホームページ: <https://www.gnu.org/software/diffutils/>
 ダウンロード: <https://ftp.gnu.org/gnu/diffutils/diffutils-3.8.tar.xz>
 MD5 sum: 6a6b0fdc72acfe3f2829aab477876fbc
- E2fsprogs (1.46.5) - 9,307 KB:
 ホームページ: <http://e2fsprogs.sourceforge.net/>
 ダウンロード: <https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.46.5/e2fsprogs-1.46.5.tar.gz>
 MD5 sum: 3da91854c960ad8a819b48b2a404eb43
- Elfutils (0.187) - 9,024 KB:
 ホームページ: <https://sourceware.org/elfutils/>
 ダウンロード: <https://sourceware.org/ftp/elfutils/0.187/elfutils-0.187.tar.bz2>
 MD5 sum: cc04f07b53a71616b22553c0a458cf4b
- Eudev (3.2.11) - 2,075 KB:
 ダウンロード: <https://github.com/eudev-project/eudev/releases/download/v3.2.11/eudev-3.2.11.tar.gz>
 MD5 sum: 417ba948335736d4d81874fba47a30f7
- Expat (2.4.8) - 444 KB:
 ホームページ: <https://libexpat.github.io/>
 ダウンロード: <https://prdownloads.sourceforge.net/expat/expat-2.4.8.tar.xz>
 MD5 sum: 0584a7318a4c007f7ec94778799d72fe
- Expect (5.45.4) - 618 KB:
 ホームページ: <https://core.tcl.tk/expect/>
 ダウンロード: <https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz>
 MD5 sum: 00fce8de158422f5ccd2666512329bd2

- File (5.42) - 1,080 KB:
 ホームページ: <https://www.darwinsys.com/file/>
 ダウンロード: <https://astron.com/pub/file/file-5.42.tar.gz>
 MD5 sum: 4d4f70c3b08a8a70d8baf67f085d7e92
- Findutils (4.9.0) - 1,999 KB:
 ホームページ: <https://www.gnu.org/software/findutils/>
 ダウンロード: <https://ftp.gnu.org/gnu/findutils/findutils-4.9.0.tar.xz>
 MD5 sum: 4a4a547e888a944b2f3af31d789a1137
- Flex (2.6.4) - 1,386 KB:
 ホームページ: <https://github.com/westes/flex>
 ダウンロード: <https://github.com/westes/flex/releases/download/v2.6.4/flex-2.6.4.tar.gz>
 MD5 sum: 2882e3179748cc9f9c23ec593d6adc8d
- Gawk (5.1.1) - 3,075 KB:
 ホームページ: <https://www.gnu.org/software/gawk/>
 ダウンロード: <https://ftp.gnu.org/gnu/gawk/gawk-5.1.1.tar.xz>
 MD5 sum: 83650aa943ff2fd519b2abedf8506ace
- GCC (12.2.0) - 82,662 KB:
 ホームページ: <https://gcc.gnu.org/>
 ダウンロード: <https://ftp.gnu.org/gnu/gcc/gcc-12.2.0/gcc-12.2.0.tar.xz>
 MD5 sum: 73bafd0af874439dcdb9fc063b6fb069
 SHA256 sum:
- GDBM (1.23) - 1,092 KB:
 ホームページ: <https://www.gnu.org/software/gdbm/>
 ダウンロード: <https://ftp.gnu.org/gnu/gdbm/gdbm-1.23.tar.gz>
 MD5 sum: 8551961e36bf8c70b7500d255d3658ec
- Gettext (0.21) - 9,487 KB:
 ホームページ: <https://www.gnu.org/software/gettext/>
 ダウンロード: <https://ftp.gnu.org/gnu/gettext/gettext-0.21.tar.xz>
 MD5 sum: 40996bbaf7d1356d3c22e33a8b255b31
- Glibc (2.36) - 18,175 KB:
 ホームページ: <https://www.gnu.org/software/libc/>
 ダウンロード: <https://ftp.gnu.org/gnu/glibc/glibc-2.36.tar.xz>
 MD5 sum: 00e9b89e043340f688bc93ec03239b57
- GMP (6.2.1) - 1,980 KB:
 ホームページ: <https://www.gnu.org/software/gmp/>
 ダウンロード: <https://ftp.gnu.org/gnu/gmp/gmp-6.2.1.tar.xz>
 MD5 sum: 0b82665c4a92fd2ade7440c13fcaa42b
- Gperf (3.1) - 1,188 KB:
 ホームページ: <https://www.gnu.org/software/gperf/>
 ダウンロード: <https://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz>
 MD5 sum: 9e251c0a618ad0824b51117d5d9db87e
- Grep (3.7) - 1,603 KB:
 ホームページ: <https://www.gnu.org/software/grep/>
 ダウンロード: <https://ftp.gnu.org/gnu/grep/grep-3.7.tar.xz>
 MD5 sum: 7c9cca97fa18670a21e72638c3e1dabf
- Groff (1.22.4) - 4,044 KB:
 ホームページ: <https://www.gnu.org/software/groff/>
 ダウンロード: <https://ftp.gnu.org/gnu/groff/groff-1.22.4.tar.gz>
 MD5 sum: 08fb04335e2f5e73f23ea4c3adbf0c5f
- GRUB (2.06) - 6,428 KB:
 ホームページ: <https://www.gnu.org/software/grub/>
 ダウンロード: <https://ftp.gnu.org/gnu/grub/grub-2.06.tar.xz>
 MD5 sum: cf0fd928b1e5479c8108ee52cb114363

- Gzip (1.12) - 807 KB:
 ホームページ: <https://www.gnu.org/software/gzip/>
 ダウンロード: <https://ftp.gnu.org/gnu/gzip/gzip-1.12.tar.xz>
 MD5 sum: 9608e4ac5f061b2a6479dc44e917a5db
- Iana-Etc (20220812) - 584 KB:
 ホームページ: <https://www.iana.org/protocols>
 ダウンロード: <https://github.com/Mic92/iana-etc/releases/download/20220812/iana-etc-20220812.tar.gz>
 MD5 sum: 851a53efd53c77d0ad7b3d2b68d8a3fc
- Inetutils (2.3) - 1,518 KB:
 ホームページ: <https://www.gnu.org/software/inetutils/>
 ダウンロード: <https://ftp.gnu.org/gnu/inetutils/inetutils-2.3.tar.xz>
 MD5 sum: e73e2ed42d73ceb47616b20131236036
 SHA256 sum:
- Intltool (0.51.0) - 159 KB:
 ホームページ: <https://freedesktop.org/wiki/Software/intltool>
 ダウンロード: <https://launchpad.net/intltool/trunk/0.51.0/+download/intltool-0.51.0.tar.gz>
 MD5 sum: 12e517cac2b57a0121cda351570f1e63
- IPRoute2 (5.19.0) - 872 KB:
 ホームページ: <https://www.kernel.org/pub/linux/utils/net/iproute2/>
 ダウンロード: <https://www.kernel.org/pub/linux/utils/net/iproute2/iproute2-5.19.0.tar.xz>
 MD5 sum: 415bd9eeb8515a585e245809d2fe45a6
- Kbd (2.5.1) - 1,457 KB:
 ホームページ: <https://kbd-project.org/>
 ダウンロード: <https://www.kernel.org/pub/linux/utils/kbd/kbd-2.5.1.tar.xz>
 MD5 sum: 10f10c0a9d897807733f2e2419814abb
- Kmod (30) - 555 KB:
 ダウンロード: <https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-30.tar.xz>
 MD5 sum: 85202f0740a75eb52f2163c776f9b564
- Less (590) - 348 KB:
 ホームページ: <https://www.greenwoodsoftware.com/less/>
 ダウンロード: <https://www.greenwoodsoftware.com/less/less-590.tar.gz>
 MD5 sum: f029087448357812fba450091a1172ab
- LFS-Bootscripts (20220723) - 33 KB:
 ダウンロード: <http://lfsbookja.osdn.jp/git-ja/lfs-bootscripts-20220723.tar.xz>
 MD5 sum: 74884d0d91616f843599c99a333666da
- Libcap (2.65) - 176 KB:
 ホームページ: <https://sites.google.com/site/fullycapable/>
 ダウンロード: <https://www.kernel.org/pub/linux/libs/security/linux-privs/libcap2/libcap-2.65.tar.xz>
 MD5 sum: 3543e753dd941255c4def6cc67a462bb
- Libffi (3.4.2) - 1,320 KB:
 ホームページ: <https://sourceware.org/libffi/>
 ダウンロード: <https://github.com/libffi/libffi/releases/download/v3.4.2/libffi-3.4.2.tar.gz>
 MD5 sum: 294b921e6cf9ab0fbaea4b639f8fdb8e8
- Libpipeline (1.5.6) - 954 KB:
 ホームページ: <http://libpipeline.nongnu.org/>
 ダウンロード: <https://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.6.tar.gz>
 MD5 sum: 829c9ba46382b0b3e12dd11fcbclbb27
- Libtool (2.4.7) - 996 KB:
 ホームページ: <https://www.gnu.org/software/libtool/>
 ダウンロード: <https://ftp.gnu.org/gnu/libtool/libtool-2.4.7.tar.xz>
 MD5 sum: 2fc0b6ddcd66a89ed6e45db28fa44232

- Linux (5.19.2) - 128,553 KB:
 ホームページ: <https://www.kernel.org/>
 ダウンロード: <https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.19.2.tar.xz>
 MD5 sum: 391274e2e49a881403b0ff2e0712bf82



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、入手可能な最新安定版のカーネルを用いてください。あるいは errata に指示があればそれに従ってください。

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 (1.4.19) - 1,617 KB:
 ホームページ: <https://www.gnu.org/software/m4/>
 ダウンロード: <https://ftp.gnu.org/gnu/m4/m4-1.4.19.tar.xz>
 MD5 sum: 0d90823e1426f1da2fd872df0311298d
- Make (4.3) - 2,263 KB:
 ホームページ: <https://www.gnu.org/software/make/>
 ダウンロード: <https://ftp.gnu.org/gnu/make/make-4.3.tar.gz>
 MD5 sum: fc7a67ea86ace13195b0bce683fd4469
- Man-DB (2.10.2) - 1,860 KB:
 ホームページ: <https://www.nongnu.org/man-db/>
 ダウンロード: <https://download.savannah.gnu.org/releases/man-db/man-db-2.10.2.tar.xz>
 MD5 sum: e327f7af3786d15e5851658ae7ef47ed
- Man-pages (5.13) - 1,752 KB:
 ホームページ: <https://www.kernel.org/doc/man-pages/>
 ダウンロード: <https://www.kernel.org/pub/linux/docs/man-pages/man-pages-5.13.tar.xz>
 MD5 sum: 3ac24e8c6fae26b801cb87ceb63c0a30
- Meson (0.63.1) - 2,016 KB:
 ホームページ: <https://mesonbuild.com>
 ダウンロード: <https://github.com/mesonbuild/meson/releases/download/0.63.1/meson-0.63.1.tar.gz>
 MD5 sum: 078e59d11a72b74c3bd78cb8205e9ed7
- MPC (1.2.1) - 820 KB:
 ホームページ: <http://www.multiprecision.org/>
 ダウンロード: <https://ftp.gnu.org/gnu/mpc/mpc-1.2.1.tar.gz>
 MD5 sum: 9f16c976c25bb0f76b50be749cd7a3a8
- MPFR (4.1.0) - 1,490 KB:
 ホームページ: <https://www.mpfr.org/>
 ダウンロード: <https://ftp.gnu.org/gnu/mpfr/mpfr-4.1.0.tar.xz>
 MD5 sum: bdd3d5efba9c17da8d83a35ec552baef
- Ncurses (6.3) - 3,500 KB:
 ホームページ: <https://www.gnu.org/software/ncurses/>
 ダウンロード: <https://invisible-mirror.net/archives/ncurses/ncurses-6.3.tar.gz>
 MD5 sum: a2736befde5fee7d2b7eb45eb281cdbe
- Ninja (1.11.0) - 228 KB:
 ホームページ: <https://ninja-build.org/>
 ダウンロード: <https://github.com/ninja-build/ninja/archive/v1.11.0/ninja-1.11.0.tar.gz>
 MD5 sum: 7d1a1a2f5cdc06795b3054df5c17d5ef
- OpenSSL (3.0.5) - 14,722 KB:
 ホームページ: <https://www.openssl.org/>
 ダウンロード: <https://www.openssl.org/source/openssl-3.0.5.tar.gz>
 MD5 sum: 163bb3e58c143793d1dc6a6ec7d185d5

- Patch (2.7.6) - 766 KB:
 ホームページ: <https://savannah.gnu.org/projects/patch/>
 ダウンロード: <https://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz>
 MD5 sum: 78ad9937e4caadcba1526ef1853730d5
- Perl (5.36.0) - 12,746 KB:
 ホームページ: <https://www.perl.org/>
 ダウンロード: <https://www.cpan.org/src/5.0/perl-5.36.0.tar.xz>
 MD5 sum: 826e42da130011699172fd655e49cfa2
- Pkg-config (0.29.2) - 1,970 KB:
 ホームページ: <https://www.freedesktop.org/wiki/Software/pkg-config>
 ダウンロード: <https://pkg-config.freedesktop.org/releases/pkg-config-0.29.2.tar.gz>
 MD5 sum: f6e931e319531b736fadc017f470e68a
- Procps (4.0.0) - 979 KB:
 ホームページ: <https://sourceforge.net/projects/procps-ng>
 ダウンロード: <https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-4.0.0.tar.xz>
 MD5 sum: eedf93f2f6083afb7abf72188018e1e5
- Psmisc (23.5) - 395 KB:
 ホームページ: <https://gitlab.com/psmisc/psmisc>
 ダウンロード: <https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.5.tar.xz>
 MD5 sum: 014f0b5d5ab32478a2c57812ad01e1fb
- Python (3.10.6) - 19,142 KB:
 ホームページ: <https://www.python.org/>
 ダウンロード: <https://www.python.org/ftp/python/3.10.6/Python-3.10.6.tar.xz>
 MD5 sum: afc7e14f7118d10d1ba95ae8e2134bf0
- Python Documentation (3.10.6) - 7,321 KB:
 ダウンロード: <https://www.python.org/ftp/python/doc/3.10.6/python-3.10.6-docs-html.tar.bz2>
 MD5 sum: 8f32c4f4f0b18ec56e8b3822bbaeb017
- Readline (8.1.2) - 2,923 KB:
 ホームページ: <https://tiswww.case.edu/php/chet/readline/rltop.html>
 ダウンロード: <https://ftp.gnu.org/gnu/readline/readline-8.1.2.tar.gz>
 MD5 sum: 12819fa739a78a6172400f399ab34f81
- Sed (4.8) - 1,317 KB:
 ホームページ: <https://www.gnu.org/software/sed/>
 ダウンロード: <https://ftp.gnu.org/gnu/sed/sed-4.8.tar.xz>
 MD5 sum: 6d906edfdb3202304059233f51f9a71d
- Shadow (4.12.2) - 1,706 KB:
 ホームページ: <https://shadow-maint.github.io/shadow/>
 ダウンロード: <https://github.com/shadow-maint/shadow/releases/download/4.12.2/shadow-4.12.2.tar.xz>
 MD5 sum: 52637cb34c357acf85c617cf95da34a6
- Sysklogd (1.5.1) - 88 KB:
 ホームページ: <https://www.infodrom.org/projects/sysklogd/>
 ダウンロード: <https://www.infodrom.org/projects/sysklogd/download/sysklogd-1.5.1.tar.gz>
 MD5 sum: c70599ab0d037fde724f7210c2c8d7f8
- Sysvinit (3.04) - 216 KB:
 ホームページ: <https://savannah.nongnu.org/projects/sysvinit>
 ダウンロード: <https://download.savannah.gnu.org/releases/sysvinit/sysvinit-3.04.tar.xz>
 MD5 sum: 9a00e5f15dd2f038f10feee50677ebff
- Tar (1.34) - 2,174 KB:
 ホームページ: <https://www.gnu.org/software/tar/>
 ダウンロード: <https://ftp.gnu.org/gnu/tar/tar-1.34.tar.xz>
 MD5 sum: 9a08d29a9ac4727130b5708347c0f5cf
- Tcl (8.6.12) - 10,112 KB:
 ホームページ: <http://tcl.sourceforge.net/>
 ダウンロード: <https://downloads.sourceforge.net/tcl/tcl8.6.12-src.tar.gz>
 MD5 sum: 87ea890821d2221f2ab5157bc5eb885f

- Tcl Documentation (8.6.12) - 1,176 KB:
ダウンロード: <https://downloads.sourceforge.net/tcl/tcl8.6.12-html.tar.gz>
MD5 sum: a0d1a5b60bbb68f2f0bd3066a19c527a
- Texinfo (6.8) - 4,848 KB:
ホームページ: <https://www.gnu.org/software/texinfo/>
ダウンロード: <https://ftp.gnu.org/gnu/texinfo/texinfo-6.8.tar.xz>
MD5 sum: a91b404e30561a5df803e6eb3a53be71
- Time Zone Data (2022c) - 423 KB:
ホームページ: <https://www.iana.org/time-zones>
ダウンロード: <https://www.iana.org/time-zones/repository/releases/tzdata2022c.tar.gz>
MD5 sum: 4e3b2369b68e713ba5d3f7456f20bfdb
- Udev-lfs Tarball (udev-lfs-20171102) - 11 KB:
ダウンロード: <https://anduin.linuxfromscratch.org/LFS/udev-lfs-20171102.tar.xz>
MD5 sum: 27cd82f9a61422e186b9d6759ddf1634
- Util-linux (2.38.1) - 7,321 KB:
ホームページ: <https://git.kernel.org/pub/scm/utils/util-linux/util-linux.git/>
ダウンロード: <https://www.kernel.org/pub/linux/utils/util-linux/v2.38/util-linux-2.38.1.tar.xz>
MD5 sum: cd11456f4ddd31f7fbfdd9488c0c0d02
- Vim (9.0.0228) - 16,372 KB:
ホームページ: <https://www.vim.org>
ダウンロード: <https://anduin.linuxfromscratch.org/LFS/vim-9.0.0228.tar.gz>
MD5 sum: bc7e0a4829d94bb4c03a7a6b4ad6a8cf



注記

vim のバージョンは日々更新されます。最新版を入手するには <https://github.com/vim/vim/tags> にアクセスしてください。

- Wheel (0.37.1) - 65 KB:
ホームページ: <https://pypi.org/project/wheel/>
ダウンロード: <https://anduin.linuxfromscratch.org/LFS/wheel-0.37.1.tar.gz>
MD5 sum: f490f1399e5903706cb1d4fbed9ecb28
- XML::Parser (2.46) - 249 KB:
ホームページ: <https://github.com/chorny/XML-Parser>
ダウンロード: <https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.46.tar.gz>
MD5 sum: 80bb18a8e6240fcf7ec2f7b57601c170
- Xz Utils (5.2.6) - 1,234 KB:
ホームページ: <https://tukaani.org/xz>
ダウンロード: <https://tukaani.org/xz/xz-5.2.6.tar.xz>
MD5 sum: d9cd5698e1ec06cf638c0d2d645e8175
- Zlib (1.2.12) - 1259 KB:
ホームページ: <https://www.zlib.net/>
ダウンロード: <https://zlib.net/zlib-1.2.12.tar.xz>
MD5 sum: 28687d676c04e7103bb6ff2b9694c471
- Zstd (1.5.2) - 1,892 KB:
ホームページ: <https://facebook.github.io/zstd/>
ダウンロード: <https://github.com/facebook/zstd/releases/download/v1.5.2/zstd-1.5.2.tar.gz>
MD5 sum: 072b10f71f5820c24761a65f31f43e73

全パッケージのサイズ合計: 約 461 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なパッチすべてです。



日本語訳情報

各パッチに付けられている簡略な名称については、訳出せずそのまま表記することにします。

- Bzip2 Documentation Patch - 1.6 KB:
ダウンロード: https://www.linuxfromscratch.org/patches/lfs/11.2/bzip2-1.0.8-install_docs-1.patch
MD5 sum: 6a5ac7e89b791aae556de0f745916f7f
- Coreutils Internationalization Fixes Patch - 166 KB:
ダウンロード: <https://www.linuxfromscratch.org/patches/lfs/11.2/coreutils-9.1-il8n-1.patch>
MD5 sum: c1ac7edf095027460716577633da9fc5
- Glibc FHS Patch - 2.8 KB:
ダウンロード: <https://www.linuxfromscratch.org/patches/lfs/11.2/glibc-2.36-fhs-1.patch>
MD5 sum: 9a5997c3452909b1769918c759eff8a2
- Kbd Backspace/Delete Fix Patch - 12 KB:
ダウンロード: <https://www.linuxfromscratch.org/patches/lfs/11.2/kbd-2.5.1-backspace-1.patch>
MD5 sum: f75cca16a38da6caa7d52151f7136895
- Sysvinit Consolidated Patch - 2.4 KB:
ダウンロード: <https://www.linuxfromscratch.org/patches/lfs/11.2/sysvinit-3.04-consolidated-1.patch>
MD5 sum: 4900322141d493e74020c9cf437b2cdc
- Zstd Upstream Fixes Patch - 4 KB:
ダウンロード: https://www.linuxfromscratch.org/patches/lfs/11.2/zstd-1.5.2-upstream_fixes-1.patch
MD5 sum: a7e576e3f87415fdf388392b257cdcf3

全パッチの合計サイズ: 約 188.8 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。 <https://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. はじめに

本章では一時システムをビルドするために、あともう少し作業を行います。\$LFS ディレクトリ内に、一連のインストールディレクトリを作ります。リスク軽減のために一般ユーザーを生成し、このユーザーにおけるビルド環境を作ります。また LFS パッケージ類の構築時間を測る手段として標準時間「SBU」について説明し、各パッケージのテストスイートについて触れます。

4.2. LFS ファイルシステムの限定的なディレクトリレイアウトの生成

LFS パーティションに対して行う最初の作業は、限定的なディレクトリ階層を作り出すことです。第 6 章（また glibc や libstdc++ においては第 5 章）においてビルドするプログラムを、最終的なディレクトリにインストールするためです。第 8 章にある一時的なプログラムを、再構築して上書きしていくために必要となります。

必要となるディレクトリレイアウトを生成するため、root ユーザーになって以下を実行します。

```
mkdir -pv $LFS/{etc,var} $LFS/usr/{bin,lib,sbin}

for i in bin lib sbin; do
  ln -sv usr/$i $LFS/$i
done

case $(uname -m) in
  x86_64) mkdir -pv $LFS/lib64 ;;
esac
```

第 6 章にあるプログラムはクロスコンパイラーによってビルドされます。（詳しくは ツールチェーンの技術的情報を参照してください。）クロスコンパイラーは他のプログラムとは切り分けるため、特別なディレクトリにインストールすることにします。ここでそのディレクトリを生成します。

```
mkdir -pv $LFS/tools
```

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可能性があります。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なります。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`

このパラメーターは、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 ディレクトリの所有者を lfs ユーザーとすることで、このディレクトリ配下の全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/{usr{,/*},lib,var,etc,bin,sbin,tools}
case $(uname -m) in
  x86_64) chown -v lfs $LFS/lib64 ;;
esac
```



注記

ホストシステムによっては、以下のコマンドを実行しても正常に処理されず、lfs ユーザーへのログインがバックグラウンドで処理中のままとってしまうことがあります。プロンプトに "lfs:~\$" という表示がすぐに現れなかった場合は、fg コマンドを入力することで解決するかもしれません。

lfs でログインします。これはディスプレイマネージャーを通じて仮想端末を用いることができます。また以下のユーザー変更コマンドを用いるのでも構いません。

```
su - lfs
```

パラメーター「-」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 ファイルの内容を読み込んで実行することはなく、代わりに .bashrc ファイルを読み込んで実行します。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=$(uname -m)-lfs-linux-gnu
PATH=/usr/bin
if [ ! -L /bin ]; then PATH=/bin:$PATH; fi
PATH=$LFS/tools/bin:$PATH
CONFIG_SITE=$LFS/usr/share/config.site
export LFS LC_ALL LFS_TGT PATH CONFIG_SITE
EOF
```

.bashrc 内の設定の意味

set +h
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/bin` 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ホストディストリビューションのディレクトリ、`/usr/bin` または `/bin` 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。

```
umask 022
```

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。)

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

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=POSIX
```

`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。`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=(uname -m)-lfs-linux-gnu
```

`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ーの構築、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 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。

```
PATH=/usr/bin
```

最近の Linux ディストリビューションでは `/bin` と `/usr/bin` をマージしているものが多くあります。その場合、第 6 章 に対しての標準の `PATH` 変数は `/usr/bin/` に設定するだけで十分です。そうでない場合は、パスに対して `/bin` を加える必要があります。

```
if [ ! -L /bin ]; then PATH=/bin:$PATH; fi
/bin がシンボリックリンクではないは PATH 変数に加える必要があります。
```

```
PATH=$LFS/tools/bin:$PATH
```

`$LFS/tools/bin` ディレクトリを `PATH` 変数の先頭に設定します。第 5 章の冒頭においてインストールしたクロスコンパイラは、インストールした直後からシェル上から実行できるようになります。この設定を行うことで、ハッシュ機能をオフにしたことと連携して、ホスト上のコンパイラが利用されないようにします。

```
CONFIG_SITE=$LFS/usr/share/config.site
```

第 5 章 と 第 6 章 においてこの変数を設定しておかないと、ディストリビューションによっては `configure` スクリプトが、ホストシステム上の `/usr/share/config.site` から設定項目を取得してしまうことがあります。ホストの影響が及ばないようにここでオーバーライドします。

```
export ...
```

上のコマンド実行は、設定済の変数を改めて設定するものになりますが、シェルを新たに呼び出しても確実に設定されるようにエクスポートを行うことにします。



重要

商用ディストリビューションの中には、`bash` の初期化を行うスクリプトとして、ドキュメント化されていない `/etc/bash.bashrc` というものを加えているものがあります。このファイルは `lfs` ユーザー環境を修正してしまう可能性があります。それにより `LFS` にとっての重要パッケージのビルドに支障をきたすことがあります。`lfs` ユーザー環境をきれいに保つため、`/etc/bash.bashrc` というファイルが存在しているかどうかを確認してください。そして存在していたらファイルを移動させてください。`root` ユーザーになって以下を実行します。

```
[ ! -e /etc/bash.bashrc ] || mv -v /etc/bash.bashrc /etc/bash.bashrc.NOUSE
```

第 7 章 の冒頭において `lfs` ユーザーの利用を終えたとき (必要であれば) `/etc/bash.bashrc` を元に戻してください。

なお「`Bash-5.1.16`」においてビルドした、`LFS` における `Bash` パッケージは、`/etc/bash.bashrc` をロードしたり読み取ったりするように設定されていません。したがって完璧な `LFS` システムであれば、このファイルは不要なものです。

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はなく、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。



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 Intel i5-6500 CPU であれば、以下のようにして同時並行の 4 つのプロセスを実行することができます。

```
export MAKEFLAGS='-j4'
```

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 -j4
```

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。そればかりか場合により make 処理に失敗することもあります。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

第 5 章 と 第 6 章 においてテストスイートを実行することはできません。各プログラムはクロスコンパイラによってコンパイルされているので、ビルドしているホスト上での実行が対応できないためです。

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 <https://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 <https://www.linuxfromscratch.org/lfs/build-logs/11.2/> 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III部 LFS クロスチェーン と一時的ツールの構築

重要な準備事項

はじめに

この部は 3 つのステージに分かれています。1 つめはクロスコンパイラと関連ライブラリをビルドします。2 つめはそのクロスコンパイラを使って、ホストのパッケージからは切り離された形で、各種ユーティリティをビルドします。そして 3 つめでは chroot 環境に入ることで、さらにホスト環境から離れて、最終システムをビルドするために必要となる残りのツール類をビルドします。



重要

この部から、新システムのビルドに向けた本格的作業を開始します。ここではより注意深く、本書が示す手順どおりに作業を進めていくことが必要です。そこで何を行っているのかを十分に理解するようにしてください。これはビルドを完成させたいという思いとは別の話です。ただ単に書かれている内容を入力だけの作業はやめてください。わかっていないことがあれば、しっかりと本書を読むようにしてください。また入力した内容やコマンドの処理結果は、ファイル出力を行うなどして記録するようにしてください。tee ユーティリティを使うことにすれば、何かおかしなことになっても調べられるようになります。

次の節では、ビルド過程における技術的な情報を示します。それに続く節では、極めて重要な 全般的なコンパイル手順を示しています。

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 と 第 6 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はシステム構築のための十分なツール類を有していて、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 8 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。

ビルド過程は クロスコンパイル を基本として行います。通常クロスコンパイルとは、ビルドを行うマシンとは異なるマシン向けにコンパイラや関連ツールチェーンをビルドすることです。これは厳密には LFS に必要なものではありません。というのも新たに作り出すシステムは、ビルドに使ったマシンと同一環境で動かすことにしているためです。しかしクロスコンパイルには大きな利点があって、クロスコンパイルによってビルドしたものは、ホスト環境上にはまったく依存できないものとなります。

クロスコンパイルについて



注記

LFS はクロスツールチェーン（あるいはネイティブツールチェーン）のビルドを説明する書ではなく、その説明は行っていません。クロスツールチェーンは、LFS のビルドとは異なる別の目的で用いるものであるため、何を行っているのかが十分に分からないまま、クロスチェーン向けのコマンドを利用することは避けてください。

クロスコンパイルには必要な捉え方があって、それだけで 1 つの節を当てて説明するだけの価値があるものです。初めて読む方は、この節を読み飛ばしてかまいません。ただしビルド過程を十分に理解するためには、後々この節に戻ってきて読んで頂くことをお勧めします。

ここにおいて取り上げる用語を定義しておきます。

ビルド (build)

ビルド作業を行うマシンのこと。他の節においてこのマシンは "ホスト (host)" と呼ぶこともあります。

ホスト (host)

ビルドされたプログラムを実行するマシンまたはシステムのこと。ここでいう "ホスト" とは、他の節でいうものとは同一ではありません。

ターゲット (target)

コンパイラにおいてのみ用いられます。コンパイラの生成コードを必要とするマシンのこと。これはビルドやホストとは異なることもあります。

例として以下のシナリオを考えてみます。(これはよく "カナディアンクロス (Canadian Cross)" とも呼ばれるものです。) コンパイラが低速なマシン上にだけあるとします。これをマシン A と呼び、コンパイラは ccA とします。これとは別に高速なマシン (マシン B) があって、ただしそこにはコンパイラがありません。そしてここから作り出すプログラムコードは、まったく別の低速マシン (マシン C) 向けであるとします。マシン C 向けにコンパイラをビルドするためには、以下の 3 つの段階を経ることになります。

段階	ビルド	ホスト	ターゲット	作業
1	A	A	B	マシン A 上の ccA を使い、クロスコンパイラ cc1 をビルド。
2	A	B	C	マシン A 上の cc1 を使い、クロスコンパイラ cc2 をビルド。
3	B	C	C	マシン B 上の cc2 を使い、コンパイラ ccC をビルド。

マシン C 上で必要となる他のプログラムは、高速なマシン B 上において cc2 を用いてコンパイルすることができます。マシン B がマシン C 向けのプログラムを実行できなかったとすると、マシン C そのものが動作するようにならない限り、プログラムのビルドやテストは一切できないこととなります。たとえば ccC をテストするには、以下の 4 つめの段階が必要になります。

段階	ビルド	ホスト	ターゲット	作業
4	C	C	C	マシン C 上にて ccC を使い ccC そのものの再ビルドとテストを実施。

上の例において cc1 と cc2 だけがクロスコンパイラです。つまりこのコンパイラは、これを実行しているマシンとは別のマシンに対するコードを生成できるものです。これに比べて ccA と ccC というコンパイラは、実行しているマシンと同一マシン向けのコードしか生成できません。そういうコンパイラのことをネイティブ コンパイラと呼びます。

LFS におけるクロスコンパイラーの実装方法



注記

ほぼすべてのビルドシステムにおいては、`cpu-vendor-kernel-os` という形式のマシントリプレット (triplet) と呼ばれる名称が用いられます。お気づきのことと思いますが、なぜ「トリプレット」といいながら 4 つの項目からなる名前なのでしょう。その理由はこれまでの経緯にあります。当初は 3 つの項目による名前を使っていれば、マシンを間違いなく特定できるものでした。しかし新たなマシン、新たなシステムが登場するようになって、これでは不十分であることがわかりました。「トリプレット」という語だけが残ったわけです。マシンのトリプレットを確認する一番簡単な方法は、`config.guess` スクリプトを実行することです。これは多くのパッケージのソースに含まれています。binutils のソースを伸張 (解凍) し、この `./config.guess` スクリプトを実行して、その出力を確認してください。たとえば 32 ビットのインテルプロセッサであれば、`i686-pc-linux-gnu` と出力されます。64 ビットシステムであれば `x86_64-pc-linux-gnu` となります。

またプラットフォームのダイナミックリンカーの名前にも注意してください。これはダイナミックローダーとも呼ばれます。(binutils の一部である標準リンカー `ld` とは別ものですから混同しないでください。) ダイナミックリンカーは Glibc によって提供されているもので、何かのプログラムが必要とする共有ライブラリを検索しロードします。そして実行できるような準備を行って、実際に実行します。32 ビットインテルマシンに対するダイナミックリンカーの名前は `ld-linux.so.2` となります。(64 ビットシステムであれば `ld-linux-x86-64.so.2` となります。) ダイナミックリンカーの名前を確実に決定するには、何でもよいのでホスト上の実行モジュールを調べます。`readelf -l <name of binary> | grep interpreter` というコマンドを実行することです。出力結果を見てください。どのようなプラットフォームであっても確実な方法は、`shlib-versions` というファイルを見てみることです。これは Glibc ソースツリーのルートに存在しています。

LFS ではクロスコンパイルに似せた作業を行うため、ホストのトリプレットを多少調整します。LFS_TGT 変数において "vendor" 項目を変更します。またクロスリンカーやクロスコンパイラーを生成する際には `--with-sysroot` オプションを利用します。これはホスト内に必要となるファイルがどこにあるかを指示するものです。第 6 章においてビルドされる他のプログラムが、ビルドマシンのライブラリにリンクできないようにするためです。以下の 2 段階は必須ですが、最後の 1 つはテスト用です。

段階	ビルド	ホスト	ターゲット	作業
1	pc	pc	lfs	pc 上の <code>cc-pc</code> を使い、クロスコンパイラー <code>cc1</code> をビルド。
2	pc	lfs	lfs	pc 上の <code>cc1</code> を使い、クロスコンパイラー <code>cc-lfs</code> をビルド。
3	lfs	lfs	lfs	lfs 上の <code>cc-lfs</code> を使い <code>cc-lfs</code> そのものの再ビルドとテストを実施。

上の表において "pc 上の" というのは、すでにそのディストリビューションにおいてインストールされているコマンドを実行することを意味します。また "lfs 上の" とは、chroot 環境下にてコマンドを実行することを意味します。

さてクロスコンパイルに関しては、まだまだあります。C 言語という単にコンパイラがあるだけでなく、標準ライブラリも定義しています。本書では glibc と呼ぶ GNU C ライブラリを用いています。このライブラリは lfs マシン向けにコンパイルされたものでなければなりません。つまりクロスコンパイラ cc1 を使うということです。しかしコンパイラには内部ライブラリというものがあって、アセンブラ命令セットだけでは利用できない複雑な命令が含まれます。その内部ライブラリは libgcc と呼ばれ、完全に機能させるには glibc ライブラリにリンクさせなければなりません。さらに C++ (libstdc++) に対する標準ライブラリも、glibc にリンクさせる必要があります。このようなニトリと卵の問題を解決するには、まず libgcc に基づいた低機能版の cc1 をビルドします。この cc1 にはスレッド処理や例外処理といった機能が含まれていません。その後、この低機能なコンパイラを使って glibc をビルドします。(glibc 自体は低機能ではありません。) そして libstdc++ をビルドします。libstdc++ もやはり、libgcc と同じく機能がいくつか欠如しています。

これで話が終わるわけではありません。上の段落における結論は以下のようになります。cc1 からは完全な libstdc++ はビルドできないということです。しかし第 2 段階においては、C/C++ ライブラリをビルドできる唯一のコンパイラです。もちろん第 2 段階においてビルドされるコンパイラ cc-lfs は、それらライブラリをビルドできます。しかし (1) GCC ビルドシステムは、それが pc 上で利用できるかどうかわかりません、そして (2) pc 上にてそれを使うと pc 内のライブラリにリンクしてしまうリスクがあります。なぜなら cc-lfs はネイティブコンパイラであるからです。そこで libstdc++ は、後々 chroot 環境内でビルドしなければならないのです。

その他の手順詳細

クロスコンパイラは、他から切り離された \$LFS/tools ディレクトリにインストールされます。このクロスコンパイラは、最終システムに含めるものではないからです。

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ーに対するさまざまな機能テストが行われるため、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ーを二箇所にインストールします。\$LFS/tools/bin と \$LFS/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。リンカー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 `$LFS_TGT-ld --verbose | grep SEARCH` を実行すると、検索するライブラリのパスとその検索順を示してくれます。ダミープログラムをコンパイルして ld に `--verbose` オプションをつけてリンクを行うと、どのファイルがリンクされたが分かります。例えば `$LFS_TGT-gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded` と実行すれば、リンカー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... /mnt/lfs/tools/i686-lfs-linux-gnu/bin/as
checking what linker to use... /mnt/lfs/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ーを確認するには `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。

次に健全化された (sanitized) Linux API ヘッダーをインストールします。これにより、標準 C ライブラリ (Glibc) が Linux カーネルが提供する機能とのインターフェースを可能とします。

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、カーネルヘッダーです。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` パラメーターに関連づけたコンパイラを用いるからです。我々の作業においてそのコンパイラとは `$LFS_TGT-gcc` になります。バイナリツールとカーネルヘッダーは多少複雑です。従って無理なことはせずに有効な configure オプションを選択することが必要です。configure 実行の後は build ディレクトリにある `config.make` ファイルに重要な情報が示されているので確認してみてください。なお `CC="$LFS_TGT-gcc"` とすれば、(`$LFS_TGT` が展開されて) 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。

すでに述べたように、標準 C++ ライブラリはこの後でコンパイルします。そして第 6 章では、プログラム自身を必要としているプログラムをすべてビルドしていきます。そのようなパッケージのインストール手順においては `DESTDIR` 変数を使い、LFS ファイルシステム内にインストールします。

第 6 章の最後には、LFS のネイティブコンパイラをインストールします。はじめに `DESTDIR` を使って `binutils` 2 回めをビルドし、他のプログラムにおいてもおなじようにインストールを行います。2 回めとなる GCC ビルドでは、`libstdc++` や不必要なライブラリは省略します。GCC の `configure` スクリプトにはハードコーディングされている部分があるので、`CC_FOR_TARGET` はホストのターゲットが同じであれば `cc` になります。しかしビルドシステムにおいては異なります。そこで `configure` オプションには `CC_FOR_TARGET=$LFS_TGT-gcc` を明示的に指定するようにしています。

第 7 章での `chroot` による環境下では、最初の作業は `libstdc++` をビルドすることです。そして各種プログラムのインストールを、ツールチェーンを適切に操作しながら実施していきます。これ以降、コアとなるツールチェーンは自己完結していきます。そしてシステムの全機能を動作させるための全パッケージの最終バージョンを、ビルドしテストしインストールします。

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。本章と後続の章でパッチを当てるものがあり、あるいは本章と後続の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。
- 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

ビルドにあたっては、ホストシステム要件にて示す要件やシンボリックリンクが、正しくインストールされていることを前提とします。

- `bash` シェルの利用を想定しています。
- `sh` は `bash` へのシンボリックリンクであるものとします。
- `/usr/bin/awk` は `gawk` へのシンボリックリンクであるものとします。
- `/usr/bin/yacc` は `bison` へのシンボリックリンクであるか、あるいは `bison` を実行するためのスクリプトであるものとします。



重要

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。
2. ソースディレクトリに入ります。
3. 各パッケージについて：
 - 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。第 5 章 と 第 6 章 では、パッケージを伸張（解凍）するのは `lfs` ユーザーとします。

パッケージ `tarball` はどこか別ディレクトリにあってもかまいませんが、ソースツリーをどこか別の場所でビルドする方法についてはサポートしていません。特にどこか別に配置しているソースコードを `cp -R` を使ってコピーすると、ソースツリー内のリンクやタイムスタンプを壊しかねません。そうするとビルドの失敗に通じることになります。

- 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。

第5章 クロスツールチェーンの構築

5.1. はじめに

本章ではクロスコンパイラと関連ツールのビルド方法を示します。ここでのクロスコンパイルは見せかけですが、その原理は本当のクロスツールチェーンと同じです。

本章にてビルドされるプログラムは `$LFS/tools` ディレクトリにインストールされます。これはそれ以降にインストールされるファイルとは区別されます。一方でライブラリについては、ビルドしたいシステムに適合するように最終的な場所にインストールします。

5.2. Binutils-2.39 - 1回め

Binutils パッケージは、リンカーやアセンブラー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: 629 MB

5.2.1. クロスコンパイル版 Binutils のインストール



注記

一般的なコンパイル手順 と書かれた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ーやアセンブラーを使って、Glibc や GCC のさまざま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、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 build
cd      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。

Binutils をコンパイルするための準備をします。 :

```
../configure --prefix=$LFS/tools \
              --with-sysroot=$LFS \
              --target=$LFS_TGT  \
              --disable-nls      \
              --enable-gprofng=no \
              --disable-werror
```

`configure` オプションの意味

`--prefix=$LFS/tools`

`configure` スクリプトに対して `binutils` プログラムを `$LFS/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。

`--with-sysroot=$LFS`

クロスコンパイル時に、ターゲットとして必要となるシステムライブラリを `$LFS` より探し出すことを指示します。

`--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ーをビルドするように `configure` スクリプトに指示するものです。

`--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。

`--enable-gprofng=no`

これは `gprofng` のビルドを無効にします。 `gprofng` は一時的ツールにおいては不要であるからです。

`--disable-werror`

ホストのコンパイラーが警告を発生した場合に、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
```

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「Binutils の構成」を参照してください。

5.3. GCC-12.2.0 - 1回め

GCC パッケージは C コンパイラーや C++ コンパイラーなどの GNU コンパイラーコレクションを提供します。

概算ビルド時間: 12 SBU
必要ディスク容量: 3.8 GB

5.3.1. クロスコンパイル版 GCC のインストール

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。個々のパッケージを GCC ソースディレクトリの中に伸張（解凍）し、ディレクトリ名を変更します。これは GCC のビルド処理においてそれらを自動的に利用できるようにするためです。



注記

本節においては誤解が多く発生しています。ここでの手順は他のものと同様であり、手順の概要（パッケージビルド手順）は説明済です。まず初めに gcc の tarball を伸張（解凍）し、生成されたソースディレクトリに移動します。それに加えて本節では、以下の手順を行うものとなります。

```
tar -xf ../mpfr-4.1.0.tar.xz
mv -v mpfr-4.1.0 mpfr
tar -xf ../gmp-6.2.1.tar.xz
mv -v gmp-6.2.1 gmp
tar -xf ../mpc-1.2.1.tar.gz
mv -v mpc-1.2.1 mpc
```

x86_64 ホストにおいて、64 ビットライブラリに対するデフォルトのディレクトリ名は「lib」です。

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
  ;;
esac
```

GCC のドキュメントでは、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 build
cd      build
```

GCC をコンパイルするための準備をします。

```
../configure \
  --target=$LFS_TGT \
  --prefix=$LFS/tools \
  --with-glibc-version=2.36 \
  --with-sysroot=$LFS \
  --with-newlib \
  --without-headers \
  --disable-nls \
  --disable-shared \
  --disable-multilib \
  --disable-decimal-float \
  --disable-threads \
  --disable-libatomic \
  --disable-libgomp \
  --disable-libquadmath \
  --disable-libssp \
  --disable-libvtv \
  --disable-libstdcxx \
  --enable-languages=c,c++
```

configure オプションの意味

`--with-glibc-version=2.36`

このオプションは、ターゲットにおいて用いられることになる glibc のバージョンを指定します。これはホストディストリビューションにある libc のバージョンとは関係がありません。1 回目の gcc によってコンパイルされるものは、すべて chroot 環境内で実行されるものであって、ホストにある libc とは切り離されているためです。

`--with-newlib`

この時点では利用可能な C ライブラリがまだ存在しません。したがって libgcc のビルド時に `inhibit_libc` 定数を定義します。これを行うことで、libc サポートを必要とするコード部分をコンパイルしないようにします。

`--without-headers`

完璧なクロスコンパイラーを構築するなら、GCC はターゲットシステムに互換性を持つ標準ヘッダーを必要とします。本手順においては標準ヘッダーは必要ありません。このスイッチは GCC がそういったヘッダーを探しにいかないようにします。

`--disable-shared`

このスイッチ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。共有ライブラリが glibc を必要としており、処理しているシステム上にはまだインストールされていないためです。

`--disable-multilib`

x86_64 に対して LFS は 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。

`--disable-decimal-float`, `--disable-threads`, `--disable-libatomic`, `--disable-libgomp`, `--disable-libquadmath`, `--disable-libssp`, `--disable-libvtv`, `--disable-libstdc++`

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、libatomic, libgomp, libquadmath, libssp, libvtv, C++ 標準ライブラリ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ると、クロスコンパイラー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。

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

このオプションは C コンパイラーおよび C++ コンパイラー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。

GCC 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

ここでの GCC ビルドにおいては、内部にあるシステムヘッダーファイルをいくつかインストールしました。そのうちの `limits.h` というものは、対応するシステムヘッダーファイルである `limits.h` を読み込むものになっています。

そのファイルはここでは `$LFS/usr/include/limits.h` になります。ただし GCC をビルドしたこの時点において `$LFS/usr/include/limits.h` は存在していません。したがってインストールされたばかりの内部ヘッダーファイルは、部分的に自己完結したファイルとなり、システムヘッダーファイルによる拡張された機能を含むものになっていません。glibc をビルドする際にはこれでもかまわないのですが、後々内部ヘッダーファイルは完全なものが必要になります。以下のようなコマンドを通じて、その内部ヘッダーファイルの完成版を作り出します。このコマンドは GCC ビルドが通常行っている方法と同じものです。

```
cd ..
cat gcc/limitx.h gcc/glimits.h gcc/limity.h > \
  `dirname $(LFS_TGT-gcc -print-libgcc-file-name)`/install-tools/include/limits.h
```

本パッケージの詳細は「GCC の構成」を参照してください。

5.4. Linux-5.19.2 API ヘッダー

Linux API ヘッダー (linux-5.19.2.tar.xz 内) 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: 1.4 GB

5.4.1. Linux API ヘッダー 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 ヘッダーファイルを「健全化 (sanitizing)」して利用します。

本パッケージ内にある不適切なファイルを残さないように、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ーファイルをソースから抽出します。推奨されている make ターゲット「headers_install」は利用できません。なぜなら rsync が必要となり、この時点では利用できないからです。ヘッダーファイルは初めに ./usr にコピーし、その後に必要な場所にコピーされます。

```
make headers
find usr/include -type f ! -name '*.h' -delete
cp -rv usr/include $LFS/usr
```

5.4.2. Linux API ヘッダー の構成

インストールヘッダー: /usr/include/asm/*.h, /usr/include/asm-generic/*.h, /usr/include/drm/*.h, /usr/include/linux/*.h, /usr/include/misc/*.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/misc, /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	Linux API ASM ヘッダーファイル
/usr/include/asm-generic/*.h	Linux API ASM の汎用的なヘッダーファイル
/usr/include/drm/*.h	Linux API DRM ヘッダーファイル
/usr/include/linux/*.h	Linux API Linux ヘッダーファイル
/usr/include/misc/*.h	Linux API のさまざまなヘッダーファイル
/usr/include/mtd/*.h	Linux API MTD ヘッダーファイル
/usr/include/rdma/*.h	Linux API RDMA ヘッダーファイル
/usr/include/scsi/*.h	Linux API SCSI ヘッダーファイル
/usr/include/sound/*.h	Linux API Sound ヘッダーファイル
/usr/include/video/*.h	Linux API Video ヘッダーファイル
/usr/include/xen/*.h	Linux API Xen ヘッダーファイル

5.5. Glibc-2.36

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: 4.4 SBU

必要ディスク容量: 821 MB

5.5.1. Glibc のインストール

はじめに LSB コンプライアンスに合うように、シンボリックリンクを生成します。さらに x86_64 向けとして、互換のシンボリックリンクを生成して、ダイナミックライブラリローダーが適切に動作するようにします。

```
case $(uname -m) in
  i?86)   ln -sfv ld-linux.so.2 $LFS/lib/ld-lsb.so.3
  ;;
  x86_64) ln -sfv ../lib/ld-linux-x86-64.so.2 $LFS/lib64
          ln -sfv ../lib/ld-linux-x86-64.so.2 $LFS/lib64/ld-lsb-x86-64.so.3
  ;;
esac
```



注記

上記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。

Glibc のプログラムの中で、FHS コンプライアンスに適合しない /var/db ディレクトリを用いているものがあり、そこに実行時データを保存しています。以下のパッチを適用することで、実行時データの保存ディレクトリを FHS に合致するものとします。

```
patch -Np1 -i ../glibc-2.36-fhs-1.patch
```

Glibc のドキュメントでは、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 build
cd      build
```

ldconfig と sln ユーティリティーを /usr/sbin にインストールするようにします。

```
echo "rootsbindir=/usr/sbin" > configparms
```

次に Glibc をコンパイルするための準備をします。

```
../configure \
  --prefix=/usr \
  --host=$LFS_TGT \
  --build=$(../scripts/config.guess) \
  --enable-kernel=3.2 \
  --with-headers=$LFS/usr/include \
  libc_cv_slibdir=/usr/lib
```

configure オプションの意味

`--host=$LFS_TGT, --build=$(../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ー、クロスリンカーを使って Glibc がクロスコンパイルされるようになります。

`--enable-kernel=3.2`

Linux カーネル 3.2 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=$LFS/usr/include`

これまでに \$LFS/usr/include ディレクトリにインストールしたヘッダーファイルを用いて 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 が得られることとなります。

`libc_cv_slibdir=/usr/lib`

この指定は 64 ビットマシンにおいて、ライブラリのインストール先をデフォルトの /lib64 ではなく /usr/lib とします。

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。



注記

本パッケージは "並行ビルド (parallel make)" を行うとビルドに失敗するとの報告例があります。もしビルドに失敗した場合は make コマンドに "-j1" オプションをつけて再ビルドしてください。

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。



警告

LFS が適切に設定されていない状態で、推奨する方法とは異なり root によってビルドを行うと、次のコマンドはビルドした glibc をホストシステムにインストールしてしまいます。これを行ってしまうと、ほぼ間違いなくホストが利用不能になります。したがってその環境変数が適切に設定されていることを確認してから、以下のコマンドを実行してください。

```
make DESTDIR=$LFS install
```

make install オプションの意味

```
DESTDIR=$LFS
```

make 変数 DESTDIR はほとんどすべてのパッケージにおいて、そのパッケージをインストールするディレクトリを定義するために利用されています。これが設定されていない場合のデフォルトは、ルートディレクトリ (/) となります。ここではパッケージのインストール先を \$LFS とします。これは「Chroot 環境への移行」に入ってからルートディレクトリとなります。

ldd スクリプト内にある実行可能なローダーへのパスがハードコーディングされているので、これを修正します。

```
sed '/RTLDLIST=/s@/usr@@g' -i $LFS/usr/bin/ldd
```



注意

この時点で以下を必ず実施します。新しいツールチェーンの基本的な機能 (コンパイルやリンク) が正常に処理されるかどうかを確認することです。健全性のチェック (sanity check) を行うものであり、以下のコマンドを実行します。

```
echo 'int main(){}' | gcc -xc -
readelf -l a.out | grep ld-linux
```

すべてが正常に処理され、エラーが発生しなければ、最終のコマンドの実行結果として以下が出力されるはずです。

```
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

インタプリタ名は 32 ビットマシンの場合 /lib/ld-linux.so.2 となります。

出力結果が上とは異なったり、あるいは何も出力されなかったりした場合は、どこかに不備があります。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。

すべてが完了したら、テストファイルを削除します。

```
rm -v a.out
```



注記

次節にてビルドするパッケージでは、ツールチェーンが正しく構築できたかどうかを再度チェックすることになります。特に binutils 2 回めや gcc 2 回めのビルドに失敗したら、それ以前にインストールしてきた binutils, GCC, glibc のいずれかにてビルドがうまくできていないことを意味します。

ここでクロスツールチェーンが完成しました。そこで limits.h のインストールを確定させます。これには GCC 開発者が提供するユーティリティを実行します。

```
$LFS/tools/libexec/gcc/$LFS_TGT/12.2.0/install-tools/mkheaders
```

本パッケージの詳細は「Glibc の構成」を参照してください。

5.6. GCC-12.2.0 から取り出した libstdc++

Libstdc++ は標準 C++ ライブラリです。(GCC の一部が C++ によって書かれているため) C++ をコンパイルするために必要となります。ただし gcc 1 回め をビルドするにあたっては、このライブラリのインストールを個別に行わなければなりません。それはこのライブラリが glibc に依存していて、対象ディレクトリ内ではまだ glibc が利用できない状態にあるからです。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 1.1 GB

5.6.1. Libstdc++ のインストール



注記

libstdc++ のソースは GCC に含まれます。したがってまずは GCC の tarball を伸張 (解凍) した上で gcc-12.2.0 ディレクトリに入って作業を進めます。

libstdc++ のためのディレクトリを新たに生成して移動します。

```
mkdir -v build
cd build
```

libstdc++ をコンパイルするための準備をします。

```
../libstdc++-v3/configure \
--host=$LFS_TGT \
--build=$(../config.guess) \
--prefix=/usr \
--disable-multilib \
--disable-nls \
--disable-libstdcxx-pch \
--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/12.2.0
```

configure オプションの意味

`--host=...`

利用するクロスコンパイラを指示するものであり、/usr/bin にあるものではなく、まさに先ほど作り出したものを指定するものです。

`--disable-libstdcxx-pch`

本スイッチは、既にコンパイルされたインクルードファイルをインストールしないようにします。これはこの時点では必要ないためです。

`--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/12.2.0`

インクルードファイルをインストールするディレクトリを指定します。libstdc++ は LFS における標準 C++ ライブラリであるため、そのディレクトリは C++ コンパイラ (\$LFS_TGT-g++) が標準 C++ インクルードファイルを探し出すディレクトリでなければなりません。通常のビルドにおいてそのディレクトリ情報は、最上位ディレクトリの configure のオプションにて指定します。ここでの作業では、上のようにして明示的に指定します。C++ コンパイラは sysroot パスに \$LFS (GCC 1 回めのビルド時に指定) をインクルードファイルの検索パスに加えます。したがって実際には \$LFS/tools/\$LFS_TGT/include/c++/12.2.0 となります。DESTDIR 変数 (以下の make install にて指定) とこのスイッチを組み合わせることで、ヘッダーファイルをそのディレクトリにインストールするようにします。

libstdc++ をコンパイルします。

```
make
```

ライブラリをインストールします。

```
make DESTDIR=$LFS install
```

クロスコンパイルにとっては libtool アーカイブファイルが邪魔になるため削除します。

```
rm -v $LFS/usr/lib/lib{stdc++,stdc++fs,supc++}.la
```

本パッケージの詳細は「GCC の構成」を参照してください。

第6章 クロスコンパイルによる一時的ツール

6.1. はじめに

本章では、つい先ほど作り出したクロスツールチェーンを利用して、基本ユーティリティーをクロスコンパイルする方法を示します。このユーティリティーは最終的な場所にインストールされますが、まだ利用することはできません。基本的な処理タスクは、まだホストのツールに依存します。ただしインストールされたライブラリは、リンクの際に利用されます。

ユーティリティーの利用は次の章において、「chroot」環境に入ってから可能になります。ただしそこに至る前の章の中で、パッケージをすべて作り出しておく必要があります。したがってホストシステムからは、まだ独立している状態ではありません。

ここでもう一度確認しておきますが、root ユーザーとしてビルドを行う際にも LFS の適切な設定が必要です。それができていないと、コンピューターが利用できなくなる可能性があります。本章は全体にわたって、lfs ユーザーにより操作します。環境は「環境設定」に示したものとなっている必要があります。

6.2. M4-1.4.19

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: 32 MB

6.2.1. M4 のインストール

M4 をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「M4 の構成」を参照してください。

6.3. Ncurses-6.3

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:  50 MB
```

6.3.1. Ncurses のインストール

ビルドにあたって `gawk` が必ず最初に見つかるようにします。

```
sed -i s/mawk// configure
```

そして以下のコマンドを実行して、ビルドホスト上に「tic」プログラムをビルドします。

```
mkdir build
pushd build
  ../configure
  make -C include
  make -C progs tic
popd
```

Ncurses をコンパイルするための準備をします。

```
./configure --prefix=/usr          \
            --host=$LFS_TGT        \
            --build=$(./config.guess) \
            --mandir=/usr/share/man \
            --with-manpage-format=normal \
            --with-shared          \
            --without-normal       \
            --with-cxx-shared      \
            --without-debug        \
            --without-ada          \
            --disable-stripping    \
            --enable-widenc
```

`configure` オプションの意味

`--with-manpage-format=normal`

本パラメーターは Ncurses が圧縮された man ページをインストールしないようにします。ホストディストリビューションそのものが圧縮 man ページを利用していると、同じようになってしまうからです。

`--with-shared`

これは Ncurses において共有 C ライブラリをビルドしインストールします。

`--without-normal`

これは Ncurses においてスタティクな C ライブラリのビルドおよびインストールを行わないようにします。

`--without-debug`

これは Ncurses においてデバッグライブラリのビルドおよびインストールを行わないようにします。

`--with-cxx-shared`

これは Ncurses において共有 C++ バインディングをビルドしインストールします。同時にスタティクな C++ バインディングのビルドおよびインストールは行わないようにします。

`--without-ada`

このオプションは Ncurses に対して Ada コンパイラー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。

`--disable-stripping`

本スイッチは、ホスト上にある `strip` を使って、ビルドシステム内のプログラムのストリップを行わないようにします。クロスコンパイルされたプログラムに対して、ホスト上のツールを使うと、ビルド失敗の原因になります。

`--enable-widenc`

本スイッチは通常のライブラリ (`libncurses.so.6.3`) ではなくワイド文字対応のライブラリ (`libncursesw.so.6.3`) 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

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
```

パッケージをインストールします。

```
make DESTDIR=$LFS TIC_PATH=$(pwd)/build/progs/tic install  
echo "INPUT(-lncursesw)" > $LFS/usr/lib/libncurses.so
```

install オプションの意味

```
TIC_PATH=$(pwd)/build/progs/tic
```

ビルドマシン上において、作り出したばかりの tic のパスを示すことが必要です。こうすることで terminal データベースがエラーなく生成できることとなります。

```
echo "INPUT(-lncursesw)" > $LFS/usr/lib/libncurses.so
```

パッケージの中で、わずかですが libncurses.so を必要としているものがあります。これはすぐに生成する予定のもので、ここでこの小さなリンカースクリプトを生成します。これは 第 8 章 においてビルドします。

本パッケージの詳細は 「Ncurses の構成」を参照してください。

6.4. Bash-5.1.16

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: 64 MB

6.4.1. Bash のインストール

Bash をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --build=$(support/config.guess) \
            --host=$LFS_TGT \
            --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
```

パッケージをインストールします。

```
make DESTDIR=$LFS 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 -sv bash $LFS/bin/sh
```

本パッケージの詳細は「Bash の構成」を参照してください。

6.5. Coreutils-9.1

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: 0.6 SBU
必要ディスク容量: 163 MB

6.5.1. 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=/usr \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess) \
            --enable-install-program=hostname \
            --enable-no-install-program=kill,uptime
```

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
```

パッケージをインストールします。

```
make DESTDIR=$LFS install
```

プログラムを、最終的に期待されるディレクトリに移動させます。この一時的環境にとっては必要なことではありませんが、これを実施するのは、実行モジュールの場所をハードコーディングしているプログラムがあるからです。

```
mv -v $LFS/usr/bin/chroot $LFS/usr/sbin
mkdir -pv $LFS/usr/share/man/man8
mv -v $LFS/usr/share/man/man1/chroot.1 $LFS/usr/share/man/man8/chroot.8
sed -i 's/"1"/"8"/'
```

本パッケージの詳細は「Coreutils の構成」を参照してください。

6.6. Diffutils-3.8

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: 0.2 SBU
必要ディスク容量: 26 MB

6.6.1. 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 --host=$LFS_TGT
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Diffutils の構成」を参照してください。

6.7. File-5.42

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: 34 MB

6.7.1. File のインストール

ホストシステム上の file コマンドは、これから生成する同コマンドと同一バージョンでなければなりません。これはシグニチャーファイル生成のために必要となります。そこで以下のコマンドを実行してビルドします。

```
mkdir build
pushd build
  ../configure --disable-bzlib      \
                --disable-libseccomp \
                --disable-xzlib     \
                --disable-zlib
make
popd
```

configure オプションの意味

`--disable-*`

configure スクリプトは、ホスト上に特定のライブラリが存在するときに、それを利用しようとします。ライブラリが存在していて、かつそれに対応するヘッダーファイルが存在していないときに、コンパイルに失敗することがあります。このオプションは、そういったホストの機能は不要なので利用しないようにします。

File をコンパイルするための準備をします。

```
./configure --prefix=/usr --host=$LFS_TGT --build=$(./config.guess)
```

パッケージをコンパイルします。

```
make FILE_COMPILE=$(pwd)/build/src/file
```

パッケージをインストールします。

```
make DESTDIR=$LFS install
```

クロスコンパイルにとっては libtool アーカイブファイルが邪魔になるため削除します。

```
rm -v $LFS/usr/lib/libmagic.la
```

本パッケージの詳細は「File の構成」を参照してください。

6.8. Findutils-4.9.0

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: 0.2 SBU
必要ディスク容量: 42 MB

6.8.1. 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 \
            --localstatedir=/var/lib/locate \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Findutils の構成」を参照してください。

6.9. Gawk-5.1.1

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: 45 MB

6.9.1. Gawk のインストール

はじめに、必要のないファイルはインストールしないようにします。

```
sed -i 's/extras//' Makefile.in
```

Gawk をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Gawk の構成」を参照してください。

6.10. Grep-3.7

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: 0.2 SBU
必要ディスク容量: 25 MB

6.10.1. 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 \
            --host=$LFS_TGT
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Grep の構成」を参照してください。

6.11. Gzip-1.12

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
必要ディスク容量: 11 MB

6.11.1. 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 --host=$LFS_TGT
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Gzip の構成」を参照してください。

6.12. Make-4.3

Make パッケージは、対象となるパッケージのソースファイルを用いて、実行モジュールやそれ以外のファイルの生成、管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: 15 MB

6.12.1. 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 \
            --without-guile \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

configure オプションの意味

--without-guile

ここではクロスコンパイルをしているにもかかわらず、ビルドホスト内に guile が存在すると configure がそれを見つけて利用しようとしています。 そうなってしまうとコンパイルが失敗します。 そこで本スイッチにより、そうならないようにします。

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 DESTDIR=$LFS install
```

本パッケージの詳細は 「Make の構成」 を参照してください。

6.13. Patch-2.7.6

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
必要ディスク容量: 12 MB

6.13.1. 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=/usr \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Patch の構成」を参照してください。

6.14. Sed-4.8

Sed パッケージはストリームエディターを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 20 MB

6.14.1. 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 \
            --host=$LFS_TGT
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Sed の構成」を参照してください。

6.15. Tar-1.34

Tar パッケージは tar アーカイブの生成を行うとともに、アーカイブ操作に関する多くの処理を提供します。Tar はすでに生成されているアーカイブからファイルを抽出したり、ファイルを追加したりします。あるいはすでに保存されているファイルを更新したり一覧を表示したりします。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 38 MB

6.15.1. Tar のインストール

Tar をコンパイルするための準備をします。

```
./configure --prefix=/usr          \  
            --host=$LFS_TGT        \  
            --build=$(build-aux/config.guess)
```

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 DESTDIR=$LFS install
```

本パッケージの詳細は「Tar の構成」を参照してください。

6.16. Xz-5.2.6

Xz 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 lzma フォーマットおよび新しい xz 圧縮フォーマットを取り扱います。xz コマンドによりテキストファイルを圧縮すると、従来の gzip コマンドや bzip2 コマンドに比べて、高い圧縮率を実現できます。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 16 MB

6.16.1. Xz のインストール

Xz をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess) \
            --disable-static \
            --docdir=/usr/share/doc/xz-5.2.6
```

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 DESTDIR=$LFS install
```

クロスコンパイルにとっては libtool アーカイブファイルが邪魔になるため削除します。

```
rm -v $LFS/usr/lib/liblzma.la
```

本パッケージの詳細は「Xz の構成」を参照してください。

6.17. Binutils-2.39 - 2回め

Binutils パッケージは、リンカーやアセンブラー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: 1.4 SBU
必要ディスク容量: 514 MB

6.17.1. Binutils のインストール

Binutils の tarball では、古い libtool のコピーが提供されています。これは sysroot サポートが行われていないので、ビルドされるバイナリが誤ってホストディストロのライブラリにリンクされてしまいます。この問題を以下により回避します。

```
sed '6009s/$add_dir//' -i ltmain.sh
```

ビルドのためのディレクトリを再び生成します。

```
mkdir -v build
cd      build
```

Binutils をコンパイルするための準備をします。

```
../configure \
  --prefix=/usr \
  --build=$(../config.guess) \
  --host=$LFS_TGT \
  --disable-nls \
  --enable-shared \
  --enable-gprofng=no \
  --disable-werror \
  --enable-64-bit-bfd
```

configure オプションの意味

--enable-shared
libbfd を共有ライブラリとしてビルドします。

--enable-64-bit-bfd
64 ビットサポートを有効にします (ホスト上にて、より小さなワードサイズとします)。64 ビットシステムにおいては不要ですが、不具合を引き起こすものではありません。

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 DESTDIR=$LFS install
```

クロスコンパイルにとっては libtool アrchiveファイルが邪魔になるため削除し、不要なスタティックライブラリも削除します。

```
rm -v $LFS/usr/lib/lib{bfd,ctf,ctf-nobfd,opcodes}.{a,la}
```

本パッケージの詳細は「Binutils の構成」を参照してください。

6.18. GCC-12.2.0 - 2回め

GCC パッケージは C コンパイラーや C++ コンパイラーなどの GNU コンパイラーコレクションを提供します。

```
概算ビルド時間:      15 SBU
必要ディスク容量:  4.5 GB
```

6.18.1. GCC のインストール

GCC の 1 回目のビルドと同様に、ここでも 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 -xf ../mpfr-4.1.0.tar.xz
mv -v mpfr-4.1.0 mpfr
tar -xf ../gmp-6.2.1.tar.xz
mv -v gmp-6.2.1 gmp
tar -xf ../mpc-1.2.1.tar.gz
mv -v mpc-1.2.1 mpc
```

x86_64 上でビルドしている場合は、64ビットライブラリのデフォルトディレクトリ名を「lib」にします。

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' -i.orig gcc/config/i386/t-linux64
    ;;
esac
```

libgcc と libstdc++ のヘッダーのビルドルールを変更して、これらのライブラリに対して POSIX スレッドサポートを含めてビルドするようにします。

```
sed '/thread_header =/s/@.*@/gthr-posix.h/' \
-i libgcc/Makefile.in libstdc++-v3/include/Makefile.in
```

専用のディレクトリを再度生成します。

```
mkdir -v build
cd      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。

```
../configure \
--build=$(../config.guess) \
--host=$LFS_TGT \
--target=$LFS_TGT \
LDFLAGS_FOR_TARGET=-L$PWD/$LFS_TGT/libgcc \
--prefix=/usr \
--with-build-sysroot=$LFS \
--enable-initfini-array \
--disable-nls \
--disable-multilib \
--disable-decimal-float \
--disable-libatomic \
--disable-libgomp \
--disable-libquadmath \
--disable-libssp \
--disable-libvtv \
--enable-languages=c,c++
```

configure オプションの意味

`--with-build-sysroot=$LFS`

通常は `--host` を用いれば、GCC ビルドにクロスコンパイラーが用いられ、参照すべきヘッダーやライブラリも `$LFS` にあるものが用いられるように指示されます。しかし今ビルドを行っているシステム上の GCC は別のツール

を使っているので、上のような場所を認識できていません。本スイッチは、必要なファイルをホスト内からではなく、`$LFS` から探し出すようにします。

```
--target=$LFS_TGT
```

これまでの GCC はクロスコンパイルによって作り出してきたので、コンパイル済み GCC 実行ファイルからターゲットライブラリ (`libgcc` と `libstdc++`) をビルドして作り出すことができません。なぜならその実行ファイルはホストディストリビューション上で動作させられないからです。GCC ビルドシステムはその回避策として、デフォルトではホスト上にある C および C++ コンパイラーを利用しようとしています。ただし GCC のバージョンが異なる場合に、GCC ターゲットライブラリをビルドすることはサポートされていません。したがってホスト上のコンパイラーがビルドに失敗する可能性があります。本パラメーターは、確実に GCC 1 回めの実行ファイルを使ってライブラリビルドを行うようにして、問題が発生しないようにします。

```
LDFLAGS_FOR_TARGET=...
```

GCC 1 回めではスタティックバージョンの `libgcc` をビルドしていましたが、ここでは共有の `libgcc` をビルドするようにします。これは C++ 例外処理のために必要となります。

```
--enable-initfini-array
```

本オプションを指定すれば、自動的に x86 上のネイティブコンパイラーを使って、ネイティブコンパイラーをビルドするようにします。しかしここではクロスコンパイラーを作り出すつもりでいます。したがって明示的に本オプションへ指定が必要になります。

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 DESTDIR=$LFS install
```

最後に、便利な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 コンパイラー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 コンパイラー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 -sv gcc $LFS/usr/bin/cc
```

本パッケージの詳細は「GCC の構成」を参照してください。

第7章 chroot への移行と一時的ツールの追加ビルド

7.1. はじめに

本章では、一時的システムに足りていない最後の部分をビルドしていきます。つまり、多くのパッケージビルドに必要なツールをビルドします。こうして循環的な相互参照の関係が解決するので、これまで利用してきたホストオペレーティングシステムから完全に離れて（実行中のカーネルは除きますが）”chroot”環境に入って、ビルドを行っていきます。

chroot 環境内では適切な操作とするため、実行されているカーネルとのやり取りを確実に行います。それはいわゆる仮想カーネルファイルシステムを通じて行うものです。chroot 環境に入る際には、あらかじめマウントされていなければなりません。マウントがされているかどうかを確認する場合は `findmnt` を実行します。

「Chroot 環境への移行」まで、コマンドの実行は LFS を設定した上で、root ユーザーにより行う必要があります。chroot 環境に入っても、コマンドはすべて root 実行ですが、もう安心です。LFS を構築しているコンピューター上の OS にはもうアクセスしないからです。かと言ってコマンド実行を誤れば、簡単に LFS システムを壊してしまうことになりますから、十分に注意してください。

7.2. 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。

`$LFS` ディレクトリ配下の所有者は今では lfs ユーザーであり、これはホストシステム上にのみ存在するユーザーです。この `$LFS` ディレクトリ配下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、存在しないユーザーによって所有される形を生み出すこととなります。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` の全ファイルの所有者となるので、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 `$LFS/*` ディレクトリの所有者を root ユーザーにします。以下のコマンドによりこれを実現します。

```
chown -R root:root $LFS/{usr,lib,var,etc,bin,sbin,tools}
case $(uname -m) in
  x86_64) chown -R root:root $LFS/lib64 ;;
esac
```

7.3. 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さまざま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 -pv $LFS/{dev,proc,sys,run}
```

7.3.1. /dev のマウントと有効化

通常のブートの際には、カーネルは `/dev` ディレクトリ上に `devtmpfs` ファイルシステムを自動的にマウントします。そしてデバイスが検出されたりアクセスされたりするたびに、デバイスが仮想ファイルシステムを動的生成できるようにしています。このデバイス生成処理は一般的には、システム起動時にカーネルと 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、再起動も行っていないので `/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
```

7.3.2. 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 -v --bind /dev/pts $LFS/dev/pts
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

ホストシステムによっては /dev/shm が /run/shm へのシンボリックリンクになっているものがあります。上の作業にて /run tmpfs がマウントされましたが、これはこのディレクトリを生成する必要がある時のみです。

```
if [ -h $LFS/dev/shm ]; then
    mkdir -pv $LFS/${readlink $LFS/dev/shm}
fi
```

7.4. Chroot 環境への移行

残るツール類をビルドするために必要なパッケージは、ここまでですべてビルドしました。そこで chroot 環境に入って、残りの一時的ツールをインストールしていきます。この環境は、最終システムに向けたインストールを行う際にも使います。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 "$LFS" /usr/bin/env -i \
    HOME=/root \
    TERM="$TERM" \
    PS1='(lfs chroot) \u:\w\$ ' \
    PATH=/usr/bin:/usr/sbin \
    /bin/bash --login
```

env コマンドの `-i` パラメーターは、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 が必要であれば、ここで定義しておくといいでしょう。

ここからは 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 内には存在しません。つまりクロスチェーンは chroot 環境内ではもはや利用しないということです。

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、「/dev のマウントと有効化」と「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。

7.5. ディレクトリの生成

LFS ファイルシステムにおける完全なディレクトリ構成を作り出していきます。



注記

本節において触れるディレクトリの中には、明示的な指示か、あるいは何かのパッケージインストールによってすでに生成済みであるものがあります。以下では完全を期して繰り返し生成することになります。

ルートレベルのディレクトリをいくつか生成します。これは前章において必要としていた限定的なものの中には含まれていないものです。以下のコマンドを実行して生成します。

```
mkdir -pv /{boot,home,mnt,opt,srv}
```

ルートレベル配下に、必要となる一連のサブディレクトリを、以下のコマンドにより生成します。

```
mkdir -pv /etc/{opt,sysconfig}
mkdir -pv /lib/firmware
mkdir -pv /media/{floppy,cdrom}
mkdir -pv /usr/{,local/}{include,src}
mkdir -pv /usr/local/{bin,lib,sbin}
mkdir -pv /usr/{,local/}share/{color,dict,doc,info,locale,man}
mkdir -pv /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
mkdir -pv /var/{cache,local,log,mail,opt,spool}
mkdir -pv /var/lib/{color,misc,locate}

ln -sfv /run /var/run
ln -sfv /run/lock /var/lock

install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
```

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。

7.5.1. 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 <https://refspecs.linuxfoundation.org/fhs.shtml> に示されています。) FHS では、任意のディレクトリとして /usr/local/games や /usr/share/games などを規定しています。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。他のディレクトリについては、どうぞ自由に取り決めて作成してください。

7.6. 重要なファイルとシンボリックリンクの生成

Linux のこれまでの経緯として、マウントされているファイルシステムの情報は /etc/mtab ファイルに保持されています。最新の Linux であれば、内部的にこのファイルを管理し、ユーザーに対しては /proc ファイルシステムを通じて情報提示しています。/etc/mtab ファイルの存在を前提としているプログラムが正常動作するように、以下のシンボリックリンクを作成します。

```
ln -sv /proc/self/mounts /etc/mtab
```

テストスイートの中に /etc/hosts ファイルを参照するものがあるので、単純なものをここで生成します。これは Perl の設定ファイルにおいても参照されます。

```
cat > /etc/hosts << EOF
127.0.0.1 localhost $(hostname)
:::1 localhost
EOF
```

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:/usr/bin/false
daemon:x:6:6:Daemon User:/dev/null:/usr/bin/false
messagebus:x:18:18:D-Bus Message Daemon User:/run/dbus:/usr/bin/false
uidd:x:80:80:UUID Generation Daemon User:/dev/null:/usr/bin/false
nobody:x:65534:65534:Unprivileged User:/dev/null:/usr/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。

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:daemon
sys:x:2:
kmem:x:3:
tape:x:4:
tty: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:
adm:x:16:
messagebus:x:18:
input:x:24:
mail:x:34:
kvm:x:61:
uidd:x:80:
wheel:x:97:
users:x:999:
nogroup:x:65534:
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 9 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。またテストスイートにて特定のユーザーやグループを必要としているものがあります。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。GID 5 は `tty` グループに対して広く用いられています。また数値 5 は `devpts` ファイルシステムに対して `/etc/fstab` においても用いられています。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。

ID 65534 は NFS のカーネルが利用し、マップされていないユーザーやグループに対するユーザー名前空間を切り分けます (これは NFS サーバー上や親のユーザー空間に存在しますが、ローカルマシンや分離された名前空間には存在しません)。未割り当ての ID を避けるために、この ID を `nobody` と `nogroup` に用いることにします。他のディストリビューションにおいては、この ID を異なる用い方をしている場合があるため、移植性を考慮するプログラムでは、ここでの割り当てに依存しないようにしてください。

第 8 章におけるテストの中には、通常のユーザーを必要とするものがあります。ここでそういったユーザーをここで追加し、その章の最後には削除します。

```
echo "tester:x:101:101:~/home/tester:/bin/bash" >> /etc/passwd
echo "tester:x:101:" >> /etc/group
install -o tester -d /home/tester
```

プロンプトの「I have no name!」を取り除くために新たなシェルを起動します。 /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 /usr/bin/bash --login
```

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/log/{btmp,lastlog,faillog,wtmp}  
chgrp -v utmp /var/log/lastlog  
chmod -v 664 /var/log/lastlog  
chmod -v 600 /var/log/btmp
```

/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/faillog ファイルはログインに失敗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。



注記

/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。このファイルはブートスクリプトが動的に生成します。

7.7. Gettext-0.21

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: 1.6 SBU
必要ディスク容量: 282 MB

7.7.1. 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3つのバイナリを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。

```
./configure --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
```

msgfmt, msgmerge, xgettext の各プログラムをインストールします。

```
cp -v gettext-tools/src/{msgfmt,msgmerge,xgettext} /usr/bin
```

本パッケージの詳細は「Gettext の構成」を参照してください。

7.8. Bison-3.8.2

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: 0.3 SBU
必要ディスク容量: 57 MB

7.8.1. 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 \  
            --docdir=/usr/share/doc/bison-3.8.2
```

configure オプションの意味

`--docdir=/usr/share/doc/bison-3.8.2`

ビルドシステムに対して、bison のドキュメントをインストールするディレクトリを、バージョンつきとします。

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
```

本パッケージの詳細は「Bison の構成」を参照してください。

7.9. Perl-5.36.0

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: 1.6 SBU
必要ディスク容量: 282 MB

7.9.1. Perl のインストール

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 \
-Dprivlib=/usr/lib/perl5/5.36/core_perl \
-Darchlib=/usr/lib/perl5/5.36/core_perl \
-Dsitelib=/usr/lib/perl5/5.36/site_perl \
-Dsitearch=/usr/lib/perl5/5.36/site_perl \
-Dvendorlib=/usr/lib/perl5/5.36/vendor_perl \
-Dvendorarch=/usr/lib/perl5/5.36/vendor_perl
```

Configure オプションの意味

`-des`
これは三つのオプションを組み合わせたものです。 `-d` はあらゆる項目に対してデフォルト設定を用います。 `-e` はタスクをすべて実施します。 `-s` は不要な出力は行わないようにします。

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
```

本パッケージの詳細は「Perl の構成」を参照してください。

7.10. Python-3.10.6

Python 3 パッケージは Python 開発環境を提供します。オブジェクト指向プログラミング、スクリプティング、大規模プログラムのプロトタイピング、アプリケーション開発などに有用なものです。

概算ビルド時間: 0.9 SBU
必要ディスク容量: 364 MB

7.10.1. Python のインストール



注記

「python」の名前で始まるパッケージファイルは 2 種類あります。そのうち、扱うべきファイルは Python-3.10.6.tar.xz です。（1 文字めが大文字であるものです。）

Python をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --enable-shared \
            --without-ensurepip
```

configure パラメーターの意味

`--enable-shared`

このスイッチはスタティックライブラリをインストールしないようにします。

`--without-ensurepip`

このスイッチは Python パッケージインストーラーを無効にします。この段階では必要がないからです。

パッケージをコンパイルします。

```
make
```



注記

この時点において、依存パッケージをまだインストールしていないために、ビルドできない Python 3 モジュールがあります。それでもビルドシステムは、そのようなモジュールをビルドしようとします。そして一部のファイルのコンパイルが失敗して、コンパイラメッセージには「致命的エラー」が示されます。このメッセージは無視できます。よく確認すべきなのは、トップレベルの make コマンドは失敗していないことです。任意でビルドすれば良いモジュールは、今ここでのビルドは必要ありません。それは、この後に 第 8 章においてビルドされます。

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「Python 3 の構成」を参照してください。

7.11. Texinfo-6.8

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: 113 MB

7.11.1. 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
```

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「Texinfo の構成」を参照してください。

7.12. Util-linux-2.38.1

Util-linux パッケージはさまざまなユーティリティープログラムを提供します。

概算ビルド時間: 0.6 SBU
必要ディスク容量: 149 MB

7.12.1. Util-linux のインストール

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。そこで以下によりそのディレクトリを生成します。

```
mkdir -pv /var/lib/hwclock
```

Util-linux をコンパイルするための準備をします。

```
./configure ADJTIME_PATH=/var/lib/hwclock/adjtime \  
  --libdir=/usr/lib \  
  --docdir=/usr/share/doc/util-linux-2.38.1 \  
  --disable-chfn-chsh \  
  --disable-login \  
  --disable-nologin \  
  --disable-su \  
  --disable-setpriv \  
  --disable-runuser \  
  --disable-pylibmount \  
  --disable-static \  
  --without-python \  
  runstatedir=/run
```

`configure` オプションの意味

`ADJTIME_PATH=/var/lib/hwclock/adjtime`

これはハードウェアクロックの情報を保持したファイルの場所を設定するものであり、FHS に従ったものです。一時的なツールにとって厳密には必要ではありませんが、別の場所にはファイル生成するわけにはいきません。最終的な `util-linux` パッケージをビルドする際に、上書きしたり削除したりすることができなくなるからです。

`--libdir=/usr/lib`

本スイッチは、共有ライブラリを示す `.so` シンボリックリンクを同一ディレクトリ (`/usr/lib`) に直接生成するようにします。

`--disable-*`

コンポーネントのビルドの際に、LFS にはない、あるいはまだインストールしていない別のパッケージがあり、そのために発生する警告メッセージを無効にします。

`--without-python`

本スイッチは Python を用いないようにします。ビルドの際に不要なバインディングを作らないようにするためです。

`runstatedir=/run`

本スイッチは `uidd` や `libuuid` が利用するソケットの場所を適切に設定します。

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
```

本パッケージの詳細は「Util-linux の構成」を参照してください。

7.13. 一時的システムのクリーンアップと保存

7.13.1. クリーンアップ

はじめに、現在インストールされているドキュメントは削除します。これを最終的なシステムに持ち込みません。これによって 35 MB を節約します。

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

libtool の .la ファイルはスタティックライブラリにリンクするときだけ使います。これらはダイナミック共有ライブラリを用いるとき、そして特に autotools 以外のビルドシステムを利用するときには不要であり、潜在的には支障を及ぼします。したがって chroot の中で、不要なファイルは削除します。

```
find /usr/{lib,libexec} -name \*.la -delete
```

現在のシステムサイズは、およそ 3 GB になりました。そして /tools ディレクトリは、もう必要がありません。ディスク容量は 1 GB 近くを占めています。ここで削除します。

```
rm -rf /tools
```

7.13.2. バックアップ

この時点において、基本的なプログラムやライブラリが生成されたので、現在の LFS システムの状態は良好なものです。このシステムを、後に再利用できるように、ここでバックアップを取ることができます。ここから先の章において、致命的な失敗をしてしまった場合は、すべてを削除して（今度はより慎重に）やり直すのが、一番のやり方であるのは、明らかです。ただし、そのときには一時システムも失ってしまっている状態です。余計な時間を費やすことなく、ビルドに成功したところまでのシステムを使ってやり直す策を考えるのであれば、ここで LFS システムのバックアップをとっておくことが、後々の役に立つかもしれません。



注記

本節の残りの作業は必須ではありません。ただし第 8 章においてパッケージのインストールを始めていくと、一時的ツールは上書きされていきます。そこで以下に示すように、現時点でのシステムのバックアップをとっておくのが良いでしょう。

以下の手順は chroot 環境の外から実施します。これはつまり chroot 環境から抜け出してから手順を進めていくということです。こうする理由は、バックアップアーカイブの保存や読み込みをするなら、ファイルシステムへのアクセスは chroot 環境の外部から行うべきであって、安全のため \$LFS ディレクトリ階層の内部において行うべきではないからです。

バックアップを取ることにしているのであれば、ここで chroot 環境から抜け出ます。

```
exit
```



重要

以降の手順はすべて、ホストシステム上の root ユーザーにより実施します。特にコマンド実行は、よく注意しながら行ってください。誤ったことをすると、ホストシステムを書き換えてしまうことになります。環境変数 LFS はデフォルトで lfs ユーザーにおいて設定していましたが、root ユーザーにおいては設定していないかもしれません。

root ユーザーによってコマンド実行する際にも、必ず LFS が設定されていることを確認してください。

このことは「変数 \$LFS の設定」において説明済みです。

バックアップを取る前には、仮想ファイルシステムをアンマウントします。

```
umount $LFS/dev/pts
umount $LFS/{sys,proc,run,dev}
```

バックアップアーカイブを生成したディレクトリを含むファイルシステムにおいて、未使用のディスク容量が最低でも 1 GB はあることを確認してください。（ソース tarball もバックアップアーカイブに含めます。）

なお、これ以降の手順説明においては、ホストシステム上の root ユーザーのホームディレクトリを用いています。これは通常、ルートファイルシステムに置かれているものです。

root ユーザーのホームディレクトリにバックアップを生成したくない場合は、\$HOME の内容を適切に書き換えてください。

バックアップアーカイブを生成するために、以下のコマンドを実行します。



注記

バックアップアーカイブは圧縮するので、かなりの高速なシステムを利用していても、比較的長い時間（10分以上）を要します。

```
cd $LFS
tar -cJpf $HOME/lfs-temp-tools-11.2.tar.xz .
```



注記

第 8 章を続けるのであれば、以降に示す「重要」の説明のように、chroot 環境に再度入ることを忘れないでください。

7.13.3. 復元

誤操作をしてしまい、初めからやり直す必要が出てきたとします。そんなときは上のバックアップを復元し、すばやく回復させることにしましょう。\$LFS 配下にソースも配置することになっているので、バックアップアーカイブ内にはそれらも含まれています。したがって再度ダウンロードする必要はありません。\$LFS が適切に設定されていることを再度確認した上で、バックアップの復元を行うための以下のコマンドを実行します。



警告

以下に示すコマンドは相当に危険です。root ユーザーになって rm -rf ./ * を実行する際に、\$LFS ディレクトリに移動していない、あるいは環境変数 LFS を設定していないとしたら、システム全体を破壊することになります。厳に警告しておきます。

```
cd $LFS
rm -rf ./ *
tar -xpf $HOME/lfs-temp-tools-11.2.tar.xz
```

環境変数が適切に設定されていることを再度確認の上、ここから続くシステムビルドに進んでいきます。



重要

chroot 環境から抜け出して、バックアップの生成を行った場合、あるいはビルド作業を再開する場合は、「仮想カーネルファイルシステムの準備」において説明している、カーネル仮想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 (findmnt | grep \$LFS)。もしマウントされていなかったら、マウントを行ってから、再び chroot 環境に入るようにしてください（「Chroot 環境への移行」参照）。

第IV部 LFSシステムの構築

第8章 基本的なソフトウェアのインストール

8.1. はじめに

この章では LFS システムの構築作業を始めます。

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、たいていは手短か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。

コンパイラー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。コンパイラー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。もしコンパイラー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う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ない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つたとしても、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が。

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

第 8 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。なお SBU 値はすべて、単一の CPU コア (-j1) を用いて算出しています。

8.1.1. ライブラリについて

LFS 編集者は全般にスタティックライブラリは作らないものとしています。スタティックライブラリが作られたそもその目的は、現在の Linux システムにとってはもはや古いものです。スタティックライブラリをリンクすると障害となることすらあります。例えばセキュリティ問題を解決するためにライブラリリンクを更新しなければならなくなったなら、スタティックライブラリにリンクしていたプログラムはすべて再構築しなければなりません。したがってスタティックライブラリを使うべきかどうかは、いつも迷うところであり、関連するプログラム（あるいはリンクされるプロシージャ）であってもどちらかに定めなければなりません。

本章の手順では、スタティックライブラリのインストールはたいてい行わないようにしています。多くのケースでは `configure` に対して `--disable-static` を与えることで実現しますが、これができない場合には他の方法を取ります。ただし `glibc` や `gcc` においては、一般的なパッケージビルドに必要であるため、スタティックライブラリを利用します。

ライブラリに関してのより詳細な議論については BLFS ブックの `Libraries: Static or shared?` を参照してください。

8.2. 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はなく、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で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せん。

LFS や BLFS においてパッケージ管理ツールに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かの方法があり、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。望むものがあるかどうか確認してみてください。

8.2.1. 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。

- カーネルをアップグレードする必要がある場合 (たとえば 5.10.17 から 5.10.18 や 5.11.1 へ、など)、これ以外に再ビルドを必要とするものではありません。カーネルとユーザー空間の境界が適切に定義されているため、システムは動作し続けるはずで、特に Linux API ヘッダーは、カーネルに伴ってアップグレードする必要もありません (次に説明するように、アップグレードしてはなりません)。アップグレードしたカーネルは、システムを再起動すれば利用できるようになります。
- Linux API ヘッダーや Glibc を新しいバージョン (例えば glibc-2.31 から glibc-2.32) にアップグレードする必要が発生した場合は LFS を再構築することが安全です。必要なパッケージの依存順を知っていれば再構築できるかもしれませんが、これはお勧めしません。
- 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、そのライブラリ名が変更になったとします。この場合は、このライブラリに動的リンクを行っていたパッケージは、新たなライブラリに向けてのリンクとなるように再コンパイルすることが必要になります。(なおパッケージバージョンとライブラリ名には関連性はありません。) たとえば foo-1.2.3 というパッケージがあって、これが共有ライブラリ libfoo.so.1 をインストールしているとします。そして新バージョン foo-1.2.4 が共有ライブラリ libfoo.so.2 を持っていて、これにアップグレードするものとします。この場合 libfoo.so.1 に動的リンクを行っていたパッケージは、すべて新ライブラリバージョン libfoo.so.2 へのリンクを行うように再コンパイルしなければなりません。そのように依存していたパッケージをすべて再コンパイルしてからでないと、古いバージョンのライブラリは削除するべきではありません。
- 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、そのライブラリ名には変更がなかったとします。ただしライブラリ名の変更はなくても、ライブラリファイルのバージョン番号が減らされたとします。(たとえばライブラリ名 libfoo.so.1 はそのまま不変であったとして、ライブラリファイル名が libfoo.so.1.25 から libfoo.so.1.24 に変更となった場合です。) この場合、それまでインストールされていたバージョン (例では libfoo.so.1.25) のライブラリファイルは削除すべきです。そうしておかないと、ldconfig を実行したときに (自分でコマンドライン実行したり、別のパッケージをインストールする際に実施されたりしたときに)、シンボリックリンク libfoo.so.1 がリセットされますが、それが指し示す先が古いライブラリファイルとなってしまいます。なぜならバージョン番号がより大きい方なので、そのバージョンの方が「より新しい」と解釈されるためです。こういった状況は、パッケージをダウングレードした場合や、パッケージにおけるバージョン番号づけの取り決めが突然変わってしまった場合に起こり得るものです。
- 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、そのライブラリ名に変更はなかったとします。ただしそこでは重大な問題 (特にセキュリティぜい弱性) が解消されているような場合は、この共有ライブラリにリンクしている実行中プログラムは、すべて再起動してください。アップグレードした後に、以下のコマンドを root で実行すると、どういったプログラムが古いバージョンのライブラリを利用しているかの一覧が表示されます。(libfoo の部分は、目的のライブラリ名に置き換えてください。)

```
grep -l -e 'libfoo.*deleted' /proc/*/maps |
tr -cd 0-9\\n | xargs -r ps u
```

OpenSSH を利用してシステムにアクセスしている場合であって、これがリンクするライブラリがアップデートされたとします。その場合は sshd サービスの再起動が必要です。またシステムからはいったんログアウトしてログインし直し、その後上のコマンドをもう一度実行して、削除されたライブラリを利用していないかどうかの確認を行ってください。

- バイナリや共有ライブラリが上書きされると、そのバイナリや共有バイナリ内のコードやデータを利用するプロセスがクラッシュすることがあります。プロセスがクラッシュしないように、バイナリや共有ライブラリを正しく更新する方法は、まず初めに削除を行ってから、新たなものをそこにインストールすることです。Coreutils が提供する install コマンドは、すでにこの処理が実装されているため、たいいていのパッケージにおいて、バイナリやライブラリのインストールに利用されています。したがってそのような問題に悩まされることは、これまでほとんどなかったはずで、しかしパッケージの中には (特に BLFS にある Mozilla JS など)、すでにあるファイルを上書きする方式をとっているため、クラッシュするものがあります。そこでパッケージ更新の前には、それまでの作業を保存して、不要な起動プロセスは停止することが安全です。

8.2.2. 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ーを用いる前に、さまざま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。

8.2.2.1. 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。

8.2.2.2. 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。

8.2.2.3. 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となります。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
```

この手法をサポートするパッケージは数多く存在しますが、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、手作業で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簡単かもしれません。

8.2.2.4. 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という利点がありますが、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。

8.2.2.5. 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。

インストールされるライブラリを事前にロードする場所を環境変数 `LD_PRELOAD` に定めておいてそれからインストールを行う方法です。パッケージのインストール中には `cp`、`install`、`mv` など、さまざま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します。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、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。

8.2.2.6. 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。インストールの後は、インストールされたファイルのアーカイブが生成されます。このアーカイブはローカルPCへのインストールに用いられ、他のPCへのインストールに利用されたりします。

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 <https://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。

8.2.2.7. 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 https://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。

8.2.3. 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ー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/sysconfig/rc.site`、`/etc/sysconfig/network`、`/etc/sysconfig/ifconfig.eth0`

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。

注記

類似するアーキテクチャーのシステム間にてコピーを行う際には問題が生じるとの報告があります。例えばインテルアーキテクチャーに対する命令セットは `AMD` プロセッサに対するものと完全に一致しているわけではないため、一方の命令セットが後に他方で動作しなくなることも考えられます。

最後に新システムを起動可能とするために 「`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。

8.3. Man-pages-5.13

Man-pages パッケージは 2,200 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: 33 MB

8.3.1. 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 prefix=/usr install
```

8.3.2. Man-pages の構成

インストールファイル: さまざまな 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。

8.4. Iana-Etc-20220812

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: 4.8 MB

8.4.1. Iana-Etc のインストール

このパッケージでは、必要とするファイルを所定の場所にコピーするだけにします。

```
cp services protocols /etc
```

8.4.2. 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 により利用可能なさまざまな 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。

8.5. Glibc-2.36

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: 24 SBU
必要ディスク容量: 2.8 GB

8.5.1. Glibc のインストール

Glibc のプログラムの中には /var/db ディレクトリに実行データを収容するものがあり、これは FHS に準拠していません。以下のパッチを適用することで、実行データの収容先を FHS 準拠のディレクトリとします。

```
patch -Np1 -i ../glibc-2.36-fhs-1.patch
```

Glibc のドキュメントでは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 build
cd build
```

ldconfig と sln ユーティリティーを /usr/sbin にインストールするようにします。

```
echo "rootsbindir=/usr/sbin" > configparms
```

Glibc をコンパイルするための準備をします。

```
../configure --prefix=/usr \
              --disable-werror \
              --enable-kernel=3.2 \
              --enable-stack-protector=strong \
              --with-headers=/usr/include \
              libc_cv_slibdir=/usr/lib
```

configure オプションの意味

`--disable-werror`

GCC に対して `-Werror` オプションを利用しないようにします。テストスイートを実行するために必要となります。

`--enable-kernel=3.2`

本オプションはビルドシステムに対して、カーネルバージョンが古くても 3.2 を用いることを指示します。これより古いバージョンにおけるシステムコールが用いられないようにするため、その回避策をとるものです。

`--enable-stack-protector=strong`

このオプション指定によりスタックに積まれる関数プロンプル内に、追加のコードを付与することにより、システムセキュリティを向上させます。その追加コードは、スタック破壊攻撃 (stack smashing attacks) のようなバッファオーバーフローをチェックします。

`--with-headers=/usr/include`

このオプションはビルドシステムにおいて、カーネル API ヘッダーを探す場所を指定します。

`libc_cv_slibdir=/usr/lib`

この変数によって、あらゆるシステムにおけるライブラリを正しく設定します。lib64 は利用しません。

パッケージをコンパイルします。

```
make
```

重要

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。

全般にテストの中には失敗するものがありますが、以下に示すものであれば無視しても構いません。

```
make check
```

テストに失敗する場合があります。これは Glibc のテストスイートがホストシステムにある程度依存しているためです。4200 を超えるテストの中で、ほんの少数のテストは失敗しますが、無視できるものです。LFS の当バージョンにおいて発生しがちな問題を以下に示します。

- `io/tst-lchmod` は LFS の `chroot` 環境においては失敗します。
- `misc/tst-ttyname` は LFS の `chroot` 環境においては失敗します。
- `nss/tst-nss-files-hosts-long` は失敗することがあります。これはシステム内にループバック以外の IP アドレスがない場合です。
- ホストのカーネルが比較的古い場合に `stdlib/tst-arc4random-thread` というテストが失敗します。
- `nss/tst-nss-files-hosts-multi` のようなテストでは、内部のタイムアウトが原因で比較的遅くなるシステム上では失敗します。

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
```

Makefile を修正して、不要な健全性チェックを無効にします。これは、この段階での LFS 環境では失敗するためです。

```
sed '/test-installation/s@$(PERL)@echo not running@' -i ../Makefile
```

パッケージをインストールします。

```
make install
```

`ldd` スクリプト内にある実行可能なローダーへのパスがハードコーディングされているので、これを修正します。

```
sed '/RTLDLIST=/s@/usr@@g' -i /usr/bin/ldd
```

`nscd` コマンドに対する設定ファイルや実行ディレクトリをインストールします。

```
cp -v ../nscd/nscd.conf /etc/nscd.conf  
mkdir -pv /var/cache/nscd
```

システムを各種の言語に対応させるためのロケールをインストールします。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が、将来的にはロケールが不足していることによって、重要なテストが実施されずに見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 2 つめの `localedef` では、キャラクターセットには依存しないロケール定義 `/usr/share/i18n/locales/cs_CZ` とキャラクターマップ定義 `/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 POSIX -f UTF-8 C.UTF-8 2> /dev/null || true
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 el_GR -f ISO-8859-7 el_GR
localedef -i en_GB -f ISO-8859-1 en_GB
localedef -i en_GB -f UTF-8 en_GB.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_ES -f ISO-8859-15 es_ES@euro
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 is_IS -f ISO-8859-1 is_IS
localedef -i is_IS -f UTF-8 is_IS.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i it_IT -f ISO-8859-15 it_IT@euro
localedef -i it_IT -f UTF-8 it_IT.UTF-8
localedef -i ja_JP -f EUC-JP ja_JP
localedef -i ja_JP -f SHIFT_JIS ja_JP.SJIS 2> /dev/null || true
localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
localedef -i nl_NL@euro -f ISO-8859-15 nl_NL@euro
localedef -i ru_RU -f KOI8-R ru_RU.KOI8-R
localedef -i ru_RU -f UTF-8 ru_RU.UTF-8
localedef -i se_NO -f UTF-8 se_NO.UTF-8
localedef -i ta_IN -f UTF-8 ta_IN.UTF-8
localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
localedef -i zh_HK -f BIG5-HKSCS zh_HK.BIG5-HKSCS
localedef -i zh_TW -f UTF-8 zh_TW.UTF-8
```

上に加えて、あなたの国、言語、キャラクター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-2.36/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-2.36/localedata/SUPPORTED` ファイルに示されていないロケールは `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。たとえば以下の 2 つのロケールは、本章で後に実施するテストにおいて必要になります。

```
localedef -i POSIX -f UTF-8 C.UTF-8 2> /dev/null || true
localedef -i ja_JP -f SHIFT_JIS ja_JP.SJIS 2> /dev/null || true
```



注記

現状の Glibc は、国際ドメイン名の解決に `libidn2` を利用します。これは実行時に依存するパッケージです。この機能が必要である場合は、BLFS にある `libidn2` ページに示されているインストール手順を参照してください。

8.5.2. Glibc の設定

8.5.2.1. nsswitch.conf の追加

/etc/nsswitch.conf ファイルを作成しておく必要があります。このファイルが無い場合、Glibc のデフォルト値ではネットワーク環境下にて Glibc が正しく動作しません。

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
```

8.5.2.2. タイムゾーンデータの追加

以下によりタイムゾーンデータをインストールし設定します。

```
tar -xvf ../../tzdata2022c.tar.gz

ZONEINFO=/usr/share/zoneinfo
mkdir -pv $ZONEINFO/{posix,right}

for tz in etcetera southamerica northamerica europe africa antarctica \
        asia australasia backward; do
    zic -L /dev/null -d $ZONEINFO ${tz}
    zic -L /dev/null -d $ZONEINFO/posix ${tz}
    zic -L leapseconds -d $ZONEINFO/right ${tz}
done

cp -v zone.tab zone1970.tab iso3166.tab $ZONEINFO
zic -d $ZONEINFO -p America/New_York
unset ZONEINFO
```

zic コマンドの意味

```
zic -L /dev/null ...
```

うるう秒を含まない posix タイムゾーンデータを生成します。これらは zoneinfo や zoneinfo/posix に収容するものとして適切なものです。zoneinfo へは POSIX 準拠のタイムゾーンデータを含めることが必要であり、こうしておかないと数々のテストスイートにてエラーが発生してしまいます。組み込みシステムなどでは容量の制約が厳しいため、タイムゾーンデータはあまり更新したくない場合があり、posix ディレクトリを設けなければ 1.9 MB もの容量を節約できます。ただしアプリケーションやテストスイートによっては、エラーが発生するかもしれません。

```
zic -L leapseconds ...
```

うるう秒を含んだ正しいタイムゾーンデータを生成します。組み込みシステムなどでは容量の制約が厳しいため、タイムゾーンデータはあまり更新したくない場合や、さほど気かけない場合もあります。right ディレクトリを省略することにすれば 1.9MB の容量を節約することができます。

```
zic ... -p ...
```

posixrules ファイルを生成します。ここでは New York を用います。POSIX では、日中の保存時刻として US ルールに従うことを規程しているためです。

ローカルなタイムゾーンの設定を行う1つの方法として、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。

```
ln -sfv /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。

8.5.2.3. ダイナミックローダー の設定

ダイナミックリンカー (/lib/ld-linux.so.2) がダイナミックライブラリを検索するデフォルトのディレクトリが /usr/lib ディレクトリです。各種プログラムが実行される際にはここから検索されたダイナミックライブラリがリンクされます。もし /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ー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ーの検索パスとして、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
EOF
```

必要がある場合には、ダイナミックローダーに対する設定として、他ディレクトリにて指定されるファイルをインクルードするようにもできます。通常は、そのファイル内の1行に、必要となるライブラリパスを記述します。このような設定を利用する場合には以下のようなコマンドを実行します。

```
cat >> /etc/ld.so.conf << "EOF"
# Add an include directory
include /etc/ld.so.conf.d/*.conf
EOF
mkdir -pv /etc/ld.so.conf.d
```

8.5.3. Glibc の構成

インストールプログラム:	gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, ld.so (ld-linux-x86-64.so.2 または ld-linux.so.2 へのリンク), locale, localedef, makedb, mtrace, nscd, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, zic
インストールライブラリ:	ld-linux-x86-64.so.2, ld-linux.so.2, libBrokenLocale.{a,so}, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libc_malloc_debug.so, libcrypt.{a,so}, libdl.{a,so.2}, libg.a, libm.{a,so}, libmcheck.a, libmemusage.so, libmvec.{a,so}, libnsl.so.1, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libpcprofile.so, libpthread.{a,so.0}, libresolv.{a,so}, librt.{a,so.1}, libthread_db.so, and libutil.{a,so.1}
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/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/il8n, /usr/share/zoneinfo, /var/cache/nscd, /var/lib/nss_db

概略説明

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	ダイナミックリンカーの実行時バインディングを設定します。
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。これは x86_64 のような最新アーキテクチャーには存在しません。
locale	現在のロケールに対するさまざま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。
makedb	テキストを入力として単純なデータベースを生成します。
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た情報をダンプします。
pldd	稼動中のプロセスにて利用されている動的共有オブジェクト (dynamic shared objects) を一覧出力します。
sln	スタティックなリンクを行う ln プログラム。
sotruss	指定されたコマンドの共有ライブラリ内のプロシジャーコールをトレースします。
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ー。
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-2.36/locale/broken_cur_max.c に書かれたコメントを参照してください。
libanl	非同期の名前解決 (asynchronous name lookup) ライブラリ。
libc	主要な C ライブラリ。
libc_malloc_debug	プリロード時のメモリ割り当てチェックをオンにします。
libcrypt	暗号化ライブラリ。
libdl	何の関数も含まないダミーライブラリ。以前はダイナミックリンクインターフェースライブラリでしたが、現在その関数は libc に含まれるようになりました。
libg	何の関数も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。
libm	数学ライブラリ。
libmvec	ベクトル数学ライブラリ。libm が用いられる際に必要となるためリンクされます。
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。現在は非推奨。
libnss_*	NSS (Name Service Switch) モジュール。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。/etc/nsswitch.conf での設定に従って libc によりロードされます。
libpcprofile	PC プロファイルにたいして実行モジュールをプリロードするために用いられます。
libpthread	関数を全く含まないダミーのライブラリ。かつては POSIX.1b Realtime Extension によって規定されているインターフェースをほとんど含めた関数を提供していました。現在その関数は libc に含まれるようになりました。

<code>libresolv</code>	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。
<code>librt</code>	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。
<code>libthread_db</code>	マルチスレッドプログラム用のデバッガーを構築するための有用な関数を提供します。
<code>libutil</code>	何の関数も含まないダミーライブラリ。以前は、さまざまな Unix ユーティリティーに用いられる「標準的な」関数のコードを含んでいました。現在その関数は <code>libc</code> に含まれるようになりました。

8.6. Zlib-1.2.12

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: 6.1 MB

8.6.1. Zlib のインストール

Zlib 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
```

不要なスタティックライブラリを削除します。

```
rm -fv /usr/lib/libz.a
```

8.6.2. Zlib の構成

インストールライブラリ: libz.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。

8.7. Bzip2-1.0.8

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: less than 0.1 SBU
必要ディスク容量: 7.2 MB

8.7.1. Bzip2 のインストール

本パッケージのドキュメントをインストールするためにパッチを適用します。

```
patch -Np1 -i ../bzip2-1.0.8-install_docs-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
```

man ページのインストール先を正しいディレクトリに修正します。

```
sed -i "s@(PREFIX)/man@(PREFIX)/share/man@g" 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 パラメーターの意味

`-f Makefile-libbz2_so`

このパラメーターは 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 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
```

共有ライブラリをインストールします。

```
cp -av libbz2.so.* /usr/lib
ln -sv libbz2.so.1.0.8 /usr/lib/libbz2.so
```

共有化された bzip2 実行モジュールを /usr/bin ディレクトリにインストールします。またシンボリックリンクにより bzip2 のコピーを 2 つ作ります。

```
cp -v bzip2-shared /usr/bin/bzip2
for i in /usr/bin/{bzcat,bunzip2}; do
  ln -sfv bzip2 $i
done
```

不要なスタティックライブラリを削除します。

```
rm -fv /usr/lib/libbz2.a
```

8.7.2. 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 (bzdifff へのリンク), bzdifff, bzegrep (bzgrep へのリンク), bzfgrep (bzgrep へのリンク), bzgrep, bzip2, bzip2recover, bzless (bzmor へのリンク), bzmor
インストールライブラリ: libbz2.so
インストールディレクトリ: /usr/share/doc/bzip2-1.0.8

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。

8.8. Xz-5.2.6

Xz 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 lzma フォーマットおよび新しい xz 圧縮フォーマットを取り扱います。xz コマンドによりテキストファイルを圧縮すると、従来の gzip コマンドや bzip2 コマンドに比べて、高い圧縮率を実現できます。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 16 MB

8.8.1. Xz のインストール

Xz をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-static \
            --docdir=/usr/share/doc/xz-5.2.6
```

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場合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
```

8.8.2. Xz の構成

インストールプログラム: lzcat (xz へのリンク), lzcmp (xzdifff へのリンク), lzdifff (xzdifff へのリンク), lzgrep (xzgrep へのリンク), lzfgrep (xzgrep へのリンク), lzgrep (xzgrep へのリンク), lzless (xzless へのリンク), lzma (xz へのリンク), lzmadec, lzmainfo, lzmore (xzmore へのリンク), unlzma (xz へのリンク), unxz (xz へのリンク), xz, xzcat (xz へのリンク), xzcmp (xzdifff へのリンク), xzdec, xzdifff, xzgrep (xzgrep へのリンク), xzfgrep (xzgrep へのリンク), xzgrep, xzless, xzmore

インストールライブラリ: liblzma.so

インストールディレクトリ: /usr/include/lzma, /usr/share/doc/xz-5.2.6

概略説明

lzcat ファイルを伸張（解凍）し標準出力へ出力します。

lzcmp LZMA 圧縮されたファイルに対して cmp を実行します。

lzdifff LZMA 圧縮されたファイルに対して diff を実行します。

lzgrep LZMA 圧縮されたファイルに対して egrep を実行します。

lzfgrep LZMA 圧縮されたファイルに対して fgrep を実行します。

lzgrep LZMA 圧縮されたファイルに対して grep を実行します。

lzless LZMA 圧縮されたファイルに対して less を実行します。

lzma LZMA フォーマットによりファイルの圧縮と伸張（解凍）を行います。

lzmadec LZMA 圧縮されたファイルを高速に伸張（解凍）するコンパクトなプログラムです。

lzmainfo LZMA 圧縮されたファイルのヘッダーに保持されている情報を表示します。

lzmore LZMA 圧縮されたファイルに対して more を実行します。

unlzma LZMA フォーマットされたファイルを伸張（解凍）します。

unxz XZ フォーマットされたファイルを伸張（解凍）します。

xz XZ フォーマットによりファイルの圧縮と伸張（解凍）を行います。

xzcat ファイルの伸張（解凍）を行い標準出力へ出力します。

xzcmp XZ 圧縮されたファイルに対して cmp を実行します。

xzdec XZ 圧縮されたファイルを高速に伸張（解凍）するコンパクトなプログラムです。

xzdiff	XZ 圧縮されたファイルに対して diff を実行します。
xzegrep	XZ 圧縮されたファイルに対して egrep を実行します。
xzfgrep	XZ 圧縮されたファイルに対して fgrep を実行します。
xzgrep	XZ 圧縮されたファイルに対して grep を実行します。
xzless	XZ 圧縮されたファイルに対して less を実行します。
xzmore	XZ 圧縮されたファイルに対して more を実行します。
liblzma	Lempel-Ziv-Markov のチェーンアルゴリズムを利用し、損失なくブロックソートによりデータ圧縮を行う機能を提供するライブラリです。

8.9. Zstd-1.5.2

Zstandard とはリアルタイムの圧縮アルゴリズムのことであり、高圧縮率を実現します。圧縮、処理速度間のトレードオフを広範囲に提供するとともに、高速な伸張（解凍）処理を実現します。

概算ビルド時間: 1.1 SBU
必要ディスク容量: 56 MB

8.9.1. Zstd のインストール

アップストリームが認識している問題修正のためのパッチを適用します。

```
patch -Np1 -i ../zstd-1.5.2-upstream_fixes-1.patch
```

パッケージをコンパイルします。

```
make prefix=/usr
```



注記

テスト結果の出力の中に 'failed' と示される箇所があります。これは実際のテストが失敗したときだけ 'FAIL' と出力されるものです。したがってテスト失敗ではありません。

ビルド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 prefix=/usr install
```

スタティックライブラリを削除します。

```
rm -v /usr/lib/libzstd.a
```

8.9.2. Zstd の構成

インストールプログラム: zstd, zstdcat (zstd へのリンク), zstdgrep, zstdless, zstdmt (zstd へのリンク), unzstd (zstd へのリンク)
インストールライブラリ: libzstd.so

概略説明

zstd ZSTD 形式によりファイルを圧縮、伸張（解凍）します。
zstdgrep ZSTD 圧縮ファイルに対して grep を実行します。
zstdless ZSTD 圧縮ファイルに対して less を実行します。
libzstd ZSTD アルゴリズムを利用した可逆データ圧縮を実装するライブラリ。

8.10. File-5.42

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: 0.1 SBU
必要ディスク容量: 16 MB

8.10.1. 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
```

8.10.2. 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.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。

8.11. Readline-8.1.2

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: 0.1 SBU
必要ディスク容量: 15 MB

8.11.1. 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 をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-static \
            --with-curses \
            --docdir=/usr/share/doc/readline-8.1.2
```

configure オプションの意味

`--with-curses`

このオプションは Readline パッケージに対して、termcap ライブラリ関数の探し場所を、切り離されている termcap ライブラリではなく curses ライブラリとすることを指示します。これにより readline.pc ファイルが適切に生成されます。

パッケージをコンパイルします。

```
make SHLIB_LIBS="-lncursesw"
```

make オプションの意味

`SHLIB_LIBS="-lncursesw"`

このオプションにより Readline を 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 SHLIB_LIBS="-lncursesw" install
```

必要ならドキュメントをインストールします。

```
install -v -m644 doc/*.{ps,pdf,html,dvi} /usr/share/doc/readline-8.1.2
```

8.11.2. Readline の構成

インストールライブラリ: libhistory.so, libreadline.so
インストールディレクトリ: /usr/include/readline, /usr/share/doc/readline-8.1.2

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 プログラムの対話セッションから入力されるテキストを処理するための一連のコマンドを提供します。

8.12. M4-1.4.19

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: 0.6 SBU
必要ディスク容量: 49 MB

8.12.1. M4 のインストール

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
```

ビルド結果をテストする場合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
```

8.12.2. 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ー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。

8.13. Bc-6.0.1

Bc パッケージは、任意精度 (arbitrary precision) の演算処理言語を提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 7.4 MB

8.13.1. Bc のインストール

Bc をコンパイルするための準備をします。

```
CC=gcc ./configure --prefix=/usr -G -O3 -r
```

configure オプションの意味

`CC=gcc`

このパラメーターはコンパイラーを指定します。

`-G`

GNU bc が存在していない状態では動作しないテストスイートを省略します。

`-O3`

利用する最適化を指定します。

`-r`

bc における行編集機能を拡張するために Readline 利用を有効にします。

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場合は、以下を実行します。

```
make test
```

パッケージをインストールします。

```
make install
```

8.13.2. Bc の構成

インストールプログラム: bc, dc

概略説明

bc コマンドラインから実行する計算機 (calculator)。

dc 逆ポーランド (reverse-polish) 記法による計算機。

8.14. Flex-2.6.4

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: 0.4 SBU
必要ディスク容量: 33 MB

8.14.1. Flex のインストール

Flex をコンパイルするための準備をします。

```
./configure --prefix=/usr \  
            --docdir=/usr/share/doc/flex-2.6.4 \  
            --disable-static
```

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
```

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 という名のシンボリックリンクを生成します。このリンクが lex のエミュレーションモードとして flex を呼び出します。

```
ln -sv flex /usr/bin/lex
```

8.14.2. Flex の構成

インストールプログラム: flex, flex++ (flex へのリンク), lex (flex へのリンク)
インストールライブラリ: libfl.so
インストールディレクトリ: /usr/share/doc/flex-2.6.4

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。

flex++ flex の拡張。C++ コードやクラスの生成に利用されます。これは flex へのシンボリックリンクです。

lex lex のエミュレーションモードとして flex を実行するシンボリックリンク。

libfl flex ライブラリ。

8.15. Tcl-8.6.12

Tcl パッケージは、堅牢で汎用的なスクリプト言語であるツールコマンド言語 (Tool Command Language) を提供します。Expect パッケージは Tcl 言語によって書かれています。

概算ビルド時間: 3.2 SBU
必要ディスク容量: 88 MB

8.15.1. Tcl のインストール

本パッケージとこれに続く 2 つのパッケージ (Expect と DejaGNU) は、Binutils および GCC など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 3 つのパッケージをインストールするというのは、少々大げさなこともかもしれません。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。

はじめにドキュメントを伸張 (解凍) する以下のコマンドを実行します。

```
tar -xzf ../tcl8.6.12-html.tar.gz --strip-components=1
```

Tcl をコンパイルするための準備をします。

```
SRCDIR=$(pwd)
cd unix
./configure --prefix=/usr \
            --mandir=/usr/share/man
```

パッケージをビルドします。

```
make

sed -e "s|${SRCDIR}/unix|/usr/lib|" \
    -e "s|${SRCDIR}|/usr/include|" \
    -i tclConfig.sh

sed -e "s|${SRCDIR}/unix/pkgsrc/tdbc1.1.3|/usr/lib/tdbc1.1.3|" \
    -e "s|${SRCDIR}/pkgsrc/tdbc1.1.3/generic|/usr/include|" \
    -e "s|${SRCDIR}/pkgsrc/tdbc1.1.3/library|/usr/lib/tcl8.6|" \
    -e "s|${SRCDIR}/pkgsrc/tdbc1.1.3|/usr/include|" \
    -i pkgsrc/tdbc1.1.3/tdbcConfig.sh

sed -e "s|${SRCDIR}/unix/pkgsrc/itcl4.2.2|/usr/lib/itcl4.2.2|" \
    -e "s|${SRCDIR}/pkgsrc/itcl4.2.2/generic|/usr/include|" \
    -e "s|${SRCDIR}/pkgsrc/itcl4.2.2|/usr/include|" \
    -i pkgsrc/itcl4.2.2/itclConfig.sh

unset SRCDIR
```

"make" コマンドに続くたくさんの "sed" コマンドは、設定ファイルにあるビルドディレクトリへの参照を削除して、インストールディレクトリへの参照に置き換えます。これ以降の LFS 作業において必須のことではありませんが、後にビルドされるパッケージが Tcl を用いるかもしれないからです。

ビルド結果をテストする場合は、以下を実行します。

```
make test
```

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なります。

```
chmod -v u+w /usr/lib/libtcl8.6.so
```

Tcl のヘッダー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 -sfv tclsh8.6 /usr/bin/tclsh
```

Perl の man ページと重複するものを名称変更します。

```
mv /usr/share/man/man3/{Thread,Tcl_Thread}.3
```

任意のドキュメントをダウンロードしている場合は、以下のコマンドを実行してインストールします。

```
mkdir -v -p /usr/share/doc/tcl-8.6.12
cp -v -r ../html/* /usr/share/doc/tcl-8.6.12
```

8.15.2. Tcl の構成

インストールプログラム: tclsh (tclsh8.6 へのリンク), tclsh8.6
 インストールライブラリ: libtcl8.6.so, libtclstub8.6.a

概略説明

tclsh8.6	Tcl コマンドシェル
tclsh	tclsh8.6 へのリンク
libtcl8.6.so	Tcl ライブラリ
libtclstub8.6.a	Tcl スタブライブラリ

8.16. Expect-5.45.4

Expect パッケージには telnet, ftp, passwd, fsck, rlogin, tip といった対話処理ツールを、スクリプト化されたダイアログを通じて自動化するツールを提供します。Expect はこういったアプリケーションをテストする場合にも利用できます。また本パッケージを利用しないと相当に困難となるようなタスクを、いとも簡単に処理できるようになります。

DejaGnu フレームワークはこの Expect を用いて記述されています。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 3.9 MB

8.16.1. Expect のインストール

Expect をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --with-tcl=/usr/lib \
            --enable-shared \
            --mandir=/usr/share/man \
            --with-tclinclude=/usr/include
```

configure オプションの意味

`--with-tcl=/usr/lib`

本パラメーターは configure に対して、tclConfig.sh スクリプトが存在するディレクトリを指示するために必要となります。

`--with-tclinclude=/usr/include`

Tcl の内部ヘッダーファイルを探し出す場所を指定します。

パッケージをビルドします。

```
make
```

ビルド結果をテストする場合は、以下を実行します。

```
make test
```

パッケージをインストールします。

```
make install
ln -svf expect5.45.4/libexpect5.45.4.so /usr/lib
```

8.16.2. 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: libexpect5.45.4.so

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-5.45.4.so 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。

8.17. DejaGNU-1.6.3

DejaGnu パッケージは、GNU ツールに対してテストスイートを実行するフレームワークを提供します。これは expect によって書かれており、expect そのものは Tcl (ツールコマンド言語) を利用しています。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 6.9 MB

8.17.1. DejaGNU のインストール

アップストリームは、専用のビルドディレクトリを作成して DejaGNU をビルドすることを推奨しています。

```
mkdir -v build
cd      build
```

DejaGNU をコンパイルするための準備をします。

```
../configure --prefix=/usr
makeinfo --html --no-split -o doc/dejagnum.html ../doc/dejagnum.texi
makeinfo --plaintext      -o doc/dejagnum.txt  ../doc/dejagnum.texi
```

パッケージをビルドしてインストールします。

```
make install
install -v -dm755 /usr/share/doc/dejagnum-1.6.3
install -v -m644 doc/dejagnum.{html,txt} /usr/share/doc/dejagnum-1.6.3
```

コンパイル結果をテストするならば以下を実行します。

```
make check
```

8.17.2. DejaGNU の構成

インストールプログラム: dejagnum, runtest

概略説明

dejagnum DejaGNU の補助コマンドローンチャー。

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。

8.18. Binutils-2.39

Binutils パッケージは、リンカーやアセンブラー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: 8.2 SBU
必要ディスク容量: 2.7 GB

8.18.1. 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が。

```
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.
```

Binutils のドキュメントによると Binutils のビルドにあたっては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 build
cd      build
```

Binutils をコンパイルするための準備をします。

```
../configure --prefix=/usr      \
              --sysconfdir=/etc  \
              --enable-gold      \
              --enable-ld=default \
              --enable-plugins   \
              --enable-shared    \
              --disable-werror   \
              --enable-64-bit-bfd \
              --with-system-zlib
```

configure パラメーターの意味

`--enable-gold`

ゴールドリンカー (gold linker) をビルドし ld.gold としてインストールします。

`--enable-ld=default`

オリジナルの bfd リンカーをビルドし ld (デフォルトリンカー) と ld.bfd としてインストールします。

`--enable-plugins`

リンカーに対してプラグインサポートを有効にします。

`--enable-64-bit-bfd`

64 ビットサポート (ホスト上でのワードサイズの縮小) を有効にします。64 ビットシステムでも不要な場合がありますが、指定しておいて支障はありません。

`--with-system-zlib`

本パッケージに含まれる zlib をビルドするのではなく、既にインストール済の zlib を用いるようにします。

パッケージをコンパイルします。

```
make tooldir=/usr
```

make パラメーターの意味

`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-pc-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

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 -k check
```

パッケージをインストールします。

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

不要なスタティックライブラリを削除します。

```
rm -fv /usr/lib/lib{bfd,ctf,ctf-nobfd,opcodes}.a
```

8.18.2. Binutils の構成

インストールプログラム:	addr2line, ar, as, c++filt, dwp, elfedit, gprof, gprofng, ld, ld.bfd, ld.gold, nm, objcopy, objdump, ranlib, readelf, size, strings, strip
インストールライブラリ:	libbfd.so, libctf.so, libctf-nobfd.so, libopcodes.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ー。
c++filt	リンカー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。
dwp	DWARF パッケージユーティリティ。
elfedit	ELF ファイルの ELF ヘッダーを更新します。
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。
gprofng	性能データの収集と解析を行います。
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ー。データの再配置やシンボル参照情報の結合を行います。
ld.gold	elf オブジェクト向けファイルフォーマットのサポートにのみ特化した ld の限定バージョン。
ld.bfd	ld へのハードリンク。
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。さまざま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ー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ー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。
- libbfd バイナリファイルディスクリプター (Binary File Descriptor) ライブラリ。
- libctf Compat ANSI-C Type フォーマットタイプデバッグサポートライブラリ。
- libctf-nobfd libbfd の機能を利用しない libctf の互換ライブラリ。
- 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 のような、ビルド作業に用いるユーティリティプログラムが利用しています。

8.19. GMP-6.2.1

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: 0.9 SBU
必要ディスク容量: 53 MB

8.19.1. 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 ...
```



注記

GMP のデフォルト設定に従うと、ホストのプロセッサ向けに最適化したライブラリを生成してしまいます。ホストに比べて、やや性能の劣るプロセッサに向けたライブラリを必要とする場合は、汎用ライブラリを生成するために以下を実行します。

```
cp -v configfsf.guess config.guess
cp -v configfsf.sub config.sub
```

GMP をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --enable-cxx \
            --disable-static \
            --docdir=/usr/share/doc/gmp-6.2.1
```

configure オプションの意味

`--enable-cxx`

C++ サポートを有効にします。

`--docdir=/usr/share/doc/gmp-6.2.1`

ドキュメントのインストール先を適切に設定します。

パッケージをコンパイルし HTML ドキュメントを生成します。

```
make
make html
```



重要

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
```



注意

gmp のコードはビルドするプロセッサ向けに高度に最適化されます。このためプロセッサを特定したコードが実はシステム性能を的確に制御できないことも起こりえます。それはテストにおいてエラーを引き起こしたり、gmp を利用する他のアプリケーションにおいて "Illegal instruction" というエラーとして発生したりすることがあります。そういった場合は gmp の再ビルドが必要であり、その際にはオプション `--build=x86_64-pc-linux-gnu` をつける必要があります。

197 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 '/# PASS:/{total+=$3} ; END{print total}' gmp-check-log
```

パッケージと HTML ドキュメントをインストールします。

```
make install
make install-html
```

8.19.2. GMP の構成

インストールライブラリ: libgmp.so, libgmpxx.so
インストールディレクトリ: /usr/share/doc/gmp-6.2.1

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。

8.20. MPFR-4.1.0

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: 0.8 SBU
必要ディスク容量: 39 MB

8.20.1. 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 \
            --disable-static \
            --enable-thread-safe \
            --docdir=/usr/share/doc/mpfr-4.1.0
```

パッケージをコンパイルし HTML ドキュメントを生成します。

```
make
make html
```



重要

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
```

パッケージとドキュメントをインストールします。

```
make install
make install-html
```

8.20.2. MPFR の構成

インストールライブラリ: libmpfr.so
インストールディレクトリ: /usr/share/doc/mpfr-4.1.0

概略説明

libmpfr 倍精度演算の関数を提供します。

8.21. MPC-1.2.1

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: 21 MB

8.21.1. 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 \
            --disable-static \
            --docdir=/usr/share/doc/mpc-1.2.1
```

パッケージをコンパイルし HTML ドキュメントを生成します。

```
make
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
```

パッケージとドキュメントをインストールします。

```
make install
make install-html
```

8.21.2. MPC の構成

インストールライブラリ: libmpc.so
インストールディレクトリ: /usr/share/doc/mpc-1.2.1

概略説明

libmpc 複素数による演算関数を提供します。

8.22. Attr-2.5.1

attr パッケージは、ファイルシステム上のオブジェクトに対しての拡張属性を管理する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: 4.1 MB

8.22.1. Attr のインストール

Attr をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-static \
            --sysconfdir=/etc \
            --docdir=/usr/share/doc/attr-2.5.1
```

パッケージをコンパイルします。

```
make
```

テストは、ext2, ext3, ext4 のような拡張属性をサポートしているファイルシステム上にて実施する必要があります。テストを実施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
```

8.22.2. Attr の構成

インストールプログラム: attr, getfattr, setfattr
インストールライブラリ: libattr.so
インストールディレクトリ: /usr/include/attr, /usr/share/doc/attr-2.5.1

概略説明

attr ファイルシステム上のオブジェクトに対して、属性を拡張します。
getfattr ファイルシステム上のオブジェクトに対して、拡張属性の情報を取得します。
setfattr ファイルシステム上のオブジェクトに対して、拡張属性の情報を設定します。
libattr 拡張属性を制御するライブラリ関数を提供します。

8.23. Acl-2.3.1

Acl パッケージは、アクセスコントロールリスト (Access Control Lists) を管理するユーティリティーを提供します。これはファイルやディレクトリに対して、きめ細かく詳細にアクセス権限を設定するものとして利用されます。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 6.1 MB

8.23.1. Acl のインストール

Acl をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-static \
            --docdir=/usr/share/doc/acl-2.3.1
```

パッケージをコンパイルします。

```
make
```

Acl のテストは、Acl のライブラリによって Coreutils をビルドした後に、アクセス制御がサポートされたファイルシステム上にて実施する必要があります。テスト実施が必要である場合は、後に生成する Coreutils のビルドが終わってから、再び本パッケージに戻って `make check` 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 install
```

8.23.2. Acl の構成

インストールプログラム: chacl, getfacl, setacl
インストールライブラリ: libacl.so
インストールディレクトリ: /usr/include/acl, /usr/share/doc/acl-2.3.1

概略説明

chacl ファイルまたはディレクトリに対するアクセスコントロールリストを設定します。
getfacl ファイルアクセスコントロールリストを取得します。
setfacl ファイルアクセスコントロールリストを設定します。
libacl アクセスコントロールリスト (Access Control Lists) を制御するライブラリ関数を提供します。

8.24. Libcap-2.65

Libcap パッケージは、Linux カーネルにおいて利用される POSIX 1003.1e 機能へのユーザー空間からのインターフェースを実装します。この機能は、強力な root 権限機能を他の権限へと分散します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.7 MB

8.24.1. Libcap のインストール

スタティックライブラリをインストールしないようにします。

```
sed -i '/install -m.*STA/d' libcap/Makefile
```

パッケージをコンパイルします。

```
make prefix=/usr lib=lib
```

make オプションの意味

lib=lib

このパラメーターは x86_64 においてライブラリを /usr/lib64 ではなく /usr/lib にインストールするようにします。x86 においては何も効果はありません。

ビルド結果をテストする場合は以下を実行します。

```
make test
```

パッケージをインストールします。

```
make prefix=/usr lib=lib install
```

8.24.2. Libcap の構成

インストールプログラム: capsh, getcap, getpcaps, setcap
インストールライブラリ: libcap.so, libpsx.so

概略説明

capsh	拡張属性サポートについて制御するためのシェルラッパー。
getcap	ファイルの拡張属性を検査します。
getpcaps	指定されたプロセスの拡張属性を表示します。
setcap	ファイルの拡張属性を設定します。
libcap	POSIX 1003.1e 拡張属性を制御するライブラリ関数を提供します。
libpsx	pthread ライブラリに関する syscalls に対する POSIX セマンティックス対応の関数を提供します。

8.25. Shadow-4.12.2

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: 0.2 SBU
必要ディスク容量: 46 MB

8.25.1. 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 <https://www.linuxfromscratch.org/blfs/view/11.2/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` パラメーター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。また「Man-pages-5.13」にてインストールされている man ページはインストールしないようにします。

```
sed -i 's/groups$(EXEEXT) //' src/Makefile.in
find man -name Makefile.in -exec sed -i 's/groups\.1 / /' {} \;
find man -name Makefile.in -exec sed -i 's/getspnam\.3 / /' {} \;
find man -name Makefile.in -exec sed -i 's/passwd\.5 / /' {} \;
```

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 SHA-512 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。また `PATH` から `/bin` と `/sbin` を削除します。これらは `/usr` からのシンボリックリンクであるからです。



注記

何らかの理由により `PATH` に対して `/bin` や `/sbin` を残しておく必要がある場合は、LFS ビルドが完了した後に `.bashrc` において `PATH` を設定してください。

```
sed -e 's:#ENCRYPT_METHOD DES:ENCRYPT_METHOD SHA512:' \
-e 's:/var/spool/mail:/var/mail:' \
-e '/PATH={s@/sbin:@@;s@/bin:@@}' \
-i 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。

```
touch /usr/bin/passwd
./configure --sysconfdir=/etc \
--disable-static \
--with-group-name-max-length=32
```

`configure` オプションの意味

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

プログラムの中には `/usr/bin/passwd` のパスがそのままハードコーディングされているものがあり、このファイルが存在しなかった場合のデフォルトパスが正しくありません。

```
--with-group-name-max-length=32
```

ユーザー名とグループ名の最大文字数を 32 とします。

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 exec_prefix=/usr install
make -C man install-man
```

8.25.2. Shadow の設定

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ー、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` コマンドに対するデフォルトの設定には、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となります。この動作が不適当であれば `useradd` コマンドの実行時に `-g` パラメーターか `-N` のいずれかを利用することが必要です。あるいは `/etc/login.defs` 内にある `USERGROUPS_ENAB` の設定を書き換えてください。詳しくは `useradd(8)` を参照してください。

次にデフォルトパラメーターを変更します。そのためにはファイル `/etc/default/useradd` の生成が必要です。特定の状況に合わせてこれを設定します。まずは以下のようにして、このファイルを生成します。

```
mkdir -p /etc/default
useradd -D --gid 999
```

`/etc/default/useradd` のパラメーター説明

`GROUP=999`

このパラメーターは `/etc/group` ファイルにおいて設定されるグループ ID の先頭番号を指定します。999 という値は、上に示した `--gid` からきています。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。このパラメーターによって指定された数値が実際に利用されていた場合、その値以降で利用可能な値が採用されます。また `useradd` コマンドの実行にあたってパラメーター `-g` を利用せずに、その数値によって表される ID を持ったグループがシステム上に存在しなかった場合は、以下のようなメッセージが出力されます。 `useradd: unknown GID 999 ("GID 999 が不明です")` この場合でも、アカウントは正しく生成されます。だからこそ「重要なファイルとシンボリックリンクの生成」において、グループ ID を指定してグループ `users` を生成できたわけです。

`CREATE_MAIL_SPOOL=yes`

このパラメーターは `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 '/MAIL/s/yes/no/' /etc/default/useradd
```

8.25.3. 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
```

8.25.4. Shadow の構成

インストールプログラム:	chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, getsubids, gpasswd, groupadd, groupdel, groupmems, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgidmap, newgrp, newuidmap, 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
インストールディレクトリ:	libsubid.so

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。
getsubids	ユーザーのサブ id 範囲の一覧取得に用いられます。
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。
newgidmap	ユーザー空間における gid マッピングを設定します。
newgrp	ログインセッション中に現在の GID を変更します。
newuidmap	ユーザー空間における uid マッピングを設定します。
newusers	複数ユーザーのアカウント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	指定した名前で作成した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。
libsubid ユーザーに対するサブ ID 範囲を処理するライブラリ。

8.26. GCC-12.2.0

GCC パッケージは C コンパイラーや C++ コンパイラーなどの GNU コンパイラーコレクションを提供します。

概算ビルド時間: 160 SBU (テスト込み)
必要ディスク容量: 5.1 GB

8.26.1. GCC のインストール

x86_64 上でビルドしている場合は、64ビットライブラリのデフォルトディレクトリ名を "lib" にします。

```
case $(uname -m) in
x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
;;
esac
```

GCC のドキュメントによると GCC のビルドにあたっては、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 build
cd      build
```

GCC をコンパイルするための準備をします。

```
../configure --prefix=/usr \
             LD=ld \
             --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 パラメーターの意味

LD=ld

本パラメーターは、本章の初期段階でビルドした binutils の ld を使うことを configure スクリプトに指示します。これを指定しなかった場合は、クロスビルド版のものが用いられることになります。

--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

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。ただし相当な時間を要します。初めてビルドを行う方には、必ず実施することをお勧めします。テスト実行に要する時間は、make コマンドに *-jx* をつけることで、かなり削減できます。ここに示す *x* には、システムのコア数を指定するものです。

GCC テストスイートの中で、デフォルトの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 32768
```

一般ユーザーにてテストを行います。ただしエラーがあっても停止しないようにします。

```
chown -Rv tester .
su tester -c "PATH=$PATH make -k check"
```

テスト結果を確認するために以下を実行します。

```
../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 <https://www.linuxfromscratch.org/lfs/build-logs/11.2/> と <https://gcc.gnu.org/ml/gcc-testresults/> にある情報と比較することができます。

g++ においては PR100400 に関連するテスト 4 つが XPASS および FAIL として出力されます。この問題はテストファイルが適切に記述されていないために発生します。

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。上記の URL に示されている結果と大きく異ならなかったら、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
```

GCC のビルドディレクトリの所有者は `tester` であるため、ヘッダーがインストールされるディレクトリ（とその内容）に対する所有権が不適切なものになります。そこでその所有権を `root` ユーザーとグループに変更します。

```
chown -v -R root:root \  
/usr/lib/gcc/$gcc -dumpmachine)/12.2.0/include{,-fixed}
```

FHS の求めるところに応じてシンボリックリンクを作成します。これは慣例によるものです

```
ln -svr /usr/bin/cpp /usr/lib
```

リンク時の最適化 (Link Time Optimization; LTO) によりプログラム構築できるように、シンボリックリンクを作ります。

```
ln -sfv ../../libexec/gcc/$gcc -dumpmachine)/12.2.0/liblto_plugin.so \  
/usr/lib/bfd-plugins/
```

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。そこで健全性テストをここで実施します。

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

問題なく動作するはずで、最後のコマンドから出力される結果は以下になるはずです。（ダイナミックリンカー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: /lib64/ld-linux-x86-64.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[lin].*succeeded' dummy.log
```

上のコマンドの出力は以下になるはずです。

```
/usr/lib/gcc/x86_64-pc-linux-gnu/12.2.0/../../../../lib/crt1.o succeeded  
/usr/lib/gcc/x86_64-pc-linux-gnu/12.2.0/../../../../lib/crti.o succeeded  
/usr/lib/gcc/x86_64-pc-linux-gnu/12.2.0/../../../../lib/crtn.o succeeded
```

作業しているマシンアーキテクチャーによっては、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。注意すべき重要な点は `gcc` が `crt*.o` という 3 つのファイルを `/usr/lib` 配下から探し出しているということです。

コンパイラーが正しいヘッダー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
```

上のコマンドは以下の出力を返します。

```
#include <...> search starts here:  
/usr/lib/gcc/x86_64-pc-linux-gnu/12.2.0/include  
/usr/local/include  
/usr/lib/gcc/x86_64-pc-linux-gnu/12.2.0/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 アーキテクチャーにより異なる点に注意してください。

次に、新たなリンカー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'
```

'-linux-gnu' を含んだパスは無視すれば、最後のコマンドの出力は以下となるはずです。

```
SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib64")
SEARCH_DIR("/usr/local/lib64")
SEARCH_DIR("/lib64")
SEARCH_DIR("/usr/lib64")
SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

32ビットシステムではディレクトリが多少異なります。以下は i686 マシンでの出力例です。

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib32")
SEARCH_DIR("/usr/local/lib32")
SEARCH_DIR("/lib32")
SEARCH_DIR("/usr/lib32")
SEARCH_DIR("/usr/i686-pc-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
```

最後のコマンドの出力は以下のようになるはずです。

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

GCC が正しくダイナミックリンカーを用いているかを確認します。

```
grep found dummy.log
```

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ー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。)

```
found ld-linux-x86-64.so.2 at /usr/lib/ld-linux-x86-64.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。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
```

最後に誤ったディレクトリにあるファイルを移動します。

```
mkdir -pv /usr/share/gdb/auto-load/usr/lib
mv -v /usr/lib/*gdb.py /usr/share/gdb/auto-load/usr/lib
```

8.26.2. 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, gcc-ar, gcc-nm, gcc-ranlib, gcov, gcov-dump, gcov-tool, lto-dump
インストールライブラリ:	libasan.{a,so}, libatomic.{a,so}, libecl.so, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libitm.{a,so}, liblsan.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.a, libstdc++fs.a, libsupc++.a, libtsan.{a,so}, libubsan.{a,so}
インストールディレクトリ:	/usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc, /usr/share/gcc-12.2.0

概略説明

c++	C++ コンパイラー
cc	C コンパイラー
cpp	C プリプロセッサ。コンパイラーがこれを利用して、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++ コンパイラー
gcc	C コンパイラー
gcc-ar	ar に関連するラッパーであり、コマンドラインへのプラグインを追加します。このプログラムは「リンク時の最適化 (link time optimization)」機能を付与する場合にのみ利用されます。デフォルトのビルドオプションでは有効にはなりません。
gcc-nm	nm に関連するラッパーであり、コマンドラインへのプラグインを追加します。このプログラムは「リンク時の最適化 (link time optimization)」機能を付与する場合にのみ利用されます。デフォルトのビルドオプションでは有効にはなりません。
gcc-ranlib	ranlib に関連するラッパーであり、コマンドラインへのプラグインを追加します。このプログラムは「リンク時の最適化 (link time optimization)」機能を付与する場合にのみ利用されます。デフォルトのビルドオプションでは有効にはなりません。
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。
gcov-dump	オフラインの gcda および gcno プロファイルダンプツール。
gcov-tool	オフラインの gcda プロファイル処理ツール。
lto-dump	LTO が有効にした GCC によって生成されるオブジェクトファイルをダンプするためのツール。
libasan	アドレスサニタイザー (Address Sanitizer) のランタイムライブラリ。
libatomic	GCC 不可分 (アトミック) ビルトインランタイムライブラリ。
libgcc1	C 言語プリプロセッサライブラリ。
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。
libitm	GNU のトランザクショナル (transactional) メモリーライブラリ。
liblsan	リークサニタイザー (Leak Sanitizer) のランタイムライブラリ。
liblto_plugin	GCC の LTO プラグインは、LTO を有効にした GCC から生成されたオブジェクトファイルを binutils が処理できるようにします。
libquadmath	GCC の4倍精度数値演算 (Quad Precision Math) ライブラリ API
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
libstdc++fs	ISO/IEC TS 18822:2015 ファイルシステムライブラリ。
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。
libtsan	スレッドサニタイザー (Thread Sanitizer) のランタイムライブラリ。
libubsan	Undefined Behavior Sanitizer ランタイムライブラリ。

8.27. Pkg-config-0.29.2

pkg-config パッケージは configure や make による処理において、インクルードパスやライブラリパスの情報を提供するツールです。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 29 MB

8.27.1. Pkg-config のインストール

Pkg-config をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --with-internal-glib \
            --disable-host-tool \
            --docdir=/usr/share/doc/pkg-config-0.29.2
```

configure オプションの意味

--with-internal-glib

これは pkg-config が内包しているバージョンの glib を利用するようにします。LFS においては Glib をインストールせず利用できないからです。

--disable-host-tool

本オプションは、pkg-config プログラムに対しての不要なハードリンクを生成しないようにします。

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場合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

8.27.2. Pkg-config の構成

インストールプログラム: pkg-config
インストールディレクトリ: /usr/share/doc/pkg-config-0.29.2

概略説明

pkg-config 指定されたライブラリやパッケージに対するメタ情報を返します。

8.28. Ncurses-6.3

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: 0.4 SBU
必要ディスク容量: 45 MB

8.28.1. 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 \
            --mandir=/usr/share/man \
            --with-shared \
            --without-debug \
            --without-normal \
            --with-cxx-shared \
            --enable-pc-files \
            --enable-widenc \
            --with-pkg-config-libdir=/usr/lib/pkgconfig
```

configure オプションの意味

--with-shared

これは Ncurses において共有 C ライブラリをビルドしインストールします。

--without-normal

これは Ncurses においてスタティックな C ライブラリのビルドおよびインストールを行わないようにします。

--without-debug

これは Ncurses においてデバッグライブラリのビルドおよびインストールを行わないようにします。

--with-cxx-shared

これは Ncurses において共有 C++ バインディングをビルドしインストールします。同時にスタティックな C++ バインディングのビルドおよびインストールは行わないようにします。

--enable-pc-files

本スイッチは pkg-config 用の .pc ファイルを生成しインストールすることを指示します。

--enable-widenc

本スイッチは通常のライブラリ (libncurses.so.6.3) ではなくワイド文字対応のライブラリ (libncursesw.so.6.3) 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。

本パッケージをインストールすると、所定位置にある libncursesw.so.6.3 が上書きされます。このときに、そのライブラリファイルのコードやデータを利用しているシェルプロセスが、クラッシュする場合があります。そこで本パッケージは DESTDIR を使ってインストールして、install コマンドによってライブラリファイルを正しく置き換えるようにします。configure では取り扱われないスタティックアーカイブは不要であるため、同様に削除されます。

```
make DESTDIR=$PWD/dest install
install -vm755 dest/usr/lib/libncursesw.so.6.3 /usr/lib
rm -v dest/usr/lib/libncursesw.so.6.3
cp -av dest/* /
```

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ー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ー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 ${lib}w.pc                   /usr/lib/pkgconfig/${lib}.pc
done
```

最後に古いアプリケーションにおいて、ビルド時に `-lncurses` を指定するものがあるため、これもビルド可能なものになります。

```
rm -vf                               /usr/lib/libcursesw.so
echo "INPUT(-lncursesw)" > /usr/lib/libcursesw.so
ln -sfv libcurses.so                 /usr/lib/libcurses.so
```

必要なら Ncurses のドキュメントをインストールします。

```
mkdir -pv          /usr/share/doc/ncurses-6.3
cp -v -R doc/*     /usr/share/doc/ncurses-6.3
```



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あり、バイナリコードのアプリケーションで非ワイド文字対応のものは Ncurses 5 にリンクされています。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 \
            --with-abi-version=5
make sources libs
cp -av lib/lib*.so.5* /usr/lib
```

8.28.2. Ncurses の構成

インストールプログラム:	captainfo (tic へのリンク), clear, infocmp, infotocap (tic へのリンク), ncursesw6-config, reset (tset へのリンク), tabs, tic, toe, tput, tset
インストールライブラリ:	libcursesw.so (libncursesw.so へのシンボリックリンクおよびリンカースクリプト), libformw.so, libmenuw.so, libncursesw.so, libpanelw.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, /usr/share/doc/ncurses-6.3

概略説明

captainfo	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。
ncursesw6-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。
tabs	端末上のタブストップの設定をクリアしたり設定したりします。
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。

<code>toe</code>	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。
<code>tput</code>	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。
<code>tset</code>	端末の初期化に利用します。
<code>libcursesw</code>	<code>libncursesw</code> へのリンク。
<code>libncursesw</code>	さまざま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 <code>make menuconfig</code> の実行によって表示されるメニューがあります。
<code>libformw</code>	フォームを実装するための関数を提供します。
<code>libmenuw</code>	メニューを実装するための関数を提供します。
<code>libpanelw</code>	パネルを実装するための関数を提供します。

8.29. Sed-4.8

Sed パッケージはストリームエディターを提供します。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 31 MB

8.29.1. 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=/usr
```

パッケージをコンパイルし HTML ドキュメントを生成します。

```
make
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。

```
chown -Rv tester .
su tester -c "PATH=$PATH make check"
```

パッケージとドキュメントをインストールします。

```
make install
install -d -m755 /usr/share/doc/sed-4.8
install -m644 doc/sed.html /usr/share/doc/sed-4.8
```

8.29.2. 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-4.8

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。

8.30. Psmisc-23.5

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: 5.8 MB

8.30.1. Psmisc のインストール

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
```

8.30.2. Psmisc の構成

インストールプログラム: fuser, killall, peekfd, prtstat, pslog, 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ーを調べます。
prtstat	プロセスに関する情報を表示します。
pslog	プロセスに対する現状のログパスを表示します。
pstree	稼働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。

8.31. Gettext-0.21

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: 2.7 SBU
必要ディスク容量: 235 MB

8.31.1. 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 \
            --disable-static \
            --docdir=/usr/share/doc/gettext-0.21
```

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
chmod -v 0755 /usr/lib/preloadable_libintl.so
```

8.31.2. Gettext の構成

インストールプログラム: autpoint, envsubst, gettext, gettext.sh, gettextize, 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.so, libgettextlib.so, libgettextpo.so, libgettextsrc.so, libtextstyle.so, preloadable_libintl.so
インストールディレクトリ: /usr/lib/gettext, /usr/share/doc/gettext-0.21, /usr/share/gettext, /usr/share/gettext-0.19.8

概略説明

autpoint	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。
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。
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せん。
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	さまざまな 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	さまざまな 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。
libtextstyle	テキストスタイリングライブラリ。
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。

8.32. Bison-3.8.2

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: 8.7 SBU
必要ディスク容量: 63 MB

8.32.1. 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 --docdir=/usr/share/doc/bison-3.8.2
```

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 5.5 SBU)

```
make check
```

パッケージをインストールします。

```
make install
```

8.32.2. 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 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。

8.33. Grep-3.7

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: 0.8 SBU
必要ディスク容量: 37 MB

8.33.1. 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
```

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
```

8.33.2. 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。これは非推奨となっているため、代わりに `grep -E` を使ってください。
- fgrep 固定文字列の一覧にマッチした行を表示します。これは非推奨となっているため、代わりに `grep -F` を使ってください。
- grep 基本的な正規表現に合致した行を出力します。

8.34. Bash-5.1.16

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: 50 MB

8.34.1. Bash のインストール

Bash をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/bash-5.1.16 \
            --without-bash-malloc \
            --with-installed-readline
```

configure オプションの意味

`--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 tester 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 tester .
```

本パッケージのテストスイートは、非 root ユーザーが実行するものとされていて、利用する端末が標準入力に接続できているものとしています。この仕様を満たすためには、Expect を使って新たな疑似端末を起動します。そして tester ユーザーとしてテストを実行します。

```
su -s /usr/bin/expect tester << EOF
set timeout -1
spawn make tests
expect eof
lassign [wait] _ _ _ value
exit $value
EOF
```

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 /usr/bin/bash --login
```

8.34.2. 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/include/bash, /usr/lib/bash, /usr/share/doc/bash-5.1.16

概略説明

bash 広く活用されているコマンドインタプリター。処理実行前には、指示されたコマンドラインをさまざま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プリター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。

8.35. Libtool-2.4.7

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: 1.5 SBU
必要ディスク容量: 43 MB

8.35.1. 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。

```
make check
```



注記

マルチコアのシステム上で libtool のテストをすると、その処理時間は大幅に減ります。実行する際には、上のコマンドに TESTSUITEFLAGS=-j<N> を加えます。例えば -j4 を指定するとテスト時間は 6 割以上減ります。

LFS ビルド環境下では 5 つのテストが失敗します。これはパッケージ間の相互依存のためです。automake をインストールした後に再テストすれば、全テストが成功します。

パッケージをインストールします。

```
make install
```

不要なスタティックライブラリを削除します。

```
rm -fv /usr/lib/libltdl.a
```

8.35.2. 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.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。

8.36. GDBM-1.23

GDBM パッケージは GNU データベースマネージャーを提供します。これは拡張性のあるハッシングなど、従来の UNIX dbm と同様のデータベース機能を実現するライブラリです。このライブラリにより、キーデータの収容、キーによるデータ検索と抽出、キーに基づいたデータ削除などを行うことができます。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 13 MB

8.36.1. 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 \
            --disable-static \
            --enable-libgdbm-compat
```

configure オプションの意味

`--enable-libgdbm-compat`

このオプションは libgdbm 互換ライブラリをビルドすることを指示します。LFS パッケージ以外において、かつての古い DBM ルーチンを必要とするものがあるかもしれません。

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場合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
```

8.36.2. GDBM の構成

インストールプログラム: gdbm_dump, gdbm_load, gdbmtool
インストールライブラリ: libgdbm.so, libgdbm_compat.so

概略説明

<code>gdbm_dump</code>	GDBM データベースをファイルにダンプします。
<code>gdbm_load</code>	GDBM のダンプファイルからデータベースを再生成します。
<code>gdbmtool</code>	GDBM データベースをテストし修復します。
<code>libgdbm</code>	ハッシュデータベースを取り扱う関数を提供します。
<code>libgdbm_compat</code>	古い DBM 関数を含んだ互換ライブラリ。

8.37. Gperf-3.1

Gperf は、キーセットに基づいて完全なハッシュ関数の生成を実現します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 6.0 MB

8.37.1. Gperf のインストール

Gperf をコンパイルするための準備をします。

```
./configure --prefix=/usr --docdir=/usr/share/doc/gperf-3.1
```

パッケージをコンパイルします。

```
make
```

同時実行によるテスト (-j オプションを 1 より大きくした場合) ではテストに失敗します。ビルド結果をテストする場合は以下を実行します。

```
make -j1 check
```

パッケージをインストールします。

```
make install
```

8.37.2. Gperf の構成

インストールプログラム: gperf
インストールディレクトリ: /usr/share/doc/gperf-3.1

概略説明

gperf キーセットに基づいて、完全なハッシュ関数を生成します。

8.38. Expat-2.4.8

Expat パッケージは XML を解析するためのストリーム指向 (stream oriented) な C ライブラリを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 12 MB

8.38.1. Expat のインストール

Expat をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-static \
            --docdir=/usr/share/doc/expat-2.4.8
```

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場合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。

```
install -v -m644 doc/*.{html,css} /usr/share/doc/expat-2.4.8
```

8.38.2. Expat の構成

インストールプログラム: xmlwf
インストールライブラリ: libexpat.so
インストールディレクトリ: /usr/share/doc/expat-2.4.8

概略説明

xmlwf XML ドキュメントが整形されているかどうかをチェックするユーティリティです。

libexpat XML を処理する API 関数を提供します。

8.39. Inetutils-2.3

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: 0.3 SBU
必要ディスク容量: 31 MB

8.39.1. Inetutils のインストール

Inetutils をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --bindir=/usr/bin \
            --localstatedir=/var \
            --disable-logger \
            --disable-whois \
            --disable-rcp \
            --disable-rexec \
            --disable-rlogin \
            --disable-rsh \
            --disable-servers
```

configure オプションの意味

`--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、より最新のバージョンをインストールするためです。

`--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-r*`

これらのパラメーターは、セキュリティの問題により用いるべきではない古いプログラムを作らないようにします。古いプログラムによる機能は BLFS ブックにて示す openssh でも提供されています。

`--disable-servers`

このオプションは Inetutils パッケージに含まれるさまざま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安全に扱うことができないものもあります。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
```

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
```

各種プログラムを適切な場所に移動します。

```
mv -v /usr/{,s}bin/ifconfig
```

8.39.2. Inetutils の構成

インストールプログラム: dnsdomainname, ftp, ifconfig, hostname, ping, ping6, talk, telnet, tftp, traceroute

概略説明

dnsdomainname システムの DNS ドメイン名を表示します。

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。
ifconfig	ネットワークインターフェースを管理します。
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
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。

8.40. Less-590

Less パッケージはテキストファイルビューアーを提供します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 4.2 MB

8.40.1. 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
```

8.40.2. Less の構成

インストールプログラム: `less`, `lessecho`, `lesskey`

概略説明

- `less` ファイルビューアーまたはページャー。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。

8.41. Perl-5.36.0

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:      9.4 SBU
必要ディスク容量:  236 MB
```

8.41.1. Perl のインストール

ここでビルドするバージョンの Perl は Compress::Raw::Zlib モジュールと Compress::Raw::Bzip2 モジュールをビルドします。しかしデフォルトでは内部にコピーされたライブラリソースを用いてビルドを行います。以下のコマンドは、既にインストールされているライブラリを用いるようにします。

```
export BUILD_ZLIB=False
export BUILD_BZIP2=0
```

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動設定を進めることもできます。Perl が自動判別するデフォルト設定に従うので良ければ、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 \
-Dprivlib=/usr/lib/perl5/5.36/core_perl \
-Darchlib=/usr/lib/perl5/5.36/core_perl \
-Dsitelib=/usr/lib/perl5/5.36/site_perl \
-Dsitearch=/usr/lib/perl5/5.36/site_perl \
-Dvendorlib=/usr/lib/perl5/5.36/vendor_perl \
-Dvendorarch=/usr/lib/perl5/5.36/vendor_perl \
-Dman1dir=/usr/share/man/man1 \
-Dman3dir=/usr/share/man/man3 \
-Dpager="/usr/bin/less -isR" \
-Duseshrplib \
-Dusethreads
```

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 more プログラムでなく less プログラムが利用されるようにします。

-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。

-Dusethreads

スレッドサポートをビルドします。

-Dprivlib, -Darchlib, -Dsitelib, ...

この設定は、Perl がインストール済みのモジュールを探す場所を指定します。LFS 編集者はディレクトリ構造として Perl の Major.Minor バージョン (5.36) の形に基づいて、インストールモジュールを配置することにしています。このようにしておくこと、新たなパッチレベル (5.36.0) によるアップグレードの際に、モジュールを再インストールする必要がなくなるためです。

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。(約 11 SBU)

```
make test
```

パッケージはインストールしクリーンアップします。

```
make install
unset BUILD_ZLIB BUILD_BZIP2
```

8.41.2. Perl の構成

インストールプログラム: corelist, cpan, enc2xs, encguess, h2ph, h2xs, instmodsh, json_pp, libnetcfg, perl, perl5.36.0 (perl へのハードリンク), perlbug, perldoc, perlivp, perlthanks (perlbug へのハードリンク), piconv, pl2pm, pod2html, pod2man, pod2text, pod2usage, podchecker, podselect, prove, ptar, ptardiff, ptargrep, shasum, splain, xsubpp, zipdetails

インストールライブラリ: ここで示しきれないほど多くのライブラリ

インストールディレクトリ: /usr/lib/perl5

概略説明

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。
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。
encguess	複数ファイルのエンコーディングを調査します。
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。
json_pp	特定の入出力フォーマット間でデータを変換します。
libnetcfg	Perl モジュール libnet 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。
perl5.36.0	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。
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。
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。
ptargrep	tar アーカイブ内のファイルに対してパターンマッチングを適用するための Perl プログラム。
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。
zipdetails	Zip ファイルの内部構造に関する情報を出力します。

8.42. XML::Parser-2.46

XML::Parser モジュールは James Clark 氏による XML パーサー Expat への Perl インターフェースです。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.3 MB

8.42.1. XML::Parser のインストール

XML::Parser をコンパイルするための準備をします。

```
perl Makefile.PL
```

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場合は以下を実行します。

```
make test
```

パッケージをインストールします。

```
make install
```

8.42.2. XML::Parser の構成

インストールモジュール: Expat.so

概略説明

Expat Perl Expat インターフェースを提供します。

8.43. Intltool-0.51.0

Intltool パッケージは、プログラムソースファイルから翻訳対象の文字列を抽出するために利用する国際化ツールです。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 1.5 MB

8.43.1. Intltool のインストール

perl-5.22 以降にて発生する警告メッセージを修正します。

```
sed -i 's:\\\\$\\{:\\\\$\\{:' intltool-update.in
```



注記

上の正規表現は、バックスラッシュが多すぎて変に思うかもしれません。ここでやっているのは '\$\$' という記述の並びに対して、右プレースの前にバックスラッシュを追加して '\$\$' を作り出しています。

Intltool 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場合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
```

```
install -v -Dm644 doc/I18N-HOWTO /usr/share/doc/intltool-0.51.0/I18N-HOWTO
```

8.43.2. Intltool の構成

インストールプログラム: intltool-extract, intltool-merge, intltool-prepare, intltool-update, intltoolize
インストールディレクトリ: /usr/share/doc/intltool-0.51.0, /usr/share/intltool

概略説明

intltoolize	パッケージに対して intltool を利用できるようにします。
intltool-extract	gettext が読み込むことの出来るヘッダーファイルを生成します。
intltool-merge	翻訳された文字列をさまざまな種類のファイルにマージします。
intltool-prepare	pot ファイルを更新し翻訳ファイルにマージします。
intltool-update	po テンプレートファイルを更新し翻訳ファイルにマージします。

8.44. Autoconf-2.71

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: 0.1 SBU 以下 (テスト込みで約 6.7 SBU)
必要ディスク容量: 24 MB

8.44.1. Autoconf のインストール

Autoconf 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には、以下を実行します。

```
make check
```



注記

マルチコアのシステム上で autoconf のテストをすると、その処理時間は大幅に減ります。実行する際には、上のコマンドに TESTSUITEFLAGS=-j<N> を加えます。例えば -j4 を指定するとテスト時間は 6 割以上減ります。

パッケージをインストールします。

```
make install
```

8.44.2. 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ない場合は、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。

8.45. Automake-1.16.5

Automake パッケージは Autoconf が利用する Makefile などを作成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 (テスト込みで約 7.7 SBU)
 必要ディスク容量: 116 MB

8.45.1. 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-1.16.5
```

パッケージをコンパイルします。

```
make
```

make オプションの `-j4` を用いるとテストを速く進めることができます。たとえ 1 つのプロセッサであっても有用であり、個々のテストにおける内部遅延に関係するためです。ビルド結果をテストするには以下を実行します。

```
make -j4 check
```

テスト `t/subobj.sh` は失敗します。

パッケージをインストールします。

```
make install
```

8.45.2. Automake の構成

インストールプログラム: `aclocal`, `aclocal-1.16` (`aclocal` へのハードリンク), `automake`, `automake-1.16` (`automake` へのハードリンク)

インストールディレクトリ: `/usr/share/aclocal-1.16`, `/usr/share/automake-1.16`, `/usr/share/doc/automake-1.16.5`

概略説明

<code>aclocal</code>	<code>configure.in</code> ファイルの内容に基づいて <code>aclocal.m4</code> ファイルを生成します。
<code>aclocal-1.16</code>	<code>aclocal</code> へのハードリンク。
<code>automake</code>	<code>Makefile.am</code> ファイルから <code>Makefile.in</code> 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 <code>Makefile.in</code> 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。 <code>configure.in</code> ファイルを調べて、適切な <code>Makefile.am</code> ファイルを検索します。そして対応する <code>Makefile.in</code> ファイルを生成します。
<code>automake-1.16</code>	<code>automake</code> へのハードリンク。

8.46. OpenSSL-3.0.5

OpenSSL パッケージは暗号化に関する管理ツールやライブラリを提供します。これを利用することにより、他のパッケージにおいて暗号化機能が実現されます。例えば OpenSSH、Email アプリケーション、(HTTPS サイトアクセスを行う) ウェブブラウザなどです。

概算ビルド時間: 5.0 SBU
必要ディスク容量: 476 MB

8.46.1. OpenSSL のインストール

OpenSSL をコンパイルするための準備をします。

```
./config --prefix=/usr \
         --openssldir=/etc/ssl \
         --libdir=lib \
         shared \
         zlib-dynamic
```

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場合は以下を実行します。

```
make test
```

カーネル設定によっては (CONFIG_CRYPT_USER_API* の設定に一貫性がないと)、30-test_afalg.t というテストが 1 つだけ失敗することがわかっています。失敗しても、無視してかまいません。

パッケージをインストールします。

```
sed -i '/INSTALL_LIBS/s/libcrypto.a libssl.a//' Makefile
make MANSUFFIX=ssl install
```

ドキュメントディレクトリにバージョンを含めます。他のパッケージとの整合をとるためです。

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-3.0.5
```

必要であれば、さらにドキュメントをインストールします。

```
cp -vfr doc/* /usr/share/doc/openssl-3.0.5
```



注記

ぜい弱性への対処を行った新バージョンが公開されたら、OpenSSL をアップデートすることになります。OpenSSL 3.0.0 以降では、バージョンのつけ方が MAJOR.MINOR.PATCH という形式になりました。API/API の互換性は、同一の MAJOR バージョン番号では保証されます。本パッケージは libcrypto.so または libssl.so へのリンクを行っていますが、LFS では共有ライブラリをインストールするだけなので、MAJOR バージョン番号が同一のアップグレードである限りは、パッケージを再コンパイルする必要はありません。

そうであっても、それらのライブラリにリンクしているプログラムが稼働中であるなら、一度停止してから再起動することが必要です。詳しくは関連する話が「アップグレードに関する問題」にあるので参照してください。

8.46.2. OpenSSL の構成

インストールプログラム: c_rehash, openssl
インストールライブラリ: libcrypto.so, libssl.so
インストールディレクトリ: /etc/ssl, /usr/include/openssl, /usr/lib/engines, /usr/share/doc/openssl-3.0.5

概略説明

c_rehash ディレクトリ内のすべてのファイルをスキャンする Perl スクリプト。それらのファイルに対するハッシュ値へのシンボリックリンクを生成します。c_rehash の利用は非推奨と考えられており、この代わりに openssl rehash コマンドを使ってください。

<code>openssl</code>	OpenSSL の暗号化ライブラリが提供するさまざまな関数を、シェルから利用するためのコマンドラインツール。 <code>man 1 openssl</code> に示される数多くの関数を利用することができます。
<code>libcrypto.so</code>	各種のインターネット標準にて採用されている暗号化アルゴリズムを幅広く実装しています。このライブラリが提供する機能は、SSL、TLS、S/MIME を実装する OpenSSL において利用されており、また OpenSSH、OpenPGP、あるいはこの他の暗号化標準の実装にも利用されています。
<code>libssl.so</code>	トランスポート層セキュリティ (Transport Layer Security; TLS v1) プロトコルを実装しています。これは豊富な API 関数とそのドキュメントを提供します。ドキュメントは <code>man 3 ssl</code> の実行により参照できます。

8.47. Kmod-30

Kmod パッケージは、カーネルモジュールをロードするためのライブラリやユーティリティを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 12 MB

8.47.1. Kmod のインストール

Kmod をコンパイルするための準備をします。

```
./configure --prefix=/usr      \  
            --sysconfdir=/etc  \  
            --with-openssl    \  
            --with-xz         \  
            --with-zstd       \  
            --with-zlib
```

configure オプションの意味

`--with-openssl`

このオプションは Kmod において、カーネルモジュールに対する PKCS7 署名を取り扱えるようにします。

`--with-xz`, `--with-zlib`, `--with-zstd`

これらのオプションは、Kmod が圧縮されたカーネルモジュールを取り扱えるようにするものです。

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スイートでは、生のカーネルヘッダー（以前にインストールした「健全化 (sanitized)」されたヘッダーではないもの）が必要です。これは LFS の範囲を超えているものです。

パッケージインストールし、Module-Init-Tools パッケージとの互換性を保つためにシンボリックリンクを生成します。Module-Init-Tools パッケージは、これまで Linux カーネルモジュールを取り扱っていたものです。

make install

```
for target in depmod insmod modinfo modprobe rmmmod; do  
  ln -sfv ../bin/kmod /usr/sbin/$target  
done  
  
ln -sfv kmod /usr/bin/lsmmod
```

8.47.2. Kmod の構成

インストールプログラム: depmod (kmod へのリンク), insmod (kmod へのリンク), kmod, lsmmod (kmod へのリンク), modinfo (kmod へのリンク), modprobe (kmod へのリンク), rmmmod (kmod へのリンク)

インストールライブラリ: libkmod.so

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	稼働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。
kmod	カーネルモジュールのロード、アンロードを行います。
lsmmod	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。
rmmmod	稼働中のカーネルからモジュールをアンロードします。

`libkmod` このライブラリは、カーネルモジュールのロード、アンロードを行う他のプログラムが利用します。

8.48. Elfutils-0.187 から取り出した libelf

Libelf は、ELF (Executable and Linkable Format) 形式のファイルを扱うライブラリを提供します。

概算ビルド時間: 0.9 SBU
必要ディスク容量: 117 MB

8.48.1. Libelf のインストール

Libelf は elfutils-0.187 パッケージに含まれます。ソース tarball として elfutils-0.187.tar.bz2 を利用します。

Libelf をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-debuginfod \
            --enable-libdebuginfod=dummy
```

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場合は以下を実行します。

```
make check
```

Libelf のみをインストールします。

```
make -C libelf install
install -vm644 config/libelf.pc /usr/lib/pkgconfig
rm /usr/lib/libelf.a
```

8.48.2. Libelf の構成

インストールライブラリ: libelf.so (シンボリックリンク), libelf-0.187.so
インストールディレクトリ: /usr/include/elfutils

概略説明

libelf ELF オブジェクトファイルを取り扱うための API 関数を提供します。

8.49. Libffi-3.4.2

Libffi ライブラリは、さまざまな呼出規約 (calling conventions) に対しての、移植性に優れた高レベルのプログラミングインターフェースを提供します。このライブラリを用いることで、プログラム実行時に呼出インターフェース記述 (call interface description) による関数を指定して呼び出すことができるようになります。

概算ビルド時間: 1.8 SBU
必要ディスク容量: 10 MB

8.49.1. Libffi のインストール



注記

GMP と同じように libffi では、利用中のプロセッサに応じた最適化を行なってビルドされます。異なるシステムに向けてのビルドを行う場合は、以下のコマンドにおいて `--with-gcc-arch=` を使って、そのシステム上の CPU の実装を完全に表すアーキテクチャー名に変更してください。そうしなかった場合には、libffi をリンクするアプリケーションにおいて Illegal Operation エラーを発生させることになります。

libffi をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --disable-static \
            --with-gcc-arch=native \
            --disable-exec-static-tramp
```

configure オプションの意味

`--with-gcc-arch=native`

現状のシステムに応じて GCC が最適化されるようにします。仮にこれを指定しなかった場合、システムを誤認して誤ったコードを生成してしまう場合があります。生成されたコードが、より劣ったシステム向けのネイティブコードをコピーしていたとすると、より劣ったシステムに対するパラメーターを指定することとなります。システムに応じた詳細は the x86 options in the GCC manual を参照してください。

`--disable-exec-static-tramp`

スタティックなトランポリン (trampoline) サポートを無効にします。これは libffi における新しいセキュリティ機能ですが、BLFS パッケージの中 (特に GJS) では、この機能に対応していないものがあります。

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場合は、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
```

8.49.2. Libffi の構成

インストールライブラリ: libffi.so

概略説明

libffi 外部関数インターフェース API 関数を提供します。

8.50. Python-3.10.6

Python 3 パッケージは Python 開発環境を提供します。オブジェクト指向プログラミング、スクリプティング、大規模プログラムのプロトタイピング、アプリケーション開発などに有用なものです。

概算ビルド時間: 3.4 SBU
必要ディスク容量: 283 MB

8.50.1. Python 3 のインストール

Python をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --enable-shared \
            --with-system-expat \
            --with-system-ffi \
            --enable-optimizations
```

configure オプションの意味

`--with-system-expat`

本スイッチは、システムにインストールされている Expat をリンクすることを指示します。

`--with-system-ffi`

本スイッチは、システムにインストールされている libffi をリンクすることを指示します。

`--enable-optimizations`

本スイッチは安定的ではあるものの高くつく最適化を有効にします。

パッケージをコンパイルします。

```
make
```

この時点においてテスト実行することはお勧めしません。部分的にしか仕上がっていない LFS 環境では安定せずハングすることがあります。テストを必要とする場合は、本章を一番最後まで進めてから再度実行するか、あるいは BLFS において Python 3 をインストール際に行います。そうではなくここでテスト実行をするなら `make test` を実行します。

パッケージをインストールします。

```
make install
```

いくつかの場面において Python 3 プログラムやモジュールをインストールする際には、全ユーザー向けのインストールを行うために root ユーザーになって pip3 コマンドを用いています。このことは Python 開発者が推奨している、仮想環境内にて一般ユーザーにより（そのユーザーが pip3 を実行することで）パッケージビルドを行う方法とは相容れないものです。これを行っているため、root ユーザーとして pip3 を用いると、警告メッセージが複数出力されます。開発者がなぜその方法を推奨しているかという点、システムパッケージマネージャー（たとえば dpkg）などと衝突が発生するからです。LFS ではシステムワイドなパッケージマネージャーを利用していないため、このことは問題となりません。

また pip3 そのものが、自分の最新版が存在しないかどうかを実行時に確認します。LFS の chroot 環境においては、ドメイン名解決がまだ設定されていないので、最新版の確認は失敗して警告が出力されます。LFS システムを再起動してネットワーク設定を行えば、最新版の存在時には、あらかじめビルドされていた wheel を PyPI から更新するような警告メッセージが示されます。もっとも LFS では pip3 を Python 3 の一部として考えるので、個別に更新しないでください。したがってあらかじめビルドされた wheel を更新することは、ソースコードから Linux システムをビルドするという目的から逸脱してしまいます。このことから、最新版の pip3 を求める警告も無視してください。警告メッセージを省略したい場合は、以下のコマンドを実行します。

```
cat > /etc/pip.conf << EOF
[global]
root-user-action = ignore
disable-pip-version-check = true
EOF
```




重要

LFS や BLFS においては通常、Python モジュールのビルドとインストールには `pip3` コマンドを用いています。この両ブックにおいて実行する `pip3 install` コマンドは、Python 仮想環境内でない場合には `root` ユーザーで実行するようにしてください。 `root` ユーザー以外によって `pip3 install` を実行しても問題なく動作するよう見えるかもしれませんが、インストールしたモジュールが別のユーザーからはアクセスできない事態を作り出してしまいます。

`pip3 install` はデフォルトでは、すでにインストールされているモジュールを再インストールすることは行いません。 `pip3 install` コマンドを使ってモジュールのアップグレードを行う（たとえば `meson-0.61.3` から `meson-0.62.0` にするような場合）には、コマンドラインに `--upgrade` オプションを含めてください。またモジュールのダウングレードや再インストールが必要となる理由が確実にあるのであれば、コマンドラインに `-force-reinstall --no-deps` を含めて実行してください。

必要なら、整形済みドキュメントをインストールします。

```
install -v -dm755 /usr/share/doc/python-3.10.6/html

tar --strip-components=1 \
  --no-same-owner \
  --no-same-permissions \
  -C /usr/share/doc/python-3.10.6/html \
  -xvf ../python-3.10.6-docs-html.tar.bz2
```

ドキュメント `install` コマンドの意味

`--no-same-owner` と `--no-same-permissions`

インストールするファイルの所有者とパーミッションを適切に設定します。このオプションがないと `tar` によって展開されるファイルは、アップストリームが作り出した値になってしまうためです。

8.50.2. Python 3 の構成

インストールプログラム:	2to3, idle3, pip3, pydoc3, python3, python3-config
インストールライブラリ:	libpython3.10.so, libpython3.so
インストールディレクトリ:	/usr/include/python3.10, /usr/lib/python3, /usr/share/doc/python-3.10.6

概略説明

2to3	Python 2.x のソースコードを読み込み、種々の変更を行って Python 3.x 用の適正なソースコードに変換するための Python プログラムです
idle3	Python に特化した GUI エディターを起動するラッパースクリプト。このスクリプトを実行するには、Python より前に Tk をインストールして、Python モジュールである Tkinter をビルドしておく必要があります。
pip3	Python のパッケージインストーラー。この pip を使って Python Package Index などのインデックスサイトから各種パッケージをインストールできます。
pydoc3	Python ドキュメントツール。
python3	インタプリターであり、対話的なオブジェクト指向プログラミング言語。

8.51. Wheel-0.37.1

Wheel は Python wheel パッケージング標準に基づいた標準実装の Python ライブラリです。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 956 KB

8.51.1. Wheel のインストール

以下のコマンドを実行して wheel をインストールします。

```
pip3 install --no-index $PWD
```

pip3 オプションの意味

install

パッケージをインストールします。

--no-index

pip がオンラインパッケージリポジトリ (PyPI) からファイルを取得しないようにします。パッケージ類が適切な順番でインストールされていれば、最初にファイルを取得しておく必要はないはずです。ただしこのオプションをつけておくことで、ユーザーが操作を誤っても安全であるようにします。

\$PWD

インストールするファイルを現在のワーキングディレクトリ内から探し出します。

8.51.2. Wheel の構成

インストールプログラム: wheel
インストールディレクトリ: /usr/lib/python3.10/site-packages/wheel, /usr/lib/python3.10/site-packages/wheel-0.37.1-py3.10.egg-info

概略説明

wheel wheel パッケージの解凍、圧縮、変換を行うユーティリティです。

8.52. Ninja-1.11.0

このパッケージは、処理速度を重視した軽量なビルドシステムを提供します。

概算ビルド時間: 0.6 SBU
必要ディスク容量: 79 MB



ヒント

本節は正確に言うと `systemd` を利用しないのであれば LFS において必要ありません。一方で `meson/ninja` というものは強力なビルドシステムであり、利用する機会がかなり多いものと思われます。BLFS ブック においては、これを必要とするパッケージがいくつかあります。

8.52.1. Ninja のインストール

`ninja` は同時に最大数のプロセスにより処理実行します。そのプロセス数はデフォルトでは、システムのコア数に 2 を加えたものとなります。このことが CPU をオーバーヒートさせたり、out of memory を引き起こす場合があります。コマンドラインから実行する場合には `-jN` パラメーターを使って、並行プロセスの数を制御することもできます。ただ `ninja` の実行を組み込んでいるパッケージの場合は `-j` パラメーターを与えることができません。

以降に示す 任意 の手順を用いると、並行プロセス数を環境変数 `NINJAJOBS` から制御できるようになります。たとえば 以下のように設定します。

```
export NINJAJOBS=4
```

こうすると `ninja` の並行プロセスを 4 つに制限できます。

必要な場合は、環境変数 `NINJAJOBS` を利用するために以下を実行します。

```
sed -i '/int Guess/a \  
int    j = 0;\ \  
char*  jobs = getenv( "NINJAJOBS" );\  
if ( jobs != NULL ) j = atoi( jobs );\  
if ( j > 0 ) return j;\ \  
' src/ninja.cc
```

以下を実行して `ninja` をビルドします。

```
python3 configure.py --bootstrap
```

`build` オプションの意味

`--bootstrap`

本パラメーターは、この時点でのシステムに対して `ninja` 自身を再ビルドすることを指示します。

ビルド結果をテストする場合は、以下を実行します。

```
./ninja ninja_test  
./ninja_test --gtest_filter=--SubprocessTest.SetWithLots
```

パッケージをインストールします。

```
install -vm755 ninja /usr/bin/  
install -vDm644 misc/bash-completion /usr/share/bash-completion/completions/ninja  
install -vDm644 misc/zsh-completion /usr/share/zsh/site-functions/_ninja
```

8.52.2. Ninja の構成

インストールプログラム: `ninja`

概略説明

`ninja` Ninja ビルドシステム。

8.53. Meson-0.63.1

Meson はオープンソースによるビルドシステムです。非常に高速であり、できるかぎりユーザーフレンドリーであることを意識しています。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 38 MB



ヒント

本節は正確に言うと `systemd` を利用しないのであれば LFS において必要ありません。一方で `meson/ninja` というものは強力なビルドシステムであり、利用する機会がかなり多いものと思われます。BLFS ブック においては、これを必要とするパッケージがいくつかあります。

8.53.1. Meson のインストール

Meson をビルドするには、以下のコマンドを実行します。

```
pip3 wheel -w dist --no-build-isolation --no-deps $PWD
```

このテストスイートには、LFS の範囲外としているパッケージがいくつか必要です。

パッケージをインストールします。

```
pip3 install --no-index --find-links dist meson
install -vDm644 data/shell-completions/bash/meson /usr/share/bash-completion/completions/meson
install -vDm644 data/shell-completions/zsh/_meson /usr/share/zsh/site-functions/_meson
```

`install` パラメーターの意味

`-w dist`

生成された wheel を `dist` ディレクトリに配置します。

`--find-links dist`

`dist` ディレクトリから wheel をインストールします。

8.53.2. Meson の構成

インストールプログラム: meson
インストールディレクトリ: /usr/lib/python3.10/site-packages/meson-0.63.1.dist-info, /usr/lib/python3.10/site-packages/mesonbuild

概略説明

meson 生産性の高いビルドシステム。

8.54. Coreutils-9.1

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: 2.8 SBU
必要ディスク容量: 159 MB

8.54.1. Coreutils のインストール

POSIX によると 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、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-9.1-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、このパッチの適用前でも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。

```
autoreconf -fiv
FORCE_UNSAFE_CONFIGURE=1 ./configure \
    --prefix=/usr \
    --enable-no-install-program=kill,uptime
```

configure オプションの意味

autoreconf

国際化対応を行うパッチによって、当パッケージのビルドシステムが修正されます。したがって設定ファイル類を再生成する必要があります。

FORCE_UNSAFE_CONFIGURE=1

この環境変数は root ユーザーによりパッケージをビルドできるようにします。

--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。

ここからテストスイートを実施していきます。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=tester check-root
```

ここからは tester ユーザー向けのテストを実行します。ただしテストの中には、複数のグループに属するユーザーを必要とするものがあります。そのようなテストが確実に実施されるように、一時的なグループを作って tester ユーザーがそれに属するようにします。

```
echo "dummy:x:102:tester" >> /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 tester .
```

テストを実行します。

```
su tester -c "PATH=$PATH make RUN_EXPENSIVE_TESTS=yes check"
```

sort-NaN-infloop というテストは GCC-12 を用いると失敗します。

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/chroot /usr/sbin
mv -v /usr/share/man/man1/chroot.1 /usr/share/man/man8/chroot.8
sed -i 's/"1"/"8"/' /usr/share/man/man8/chroot.8
```

8.54.2. Coreutils の構成

インストールプログラム: [, b2sum, base32, base64, basename, basenc, 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, mktmp, mv, nice, nl, nohup, nproc, numfmt, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, realpath, rm, rmdir, runcon, seq, shasum, 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 (/usr/libexec/coreutils ディレクトリ内)

インストールディレクトリ: /usr/libexec/coreutils

概略説明

[実際のコマンドは /usr/bin/[であり、これは test コマンドへのシンボリックリンクです。
base32	base32 規格 (RFC 4648) に従ってデータのエンコード、デコードを行います。
base64	base64 規格 (RFC 3548) に従ってデータのエンコード、デコードを行います。
b2sum	Prints or checks BLAKE2 (512-bit) checksums
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。
basenc	各種アルゴリズムを利用したデータのエンコード、出コードを行います。
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	ソート済み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ー型特殊ファイル (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。
numfmt	記述された文字列と数値を互いに変換します。
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せん。
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	軽量の 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。
realpath	解析されたパスを表示します。
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。
shasum	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	標準ストリームのバッファリング操作を変更してコマンド実行します。
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。

8.55. Check-0.15.2

Check は C 言語に対してのユニットテストのフレームワークです。

概算ビルド時間: 0.1 SBU (テスト込みで約 3.6 SBU)
必要ディスク容量: 12 MB

8.55.1. Check のインストール

Check をコンパイルするための準備をします。

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

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。 テストスイートを実行する場合は、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 docdir=/usr/share/doc/check-0.15.2 install
```

8.55.2. Check の構成

インストールプログラム: checkmk
インストールライブラリ: libcheck.so

概略説明

checkmk Check ユニットテストフレームワークにて利用される、C 言語ユニットテストを生成するための Awk スクリプト。
libcheck.so テストプログラムから Check を呼び出すための関数を提供します。

8.56. Diffutils-3.8

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: 0.6 SBU
必要ディスク容量: 35 MB

8.56.1. 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
```

8.56.2. 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。

8.57. Gawk-5.1.1

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: 0.4 SBU
必要ディスク容量: 44 MB

8.57.1. Gawk のインストール

まずは不要なファイルがインストールされないようにします。

```
sed -i 's/extras//' Makefile.in
```

Gawk 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
```

必要ならドキュメントをインストールします。

```
mkdir -pv /usr/share/doc/gawk-5.1.1
cp -v doc/{awkforai.txt,*.{eps,pdf,jpg}} /usr/share/doc/gawk-5.1.1
```

8.57.2. Gawk の構成

インストールプログラム: awk (gawk へのリンク), gawk, awk-5.1.1
 インストールライブラリ: filefuncs.so, fnmatch.so, fork.so, inplace.so, intdiv.so, ordchr.so, readdir.so, readfile.so, revoutput.so, revtwoway.so, rvarray.so, time.so (すべて /usr/lib/gawk ディレクトリ内)
 インストールディレクトリ: /usr/lib/gawk, /usr/libexec/awk, /usr/share/awk, /usr/share/doc/gawk-5.1.1

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-5.1.1 gawk へのハードリンク。

8.58. Findutils-4.9.0

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: 0.8 SBU
必要ディスク容量: 52 MB

8.58.1. 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。

```
case $(uname -m) in
  i?86)  TIME_T_32_BIT_OK=yes ./configure --prefix=/usr --localstatedir=/var/lib/locate ;;
  x86_64) ./configure --prefix=/usr --localstatedir=/var/lib/locate ;;
esac
```

configure オプションの意味

TIME_32_BIT_OK=yes

この設定は 32 ビットシステム上でビルドする際に必要となります。

--localstatedir

locate データベースの場所を FHS コンプライアンスに準拠するディレクトリ /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。

```
chown -Rv tester .
su tester -c "PATH=$PATH make check"
```

パッケージをインストールします。

```
make install
```

8.58.2. Findutils の構成

インストールプログラム: find, locate, updatedb, xargs

インストールディレクトリ: /var/lib/locate

概略説明

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。
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。
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。

8.59. Groff-1.22.4

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: 0.5 SBU
必要ディスク容量: 88 MB

8.59.1. 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
```

このパッケージでは並行ビルドはサポートされていません。パッケージをコンパイルします。

```
make -j1
```

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
```

8.59.2. Groff の構成

インストールプログラム: addftinfo, afmtodit, chem, eqn, eqn2graph, gdiffmk, glilypond, gperl, gpinyin, grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, gropdf, grops, grotty, hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfmom, 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-1.22.4, /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、その差異を変更マークとして出力します。
glilypond	lilypond 言語で書かれたシートミュージック (sheet music) を groff 言語に変換します。
gperl	groff プリプロセッサであり groff ファイルへの perl コード追加を行います。
gpinyin	groff プリプロセッサであり groff ファイルへの中国語発音 Pinyin 追加を行います。
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 ドライバープログラム。
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 <code>-e</code> 、 <code>-man</code> 、 <code>-me</code> 、 <code>-mm</code> 、 <code>-ms</code> 、 <code>-p</code> 、 <code>-s</code> 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 プリンター (LBP-4 または LBP-8 シリーズのレーザープリンター) に対する groff ドライバプログラム。
grolj4	HP LaserJet 4 プリンターに対しての PCL5 フォーマットを出力する groff ドライバプログラム。
gropdf	GNU troff の出力を PDF に変換します。
grops	GNU troff の出力を PostScript に変換します。
grotty	GNU troff の出力を、タイプライター風のデバイスに適した形式に変換します。
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。
pdfmom	groff 関連ラッパー。mom マクロによるファイルから PDF を生成します。
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。
tfmtdit	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。

8.60. GRUB-2.06

GRUB パッケージは GGrand Unified Bootloader を提供します。

概算ビルド時間: 0.7 SBU
必要ディスク容量: 159 MB

8.60.1. GRUB のインストール



注記

システムが UEFI をサポートしていて、これを使って LFS を起動しようとする場合は、LFS における本パッケージは省略することができます。その場合は本章の終わりに、BLFS ページ に従って UEFI 対応の GRUB (およびその依存パッケージ) をインストールしてください。

GRUB をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --sysconfdir=/etc \
            --disable-efiemu \
            --disable-werror
```

configure オプションの意味

`--disable-werror`

本オプションは、最新の flex によって警告が出力されても、ビルドを成功させるために指定します。

`--disable-efiemu`

このオプションは LFS にとって不要な機能やテストプログラム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

本パッケージのテストスイートの利用はお勧めできません。テストのほとんどが、限定されている今の LFS 環境内では利用できないパッケージに依存しています。それでもテストを行うのであれば、`make check` を実行します。

パッケージをインストールします。

make install

```
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

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「GRUB を用いたブートプロセスの設定」で説明しています。

8.60.2. GRUB の構成

インストールプログラム:

grub-bios-setup, grub-editenv, grub-file, grub-fstest, grub-glue-efi, grub-install, grub-kbdcomp, grub-macbless, grub-menulst2cfg, grub-mkconfig, grub-mkimage, grub-mklayout, grub-mknetdir, grub-mkpasswd-pbkdf2, grub-mkrelpath, grub-mkrescue, grub-mkstandalone, grub-ofpathname, grub-probe, grub-reboot, grub-render-label, grub-script-check, grub-set-default, grub-sparc64-setup, grub-syslinux2cfg

インストールディレクトリ:

/usr/lib/grub, /etc/grub.d, /usr/share/grub, /boot/grub (grub-install が初めて起動される時)

概略説明

grub-bios-setup	grub-install に対するヘルパープログラム。
grub-editenv	環境ブロック (environment block) を編集するツール。
grub-file	FILE が指定されたタイプであるかどうかをチェックします。
grub-fstest	ファイルシステムドライバをデバッグするツール。
grub-glue-efi	32 ビットおよび 64 ビットの実行バイナリを Apple ユニバーサル形式に結合します。
grub-install	指定したドライブに GRUB をインストールします。

<code>grub-kbdcomp</code>	xkb レイアウトを GRUB が認識できる他の書式に変換するスクリプト。
<code>grub-macbless</code>	HFS または HFS+ ファイルに対する Mac 風の <code>bless</code>
<code>grub-menulst2cfg</code>	GRUB Legacy の <code>menu.lst</code> を GRUB 2 にて利用される <code>grub.cfg</code> に変換します。
<code>grub-mkconfig</code>	GRUB の設定ファイルを生成します。
<code>grub-mkimage</code>	GRUB のブートイメージ (bootable image) を生成します。
<code>grub-mklayout</code>	GRUB のキーボードレイアウトファイルを生成します。
<code>grub-mknetdir</code>	GRUB のネットブートディレクトリを生成します。
<code>grub-mkpasswd-pbkdf2</code>	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。
<code>grub-mkrelpath</code>	システムのパスをルートからの相対パスとします。
<code>grub-mkrescue</code>	フロッピーディスクや CDROM/DVD 用の GRUB のブートイメージを生成します。
<code>grub-mkstandalone</code>	スタンドアロンイメージを生成します。
<code>grub-ofpathname</code>	GRUB デバイスのパスを出力するヘルパープログラム。
<code>grub-probe</code>	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。
<code>grub-reboot</code>	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。
<code>grub-render-label</code>	Apple Mac に対して Apple <code>.disk_label</code> を提供します。
<code>grub-script-check</code>	GRUB の設定スクリプトにおける文法をチェックします。
<code>grub-set-default</code>	デフォルトのブートメニューを設定します。
<code>grub-sparc64-setup</code>	<code>grub-setup</code> に対するヘルパープログラム。
<code>grub-syslinux2cfg</code>	<code>syslinux</code> の設定ファイルを <code>grub.cfg</code> フォーマットに変換します。

8.61. Gzip-1.12

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: 0.3 SBU
必要ディスク容量: 21 MB

8.61.1. 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
```

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
```

8.61.2. Gzip の構成

インストールプログラム: gunzip, gzexe, gzip, uncompress (gunzip へのハードリンク), 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。

8.62. IPRoute2-5.19.0

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: 16 MB

8.62.1. 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 LFS では取り扱わない Berkeley DB に依存しています。したがって arpd プログラムはインストールしません。ただし arpd プログラムに対応するディレクトリや man ページはインストールされてしまいます。これをインストールしないように、以下の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 <https://www.linuxfromscratch.org/blfs/view/11.2/server/db.html#db> に示される Berkeley DB の構築手順に従ってください。

```
sed -i /ARPD/d Makefile
rm -fv man/man8/arpd.8
```

パッケージをコンパイルします。

```
make NETNS_RUN_DIR=/run/netns
```

本パッケージには有効な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 SBINDIR=/usr/sbin install
```

必要な場合はドキュメントをインストールします。

```
mkdir -pv /usr/share/doc/iproute2-5.19.0
cp -v COPYING README* /usr/share/doc/iproute2-5.19.0
```

8.62.2. IPRoute2 の構成

インストールプログラム: bridge, ctstat (lnstat へのリンク), genl, ifcfg, ifstat, ip, lnstat, nstat, routef, routil, 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, /usr/lib/tc, /usr/share/doc/iproute2-5.19.0

概略説明

bridge ネットワークブリッジを設定します。
ctstat 接続ステータスの表示ユーティリティ。
genl 汎用的な netlink ユーティリティフロントエンド。
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せん。
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/route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。

8.63. Kbd-2.5.1

Kbd パッケージは、キーテーブル (key-table) ファイル、コンソールフォント、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
必要ディスク容量: 35 MB

8.63.1. 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-2.5.1-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。

不要なプログラム `resizecons` とその man ページを削除します。(今はもう存在しない `svgalib` がビデオモードファイルを提供するために利用していたものであり、普通は `setfont` コマンドがコンソールサイズを適切に設定します。)

```
sed -i '/RESIZECONS_PROGS=/s/yes/no/' configure
sed -i 's/resizecons.8 //' docs/man/man8/Makefile.in
```

Kbd をコンパイルするための準備をします。

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

`configure` オプションの意味

`--disable-vlock`

このオプションは `vlock` ユーティリティをビルドしないようにします。そのユーティリティは PAM ライブラリが必要ですが、`chroot` 環境では利用することができません。

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場合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
```



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。

必要ならドキュメントをインストールします。

```
mkdir -pv /usr/share/doc/kbd-2.5.1
cp -R -v docs/doc/* /usr/share/doc/kbd-2.5.1
```

8.63.2. Kbd の構成

インストールプログラム: `chvt`, `deallocvt`, `dumpkeys`, `fgconsole`, `getkeycodes`, `kbinfo`, `kbd_mode`, `kbdrate`, `loadkeys`, `loadunimap`, `mapscrn`, `openvt`, `psfaddtable` (`psfxtable` へのリンク), `psfgettable` (`psfxtable` へのリンク), `psfstriptime` (`psfxtable` へのリンク), `psfxtable`, `setfont`, `setkeycodes`, `setleds`, `setmetamode`, `setvtrgb`, `showconsolefont`, `showkey`, `unicode_start`, `unicode_stop`

インストールディレクトリ: `/usr/share/consolefonts`, `/usr/share/consoletrans`, `/usr/share/keymaps`, `/usr/share/doc/kbd-2.5.1`, `/usr/share/unimaps`

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。
kbdinfo	コンソール状態に関する情報を取得します。
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	Unicode キャラクターテーブルをコンソールフォントに追加します。
psfgettable	コンソールフォントから埋め込まれた Unicode キャラクターテーブルを抽出します。
psfstrietable	コンソールフォントから埋め込まれた Unicode キャラクターテーブルを削除します。
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。
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。
setvtrgb	仮想端末すべてに対してコンソールのカラーマップを設定します。
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。

8.64. Libpipeline-1.5.6

Libpipeline パッケージは、サブプロセスのパイプラインを柔軟かつ便利に取り扱うライブラリを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 10 MB

8.64.1. Libpipeline のインストール

Libpipeline 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場合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
```

8.64.2. Libpipeline の構成

インストールライブラリ: libpipeline.so

概略説明

libpipeline このライブラリは、サブプロセス間のパイプラインを安全に構築するために利用されます。

8.65. Make-4.3

Make パッケージは、対象となるパッケージのソースファイルを用いて、実行モジュールやそれ以外のファイルの生成、管理を行うプログラムを提供します。

概算ビルド時間: 0.5 SBU
必要ディスク容量: 14 MB

8.65.1. 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
```

8.65.2. 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。

8.66. Patch-2.7.6

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: 0.2 SBU
必要ディスク容量: 12 MB

8.66.1. 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=/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
```

8.66.2. 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。

8.67. Tar-1.34

Tar パッケージは tar アーカイブの生成を行うとともに、アーカイブ操作に関する多くの処理を提供します。Tar はすでに生成されているアーカイブからファイルを抽出したり、ファイルを追加したりします。あるいはすでに保存されているファイルを更新したり一覧を表示したりします。

概算ビルド時間: 1.7 SBU
必要ディスク容量: 40 MB

8.67.1. Tar のインストール

Tar をコンパイルするための準備をします。

```
FORCE_UNSAFE_CONFIGURE=1 \
./configure --prefix=/usr
```

configure オプションの意味

FORCE_UNSAFE_CONFIGURE=1

このオプションは、mknod に対するテストを root ユーザーにて実行するようにします。一般にこのテストを root ユーザーで実行することは危険なものとされますが、ここでは部分的にビルドしたシステムでテストするものであるため、オーバーライドすることで支障はありません。

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ために以下を実行します。

```
make check
```

テストの 1 つ capabilities: binary store/restore は、LFS が selinux を含んでいないため、実行に失敗します。ただし LFS ビルドに利用するファイルシステム上において、ホストカーネルが拡張属性をサポートしていない場合、このテストはスキップされます。

パッケージをインストールします。

```
make install
```

```
make -C doc install-html docdir=/usr/share/doc/tar-1.34
```

8.67.2. Tar の構成

インストールプログラム: tar
インストールディレクトリ: /usr/share/doc/tar-1.34

概略説明

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。

8.68. Texinfo-6.8

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: 0.6 SBU
必要ディスク容量: 114 MB

8.68.1. 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 パラメーターの意味

```
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。

```
pushd /usr/share/info
  rm -v dir
  for f in *
    do install-info $f dir 2>/dev/null
  done
popd
```

8.68.2. Texinfo の構成

インストールプログラム: info, install-info, makeinfo (texi2any へのリンク), pdftexi2dvi, pod2texi, texi2any, texi2dvi, texi2pdf, texindex
インストールライブラリ: MiscXS.so, Parsetexi.so, XSParagraph.so (すべて /usr/lib/texinfo ディレクトリ内)
インストールディレクトリ: /usr/share/texinfo, /usr/lib/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。

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。

pod2texi	Pod フォーマットを Texinfo フォーマットに変換します。
texi2any	Texinfo のソースファイルを他のさまざまなフォーマットに変換します。
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。

8.69. Vim-9.0.0228

Vim パッケージは強力なテキストエディターを提供します。

概算ビルド時間: 2.5 SBU
必要ディスク容量: 217 MB



Vim の代替ソフトウェア

もし Emacs、Joe、Nano など他のエディターを用いたい場合は <https://www.linuxfromscratch.org/blfs/view/11.2/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。

8.69.1. 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
```

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、`tester` ユーザーがソースツリーに書き込みできるようにします。

```
chown -Rv tester .
```

`tester` ユーザーによりテストを実行します。

```
su tester -c "LANG=en_US.UTF-8 make -j1 test" &> vim-test.log
```

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、上に示すように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。テストが成功すれば、ログファイルの最後に "ALL DONE" と表示されます。

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-9.0.0228` 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/vim90/doc /usr/share/doc/vim-9.0.0228
```

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 <https://www.linuxfromscratch.org/blfs/view/11.2/postlfs/vim.html> 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。

8.69.2. 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。他のエディター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

" Ensure defaults are set before customizing settings, not after
source $VIMRUNTIME/defaults.vim
let skip_defaults_vim=1

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

" End /etc/vimrc
EOF
```

`set nocompatible` と設定しておくことで 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` を設定しておくことで、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` パラメーターを指定すれば vim の文法ハイライト (syntax highlighting) 機能が有効になります。 `set mouse=` を指定すると chroot 環境やリモート接続時であってもマウスによるテキスト選択が適切になります。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ー上において 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。エミュレーター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ば <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/vim90/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。

8.69.3. 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), `rview` (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。
rvm	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	エディター。
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。

8.70. Eudev-3.2.11

Eudev 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: 83 MB

8.70.1. Eudev のインストール

Eudev をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --bindir=/usr/sbin \
            --sysconfdir=/etc \
            --enable-manpages \
            --disable-static
```

パッケージをコンパイルします。

```
make
```

テスト時に必要となるディレクトリを生成します。 その一部はインストールの際にも利用します。

```
mkdir -pv /usr/lib/udev/rules.d
mkdir -pv /etc/udev/rules.d
```

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
```

LFS 環境にて有用なカスタムルールやサポートファイルをインストールします。

```
tar -xvf ../udev-lfs-20171102.tar.xz
make -f udev-lfs-20171102/Makefile.lfs install
```

8.70.2. Eudev の設定

ハードウェアデバイスに関する情報は、`/etc/udev/hwdb.d` ディレクトリおよび `/usr/lib/udev/hwdb.d` ディレクトリに収容されています。 Eudev はこの情報をとりまとめて、バイナリデータベース `/etc/udev/hwdb.bin` を作成しています。 このデータベース初期化は以下により実現します。

```
udevadm hwdb --update
```

このコマンドはハードウェア情報が更新された際には必ず実行してください。

8.70.3. Eudev の構成

インストールプログラム: `udevadm, udevd`
インストールライブラリ: `libudev.so`
インストールディレクトリ: `/etc/udev, /usr/lib/udev, /usr/share/doc/udev-udev-lfs-20171102`

概略説明

`udevadm` 汎用的な udev 管理ツール。 `udev` デーモンの制御、Udev データベースからの情報提供、`uevent` 監視、`uevent` 完了待機、Udev 設定のテスト、指定デバイスへの `uevent` 起動などを行います。

`udev` ネットリンクソケット上の `udev` を検出するデーモンであり、デバイスを生成しそのイベントに応じた外部プログラムを実行します。

`libudev` udev デバイス情報へのインターフェースライブラリ。

`/etc/udev` Udev の設定ファイル、デバイスのパーミッション、デバイス名に対するルールを設定します。

8.71. Man-DB-2.10.2

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: 40 MB

8.71.1. Man-DB のインストール

Man-DB をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/man-db-2.10.2 \
            --sysconfdir=/etc \
            --disable-setuid \
            --enable-cache-owner=bin \
            --with-browser=/usr/bin/lynx \
            --with-vgrind=/usr/bin/vgrind \
            --with-grap=/usr/bin/grap \
            --with-systemdtmpfilesdir= \
            --with-systemdsystemunitdir=
```

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。

`--enable-cache-owner=bin`

システムワイドなキャッシュファイルの所有ユーザーを bin とします。

`--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 ブラウザーです。(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。

`--with-systemd...`

これらのパラメーターは systemd に関する不要なディレクトリやファイルはインストールしないようにします。

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

8.71.2. 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8.1 8 ビット man ページのキャラクターエンコーディング

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

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
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	ギリシア語 (el)	ISO-8859-7
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



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。

8.71.3. 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`, `man-recode`, `mandb`, `manpath`, `whatis`
 インストールライブラリ: `libman.so`, `libmandb.so` (いずれも `/usr/lib/man-db` ディレクトリ内)
 インストールディレクトリ: `/usr/lib/man-db`, `/usr/libexec/man-db`, `/usr/share/doc/man-db-2.10.2`

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。
`man-recode` マニュアルページを別のエンコーディングに変換します。
`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。

libman man に対しての実行時のサポート機能を提供します。
libmandb man に対しての実行時のサポート機能を提供します。

8.72. Procps-ng-4.0.0

Procps-ng 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: 19 MB

8.72.1. Procps-ng のインストール

procps-ng をコンパイルするための準備をします。

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/procps-ng-4.0.0 \
            --disable-static \
            --disable-kill
```

configure オプションの意味

`--disable-kill`

本スイッチは `kill` コマンドをビルドしないようにします。このコマンドは `Util-linux` パッケージにてインストールされます。

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する場合は、以下を実行します。

make check

ホストディストリビューション上において、特定のアプリケーション（たとえば JVM や ウェブブラウザ）が独自のメモリ割り当てを行っている場合に、`free with commit` という名前のテストが失敗します。

パッケージをインストールします。

make install

8.72.2. Procps-ng の構成

インストールプログラム: `free`, `pgrep`, `pidof`, `pkill`, `pmap`, `ps`, `pwdx`, `slabtop`, `sysctl`, `tload`, `top`,
`uptime`, `vmstat`, `w`, `watch`
インストールライブラリ: `libproc-2.so`
インストールディレクトリ: `/usr/include/procps`, `/usr/share/doc/procps-ng-4.0.0`

概略説明

<code>free</code>	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。
<code>pgrep</code>	プロセスの名前などの属性によりプロセスを調べます。
<code>pidof</code>	指定されたプログラムの PID を表示します。
<code>pkill</code>	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。
<code>pmap</code>	指定されたプロセスのメモリマップを表示します。
<code>ps</code>	現在実行中のプロセスを一覧表示します。
<code>pwait</code>	実行前にプロセスの完了を待ちます。
<code>pwdx</code>	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。
<code>slabtop</code>	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。
<code>sysctl</code>	システム稼動中にカーネル設定を修正します。
<code>tload</code>	システムの負荷平均 (load average) をグラフ化して表示します。
<code>top</code>	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。
<code>uptime</code>	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。
<code>vmstat</code>	仮想メモリの統計情報を表示します。ここではプロセス、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-2 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。

8.73. Util-linux-2.38.1

Util-linux パッケージはさまざま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: 1.0 SBU
必要ディスク容量: 283 MB

8.73.1. Util-linux のインストール

Util-linux をコンパイルするための準備をします。

```
./configure ADJTIME_PATH=/var/lib/hwclock/adjtime \
--bindir=/usr/bin \
--libdir=/usr/lib \
--sbindir=/usr/sbin \
--docdir=/usr/share/doc/util-linux-2.38.1 \
--disable-chfn-chsh \
--disable-login \
--disable-nologin \
--disable-su \
--disable-setpriv \
--disable-runuser \
--disable-pylibmount \
--disable-static \
--without-python \
--without-systemd \
--without-systemdsystemunitdir
```

--disable と --without のオプションは、LFS では必要のないパッケージ、あるいは他のパッケージのインストールによって不整合となったパッケージに対して出力される警告をなくします。

パッケージをコンパイルします。

```
make
```

必要なら root ユーザー以外にて、以下のようにテストスイートを実行します。



警告

root ユーザーによりテストスイートを実行すると、システムに悪影響を及ぼすことがあります。テストスイートを実行するためには、カーネルオプション CONFIG_SCSI_DEBUG が現環境にて有効であり、かつモジュールとしてビルドされていなければなりません。カーネルに組み込んでいるとブートできません。またテストを完全に実施するには BLFS での各種パッケージのインストールも必要になります。テストが必要であるなら、LFS システムを完成した後に、再起動したシステムにて以下を実行します。

```
bash tests/run.sh --srcdir=$PWD --builddir=$PWD
```

```
chown -Rv tester .
su tester -c "make -k check"
```

hardlinkテストは、カーネルオプションにおいて CONFIG_CRYPT_USER_API_HASH セットが設定されていない場合は失敗します。

パッケージをインストールします。

```
make install
```

8.73.2. Util-linux の構成

インストールプログラム: addpart, agetty, blkdiscard, blkid, blkzone, blockdev, cal, cfdisk, chcpu, chmem, choom, chrt, col, colcrt, colrm, column, ctrlaltdel, delpart, dmesg, eject, fallocate, fdisk, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hexdump, hwclock, i386, ionice, ipcmk, ipcrm, ipcs, irqtop, isosize, kill, last, lastb (last へのリンク), ldattach, linux32, linux64, logger, look, losetup, lsblk, lscpu, lsipc, lsirq, lslocks, lslogins, lsmem, lsns, mcookie, mesg, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, mountpoint, namei, nsenter, partx, pivot_root, prlimit, readprofile, rename, renice, resizepart, rev, rfskill, rtcwake, script, scriptlive, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swapon, swapoff (swapon へのリンク), swapon, switch_root, taskset, uclampset, ul, umount, uname26, unshare, utmpdump, uuid, uuidgen, uuidparse, wall, wdctl, whereis, wpefs, x86_64, zramctl

インストールライブラリ: libblkid.so, libfdisk.so, libmount.so, libsmartcols.so, libuuid.so

インストールディレクトリ: /usr/include/blkid, /usr/include/libfdisk, /usr/include/libmount, /usr/include/libsmartcols, /usr/include/uuid, /usr/share/doc/util-linux-2.38.1, /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。
blkdiscard	デバイス上のセクターを取り除きます。
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。
blkzone	指定されたブロックデバイスにおいてゾーンコマンドを実行します。
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。
chcpu	CPU の状態を変更します。
chmem	メモリを設定します。
choom	OOM-killer スコアを表示し調整します。
chrt	リアルタイムプロセスの属性を操作します。
col	逆改行 (reverse 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。
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。
eject	リムーバブルメディアをイジェクトします。
fallocate	ファイルのための領域を事前割り当てします。
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。
findcore	メモリコア内にあるファイル情報のページ数を調べます。
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) を行いません。
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) を行いません。
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。
fstrim	マウントされたファイルシステム上にて、利用されていないブロックを破棄します。
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ロック (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 スケジューラクラスとスケジューラ優先度を取得または設定します。
ipcmk	さまざまな 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。
irqtop	カーネルのインタラプトカウンター情報を top(1) スタイルにより表示します。
isozsize	iso9660 ファイルシステムのサイズを表示します。
kill	プロセスに対してシグナルを送信します。
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。
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。
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。
lsblk	ブロックデバイスのすべて、あるいは指定されたものの情報を、木構造のような形式で一覧表示します。
lscpu	CPU アーキテクチャーの情報を表示します。
lsipc	システムに搭載されている IPC 機能の情報を表示します。
lsirq	カーネルのインタラプトカウンター情報を表示します。
lslocks	ローカルのシステムロックを一覧表示します。
lslogins	ユーザー、グループ、システムアカウントの情報を一覧表示します。
lsmem	オンライン状態にある利用可能なメモリ範囲を一覧表示します。
lsns	名前空間を一覧表示します。
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。
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。
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ープログラム。
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。

mountpoint	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。
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。
nsenter	他プロセスの名前空間にてプログラムを実行します。
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。
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。
prlimit	プロセスが利用するリソースの限界値を取得または設定します。
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。
resizepart	Linux カーネルに対してパーティションのリサイズを指示します。
rev	指定されたファイル内の行の並びを入れ替えます。
rkfill	ワイアレスデバイスの有効化、無効化を行うツール。
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。
scriptlive	タイミング情報を使って、セッションのタイプスクリプトを再実行します。
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ーを変更します。 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。
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。
swapon	スワップ領域の 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。
taskset	プロセスの CPU 親和性 (affinity) を表示または設定します。
uclampset	システムやプロセスの使用率クランプ属性を操作します。
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ー。
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。
uname26	setarch へのシンボリックリンク。
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。
utmpdump	指定されたログインファイルの内容を分かりやすい書式で表示します。
uudd	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。
uuidparse	ユニークな識別子を解析するためのユーティリティ。
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。
wdctl	ハードウェアの watchdog ステータスを表示します。
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ーをデバイスから消去します。
x86_64	setarch へのシンボリックリンク。
zramctl	zram (compressed ram disk) デバイスを初期化し制御するためのプログラム。

<code>libblkid</code>	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。
<code>libfdisk</code>	パーティションテーブルを操作する処理ルーチンを提供します。
<code>libmount</code>	ブロックデバイスのマウントとアンマウントに関する処理ルーチンを提供します。
<code>libsmartcols</code>	タブラー形式 (tabular form) による画面出力を補助する処理ルーチンを提供します。
<code>libuuid</code>	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。

8.74. E2fsprogs-1.46.5

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: 回転式ディスクで 4.4 SBU、SSD で 1.2 SBU
必要ディスク容量: 94 MB

8.74.1. 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      \  
              --sysconfdir=/etc  \  
              --enable-elf-shlibs \  
              --disable-libblkid  \  
              --disable-libuuid   \  
              --disable-uidd      \  
              --disable-fsck
```

configure オプションの意味

--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、uidd 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 パッケージによって、より最新のものが入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
```

u_direct_io という 1 つのテストが、システムによっては失敗する場合があります。

パッケージをインストールします。

```
make install
```

不要なスタティックライブラリを削除します。

```
rm -fv /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
```

8.74.2. E2fsprogs の構成

インストールプログラム:	badblocks, chattr, compile_et, debugfs, dumpe2fs, e2freefrag, e2fsck, e2image, e2label, e2mmpstatus, e2scrub, e2scrub_all, e2undo, e4crypt, e4defrag, filefrag, fsck.ext2, fsck.ext3, fsck.ext4, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mklost+found, resize2fs, tune2fs
インストールライブラリ:	libcom_err.so, libe2p.so, libext2fs.so, libss.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/lib/e2fsprogs, /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	フリースペースのフラグメント情報を表示します。
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。
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。
e2mmpstatus	ext4 ファイルシステムの MMP ステータスをチェックします。
e2scrub	マウントされている ext[234] ファイルシステムの内容をチェックします。
e2scrub_all	マウントされている ext[234] ファイルシステムのエラーをチェックします。
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。
e4crypt	ext4 ファイルシステムの暗号化ユーティリティ。
e4defrag	ext4 ファイルシステムにたいするオンラインのデフラグプログラム。
filefrag	特定のファイルがどのようにデフラグ化して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。
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。
mklost+found	ext2 ファイルシステム上に lost+found ディレクトリを作成します。これはそのディレクトリ内にあらかじめディスクブロックを割り当てておくことにより e2fsck コマンド処理を軽減させます。
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ーを調整します。
libcom_err	共通的なエラー表示ルーチン。

<code>libe2p</code>	<code>dumpe2fs</code> 、 <code>chattr</code> 、 <code>lsattr</code> の各コマンドが利用します。
<code>libext2fs</code>	ユーザーレベルのプログラムが <code>ext2</code> 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。
<code>libss</code>	<code>debugfs</code> コマンドが利用します。

8.75. Syslogd-1.5.1

syslogd 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: 0.6 MB

8.75.1. Syslogd のインストール

特定の条件において klogd がセグメンテーションフォールトを起こすため、この問題を修正します。また古いプログラム構造を修正します。

```
sed -i '/Error loading kernel symbols/{n;n;d}' ksym_mod.c
sed -i 's/union wait/int/' syslogd.c
```

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
```

8.75.2. Syslogd 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
```

8.75.3. Syslogd 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。

8.76. Sysvinit-3.04

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: 2.7 MB

8.76.1. Sysvinit のインストール

まず他のパッケージによりインストールされるプログラムを取り除いたり、出力メッセージの明確化、警告メッセージの修正などを行うパッチを適用します。

```
patch -Np1 -i ../sysvinit-3.04-consolidated-1.patch
```

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
```

8.76.2. Sysvinit の構成

インストールプログラム: bootlogd, fstab-decode, halt, init, killall5, poweroff (halt へのリンク),
reboot (halt へのリンク), runlevel, shutdown, telinit (init へのリンク)

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 <code>-h</code> 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 <code>/var/log/wtmp</code> 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、設定ファイルにて指定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せん。つまり本コマンドを実行した親シェルは停止しません。
poweroff	カーネルに対してシステムの停止を指示し、コンピューターの電源を切ります。(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 <code>/run/utmp</code> 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。
telinit	init に対してランレベルの変更を指示します。

8.77. 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。

8.78. ストリップ

本節での作業を行うかどうかは任意です。対象ユーザーがプログラマーではなく、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や不要シンボルを削除しても構いません。そうすれば 2 GB 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。

`strip` コマンドに `--strip-unneeded` オプションをつけて実行すると、バイナリやライブラリからデバッグシンボルをすべて削除します。そして (スタティックライブラリ向けの) リンカーや (動的リンクバイナリあるいは共有ライブラリ向けの) ダイナミックリンカーにとって不要なシンボルテーブル項目もすべて削除します。

ライブラリのいくつかについて、デバッグシンボルを持つような別ファイルを生成します。このデバッグ情報を必要とするのは BLFS における `valgrind` または `gdb` の縮退テストを実施するのに必要であるからです。

なお `strip` は、処理しているバイナリファイルやライブラリファイルを上書きします。そのファイルにあるコードやデータを利用しているプロセスは、これによってクラッシュすることがあります。仮に `strip` 自体を実行しているプロセスがその影響を受けたとすると、ストリップ最中のバイナリやライブラリは壊れてしまうかもしれません。これが起きると、システムが完全に利用不能となりかねません。これを避けるため、ライブラリやバイナリのいくつかを `/tmp` にコピーして、そこでストリップした上で、`install` コマンドを使って、元の場所にインストールし直すことにします。ここで `install` コマンドを利用する意味については、「アップグレードに関する問題」に示す関連項目を参照してください。



注記

ELF ロードナーの名前は、64 ビットシステムでは ld-linux-x86-64.so.2、32 ビットシステムでは ld-linux.so.2 です。後述の手順では、現行のアーキテクチャーに合わせて適切な名前を選ぶようになっています。ただし「g」で終わるものは除いています。そのようなものはすでにコマンド実行されているからです。

```
save_usrlib="$(cd /usr/lib; ls ld-linux*[^g])
    libc.so.6
    libthread_db.so.1
    libquadmath.so.0.0.0
    libstdc++.so.6.0.30
    libitm.so.1.0.0
    libatomic.so.1.2.0"

cd /usr/lib

for LIB in $save_usrlib; do
    objcopy --only-keep-debug $LIB $LIB.dbg
    cp $LIB /tmp/$LIB
    strip --strip-unneeded /tmp/$LIB
    objcopy --add-gnu-debuglink=$LIB.dbg /tmp/$LIB
    install -vm755 /tmp/$LIB /usr/lib
    rm /tmp/$LIB
done

online_usrbin="bash find strip"
online_usrlib="libbfd-2.39.so
    libhistory.so.8.1
    libncursesw.so.6.3
    libm.so.6
    libreadline.so.8.1
    libz.so.1.2.12
    $(cd /usr/lib; find libnss*.so* -type f)"

for BIN in $online_usrbin; do
    cp /usr/bin/$BIN /tmp/$BIN
    strip --strip-unneeded /tmp/$BIN
    install -vm755 /tmp/$BIN /usr/bin
    rm /tmp/$BIN
done

for LIB in $online_usrlib; do
    cp /usr/lib/$LIB /tmp/$LIB
    strip --strip-unneeded /tmp/$LIB
    install -vm755 /tmp/$LIB /usr/lib
    rm /tmp/$LIB
done

for i in $(find /usr/lib -type f -name \*.so* ! -name \*dbg) \
    $(find /usr/lib -type f -name \*.a) \
    $(find /usr/{bin,sbin,libexec} -type f); do
    case "$online_usrbin $online_usrlib $save_usrlib" in
        *$(basename $i)* )
            ;;
        * ) strip --strip-unneeded $i
            ;;
    esac
done

unset BIN LIB save_usrlib online_usrbin online_usrlib
```


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。

8.79. 仕切り直し

テストを通じて生成された不要なファイル等を削除します。

```
rm -rf /tmp/*
```

また /usr/lib ディレクトリと /usr/libexec ディレクトリには、拡張子が .la であるファイルがいくつかインストールされます。これは "libtool アーカイブ" ファイルというものであり、すでに説明しているように、これはスタティックライブラリとリンクする際に利用します。これらはダイナミック共有ライブラリを用いるとき、そして特に autotools 以外のビルドシステムを利用するときには不要であり、潜在的には支障を及ぼします。削除する場合は以下を実行します。

```
find /usr/lib /usr/libexec -name \*.la -delete
```

libtool アーカイブファイルについての詳細は BLFS の節 "About Libtool Archive (.la) files" を参照してください。

第 6 章 と 第 7 章 においてビルドしたコンパイラーは、部分的にしかインストールしていませんが、これ以降は必要としません。そこで以下によって削除します。

```
find /usr -depth -name $(uname -m)-lfs-linux-gnu\* | xargs rm -rf
```

最後に、本章のはじめに生成した 'tester' ユーザーアカウントを削除します。

```
userdel -r tester
```

第9章 システム設定

9.1. はじめに

Linux システムの起動時には実行されるタスクがいくつかあります。 実質的および仮想的なファイルシステムのマウント、デバイスの初期化、スワップ有効化、ファイルシステムの整合チェック、スワップパーティションなどのマウント、システムクロックの設定、ネットワーク起動、システムデーモンの起動、そしてユーザー指定によるタスクの起動です。

この処理過程は適正な順序により実行されることが必要ですが、同時に出来るだけ速く処理されることも必要になります。

9.1.1. System V

System V は古くからあるブートシステムであり、Unix や Unix ライクである Linux において 1983年頃より活用されています。 小さなプログラム `init` があり、これが `login` のような基本的なプログラムを (`getty` を通じて) 設定しスクリプトを実行します。 そのスクリプトは通常 `rc` と命名され、他のスクリプトの実行を制御します。 こうしてシステムの初期化を行うタスクが処理されます。

`init` プログラムは `/etc/inittab` ファイルにより制御されます。 そしてユーザーが設定可能なランレベルを設定します。 LFS においては以下のものが利用されます。

```
0 — 停止 (halt)
1 — シングルユーザーモード
2 — ユーザー定義
3 — フルマルチユーザーモード
4 — ユーザー定義
5 — フルマルチユーザーモード、ディスプレイマネージャーあり
6 — 再起動 (reboot)
```

通常のデフォルトランレベルは 3 か 5 です。

長所

- 確立されていて、十分に理解されているシステムであること。
- 容易にカスタマイズ可能であること。

短所

- おそらく起動が遅いこと。 中程度の処理性能による LFS システムの場合、最初のカーネルメッセージの出力からログインプロンプトまでの処理時間は 8 から 12 秒程度。 ログイン後のネットワーク接続の確立に 2 秒ほど要する。
- 起動タスクがすべて順番に行われること。 これは前項にも関係する。 ファイルシステムのチェックなどの処理に処理遅延があったとすると、起動処理全体の処理時間を遅らせることになる。
- コントロールグループ (control groups; cgroups) やユーザーごとの適正なスケジューリング共有といった、最新機能には直接対応していないこと。
- スクリプト追加にあたっては手作業を要し、固定的な順序を考慮しないとイケないこと。

9.2. LFS-ブートスクリプト-20220723

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。システム起動方法のカスタマイズに必要な設定や手順については以降の節で説明します。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 244 KB

9.2.1. 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
```

9.2.2. 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, functions, halt, ifdown, ifup, localnet, modules, mountfs, mountvirtfs, network, rc, reboot, sendsignals, setclock, ipv4-static, swap, sysctl, sysklogd, template, udev, udev_retry
インストールディレクトリ: /etc/rc.d, /etc/init.d (シンボリックリンク), /etc/sysconfig, /lib/services, /lib/lsb (シンボリックリンク)

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 /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。/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。
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します。
ifup	ネットワークデバイスを初期化します。
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。
mountvirtfs	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。
ipv4-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 /run/udev から /etc/udev/rules.d へコピーします。

9.3. デバイスとモジュールの扱いについて

第 8 章の `eudev` のビルドを通じて `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。実際に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ず。

`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、`devtmpfs` ファイルシステム上に保存されます。（`devtmpfs` 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。

9.3.1. 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、ユーザー空間にて開発することが可能になったわけです。

9.3.2. Udev の実装

9.3.2.1. 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を用いるべきなのか。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ーの場合は、対象のオブジェクトをカーネルが検出し、そのオブジェクトを `sysfs`（内部的には `devtmpfs`）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ーの場合は、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、ドライバーによって `sysfs` に登録されたデータは、ユーザー空間のプロセスと（デバイスノードの修正を含む）さまざまな処理を行う `udev` にて利用可能となります。

9.3.2.2. デバイスノードの生成

デバイスファイルはカーネルによって、`devtmpfs` ファイルシステム上に作り出されます。デバイスノードを登録しようとするドライバーは（デバイスコア経由で）`devtmpfs` を通じて登録を行います。`devtmpfs` のインスタンスが `/dev` 上にマウントされると、デバイスノードには固定的な名称、パーミッション、所有者の情報が設定され生成されます。

この後にカーネルは `udev` に対して `uevent` を送信します。`udev` は、`/etc/udev/rules.d`、`/usr/lib/udev/rules.d`、`/run/udev/rules.d` の各ディレクトリ内にあるファイルの設定ルールに従って、デバイスノードに対するシンボリックリンクを生成したり、パーミッション、所有者、グループの情報を変更したり、内部的な `udev` データベースの項目を修正したりします。

上の三つのディレクトリ内にて指定されるルールは番号づけされており、三つのディレクトリの内容は一つにまとめられます。デバイスノードの生成時に `udev` がそのルールを見つけ出せなかった時は、`devtmpfs` が利用される際の初期のパーミッションと所有者の情報のままとなります。

9.3.2.3. 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が、`udev` が提供するデフォルトの生成規則によって `udev` 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が。) 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となります。

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。

9.3.2.4. 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 `udev` が取り扱うこととなります。

9.3.3. 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。

9.3.3.1. 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-5.19.2 における `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-5.19.2 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。

9.3.3.2. 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う問題

「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、対応する `/etc/modprobe.d/<filename>.conf` ファイル内にて「`softdep`」の記述行を加えることで実現します。

```
softdep snd-pcm post: snd-pcm-oss
```

「softdep」コマンドは `pre:` を付与することもでき、あるいは `pre:` と `post:` の双方を付与することもできます。その記述方法や機能に関する詳細は `man` ページ `modprobe.d(5)` を参照してください。

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、`modules` 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。

9.3.3.3. 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。

9.3.3.4. 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。

9.3.3.5. 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。

9.3.3.6. 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。まずドライバー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、カーネルドライバーが `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ーであれば当たり前のことです。したがって `/usr/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、`udev` によって `/dev` にコピーされます。

9.3.3.7. 再起動後にデバイスの命名順がランダムになってしまう問題

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測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さまざまな `*_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「デバイスの管理」や「全般的なネットワークの設定」を参照してください。

9.3.4. 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>

9.4. デバイスの管理

9.4.1. ネットワークデバイス

Udev はデフォルトにおいて、ネットワークデバイスの名前づけを、ファームウェア/BIOS データや物理的特性、つまりバス、スロット、MACアドレスに基づいて取り決めます。このような命名規則とする目的は、複数のネットワークデバイスの命名を正確に行うためであり、検出した順番に命名することがないようにするためです。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ーにおいて、Intel 製が eth0、Realtek 製が eth1 となったとします。システムを再起動した際には、番号割り振りが逆転することもあります。

新たな命名スキーマでは、ネットワークデバイス名が例えば enp5s0 や wlp3s0 といったものになります。もしこの命名規則を望まない場合は、従来の命名規則とすることもできます。またはカスタムスキーマを定義することもできます。

9.4.1.1. カーネルコマンドラインによる持続的命名の回避

従来の命名スキーマ、例えば eth0、eth1 といったものは、カーネルコマンドラインに **net.ifnames=0** を加えることで利用できます。この設定は、イーサネットデバイスをただ一つしか持たないシステムでは適正なものとなります。一方ラップトップでは、eth0 と wlan0 といった複数のイーサネット接続が利用されることが多いものです。カーネルコマンドラインは GRUB の設定ファイルにて設定できます。詳しくは「GRUB 設定ファイルの生成」を参照してください。

9.4.1.2. Udev カスタムルールの生成

命名スキーマは udev カスタムルールを生成することによってカスタマイズが可能です。Udev には初期ルールを生成するスクリプトが含まれています。このルールを生成するには以下を実行します。

```
bash /usr/lib/udev/init-net-rules.sh
```

そして /etc/udev/rules.d/70-persistent-net.rules ファイルを参照し、どういった名前によりネットワークデバイスが定められているかを確認します。

```
cat /etc/udev/rules.d/70-persistent-net.rules
```



注記

ネットワークカードに対して手動で MAC アドレスを割り当てた場合、あるいは Qemu や Xen のような仮想環境における場合においては、ネットワークルールファイルが生成されないことがあります。これはアドレスの割り当てが確定されないためです。こういった場合は本方法を利用することはできません。

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。) またドライバーが検出できている場合には、カッコ書きでドライバー名も示されます。ハードウェア ID もドライバー名も、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ーがないためです。) 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ー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。
- **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。

9.4.2. 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リン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。このモードで作りに出されるルールは 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

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となります。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、`/etc/udev/rules.d/83-cdrom-symlinks.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 -e 's/"write_cd_rules"/"write_cd_rules mode"/' \  
-i /etc/udev/rules.d/83-cdrom-symlinks.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が、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。

9.4.3. 重複するデバイスの取り扱い方

「デバイスとモジュールの扱いについて」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変わることがあります。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「全般的なネットワークの設定」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なりありません。（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。

9.5. 全般的なネットワークの設定

9.5.1. 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

ネットワークインターフェースの起動、停止は `/etc/sysconfig/` ディレクトリ配下のファイルによって決まります。このディレクトリには、設定を行いたい各ネットワークインターフェースに対するファイル `ifconfig.xyz` を準備します。「xyz」はネットワークカードを指します。通常はインターフェース名（例えば `eth0`）を用います。そしてこのファイル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ます。ファイルベース名は `ifconfig` とすることが必要です。

注記

前節に示した手順を実施しなかった場合、udev は、システムの物理的な特性に従った `enp2s1` などのような名称をネットワークカードインターフェースに割り当てます。インターフェース名がよく分からない場合は、システム起動した後に `ip link` または `ls /sys/class/net` を実行すれば確認できます。

インターフェース名は、システム上で起動している udev デーモンの実装や設定に依存します。LFS における udev デーモン（「Eudev-3.2.11」においてインストール）は、LFS システムを起動させるまでは動作しません。したがってホストディストリビューションにおいて各コマンドを実行しても、LFS 上において用いられるインターフェース名が何であるのかは特定できません。それは chroot 環境内においても同じことです。

以下のコマンドは、`eth0` デバイスに対して固定 IP アドレスを設定するファイルを生成する例です。

```
cd /etc/sysconfig/
cat > ifconfig.eth0 << "EOF"
ONBOOT=yes
IFACE=eth0
SERVICE=ipv4-static
IP=192.168.1.2
GATEWAY=192.168.1.1
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

イタリックで示す変数の値は、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」に設定した場合、システム起動時に System V ネットワークスクリプトがネットワークインターフェースカード（network interface card; NIC）を起動します。「yes」以外に設定すると、ネットワークスクリプトからの NIC の起動がなくなり、NIC は自動では起動しくなくなります。ネットワークインターフェースは `ifup` や `ifdown` といったコマンドを使って、起動や停止を行うことができます。

`IFACE` 変数は、インターフェース名を定義します。例えば `eth0` といったものです。これはネットワークデバイスの設定を行うすべてのファイルにて必要な定義です。ファイル拡張子もこの設定に合わせます。

`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 `/lib/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ばその 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 であるならば、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。これが省略された場合は、デフォルトの 24 が用いられます。

より詳しくは `ifup` の `man` ページを参照してください。

9.5.2. `/etc/resolv.conf` ファイルの生成

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なうには、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
```

`domain` ステートメントは省略するか、`search` ステートメントで代用することが可能です。詳しくは `resolv.conf` の `man` ページを参照してください。

`<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。

注記

Google Public IPv4 DNS アドレスは 8.8.8.8 と 8.8.4.4 です。

9.5.3. ホスト名の設定

システム起動時には `/etc/hostname` が参照されてシステムのホスト名が決定されます。

以下のコマンドを実行することで `/etc/hostname` ファイルを生成するとともに、ホスト名を設定します。

```
echo "<lfs>" > /etc/hostname
```

`<lfs>` の部分は、各システムにおいて定めたい名称に置き換えてください。ここでは完全修飾ドメイン名 (Fully Qualified Domain Name; FQDN) は指定しないでください。その情報は `/etc/hosts` ファイルにて行います。

9.5.4. `/etc/hosts` ファイルの設定

IPアドレス、完全修飾ドメイン名 (Fully Qualified Domain Name; FQDN)、エイリアスの各設定を `/etc/hosts` ファイルにて行います。その文法は以下ようになります。

```
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 192.168.1.1 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

127.0.0.1 localhost.localdomain localhost
127.0.1.1 <FQDN> <HOSTNAME>
<192.168.1.1> <FQDN> <HOSTNAME> [alias1] [alias2 ...]
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# End /etc/hosts
EOF
```

<192.168.1.1>, <FQDN>, <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）。エイリアスの記述は省略しても構いません。

9.6. System V ブートスクリプトの利用と設定

9.6.1. System V 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、コンピューター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。LFS において実装されるランレベルの詳細を以下に説明します。

- 0: コンピューターの停止
- 1: シングルユーザーモード
- 2: 拡張用として予約されています。拡張がなければ 3 と同じものとして扱われます。
- 3: マルチユーザーモード、ネットワークあり
- 4: 拡張用として予約されています。拡張がなければ 3 と同じものとして扱われます。
- 5: 4 と同様。通常 (GNOME の gdm や LXDE の lxdm のような) GUI ログインに用いられます。
- 6: コンピューターの再起動



注記

従来より、上のランレベル 2 は「ネットワークなしにおけるマルチユーザーモード」として定義されていました。ただしこれは相当以前の話として、シリアルポートを介して複数ユーザーがシステムにログインするケースだけを表しています。今日のコンピューター環境においてこれは意味をなしません。そこでここでは「拡張用の予約」と位置づけます。

9.6.2. Sysvinit の設定

カーネルの初期化にあたって最初に起動するプログラムは、コマンドラインから指定されるものか、あるいはデフォルトでは `init` です。このプログラムは初期設定ファイル `/etc/inittab` を読み込みます。そのファイルは以下のようにして生成します。

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

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc S

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:S06:once:/sbin/sulogin
s1:1:respawn:/sbin/sulogin

1:2345:respawn:/sbin/agetty --noclear 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
```

この初期化ファイルに関することは `inittab` の `man` ページにて説明されています。LFS において重要となるコマンドは `rc` です。初期化ファイルは `rc` コマンドに対してスクリプトの実行を指示します。実行されるスクリプトは `/etc/rc.d/rcS.d` ディレクトリにて `S` で始まるスクリプトです。そしてその後に `/etc/rc.d/rc?.d` ディレクトリにて、同じく `S` で始まるスクリプトも実行されます。ここで `?` は、初期化を行う際の数値を示します。

扱いやすさを考慮して、`rc` スクリプトは `/lib/lsb/init-functions` ディレクトリにあるライブラリ群を読み込む形にしています。このライブラリは、さらにオプションで設定ファイル `/etc/sysconfig/rc.site` を読み込みます。本節以降に説明している、各種の設定ファイルにおけるパラメーターは、上のファイルにて設定することもできます。上のファイルは、システム上のパラメーターを1つのファイルに集約して設定できるようになっています。

デバッグがしやすいように、各ライブラリの関数スクリプトは、すべて `/run/var/bootlog` にログを出力するようになっています。`/run` ディレクトリは `tmpfs` であることから、`/run/var/bootlog` ファイルはブート前後にて恒常的なファイルではありません。ただしブート処理の最後には、恒常的なファイルである `/var/log/boot.log` に情報が出力されます。

9.6.2.1. ランレベルの変更

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ーを再起動するには `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 `rcS.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` まだが割振られ、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 (`S` で始まる) と停止 (`K` で始まる) 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` といったさまざまなパラメーター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` パラメーター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` パラメーターをつけて呼び出されるということになります。

スクリプトに対するパラメーター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。

9.6.3. Udev ブートスクリプト

初期起動スクリプト `/etc/rc.d/init.d/udev` は `udev` を起動し、カーネルにより既に生成されている "コールドプラグ" のデバイスをすべて稼働させます。そしてすべてのルールが起動完了するのを待ちます。このスクリプトは `/sbin/hotplug` のデフォルトから `uevent` ハンドラーを取り除きます。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、`udev` は、カーネルが起動する `uevent` をネットワークソケット (`netlink socket`) 上で待ち受けます。

初期起動スクリプト `/etc/rc.d/init.d/udev_retry` は、サブシステムに対するイベントの再起動を行いません。そのサブシステムとはファイルシステムに依存するもので、`mountfs` が実行されるまでマウントされません。(特に `/usr` や `/var` がこれに該当します。) `mountfs` スクリプトの後にこのスクリプトが実行されるので、(イベントが再起動されるものであれば) 二度目には成功します。このスクリプトは `/etc/sysconfig/udev_retry` ファイルにより設定が可能で、コメントを除く記述項目はすべてサブシステム名を表わし、二度目の起動時のリトライ対象となります。(デバイスのサブシステムを知るには `udevadm info --attribute-walk <device>` を実行します。ここで `<device>` は、`/dev` や `/sys` から始まる絶対パスであり `/dev/sr0` や `/sys/class/rtc` などを表します。)

カーネルモジュールのロードや `udev` に関しては「モジュールのロード」を参照してください。

9.6.4. システムクロックの設定

`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を利用するかを伝えます。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 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 <https://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。



注記

CLOCKPARAMS と UTC パラメーターは `/etc/sysconfig/rc.site` ファイルにて設定することもできます。

9.6.5. Linux コンソールの設定

この節ではブートスクリプト `console` 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（あるいはこれと同等の設定が `rc.site` にない場合）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。さまざま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 `/usr/share/keymaps` ディレクトリや `/usr/share/consolefonts` 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 -n` 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「it」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。キーマップによっては同一名に対して重複した定義を持つものもあります。(cz とその変形が `qwerty/` と `qwertz/` にあり、es は `olpc/` と `qwerty/` に、trf は `fgGlod/` と `qwerty/` にあります) こういった場合には、適切なキーマップがロードされるように、親ディレクトリを必ず指定する必要があります (`qwerty/es` など)。

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ーマップ (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ーマップとして「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ーマップを利用します。したがって "-m" パラメーターには、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"
UNICODE="1"

# 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 \mathbb{A} を得るもの」があります。Linux-5.19.2 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です。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。

9.6.6. ブート時のファイル生成

ブート時にファイルを生成したいときがあります。例えば `/tmp/.ICE-unix` ディレクトリが必要であったとします。これは `/etc/sysconfig/createfiles` スクリプトに設定を行うことで実現できます。このファイルの書式は、デフォルト設定ファイル内にコメントとして埋め込まれているので参照してください。

9.6.7. Sysklogd スクリプトの設定

`sysklogd` スクリプトは System V の一連の初期化に際して `syslogd` プログラムを起動します。オプション `-m 0` により実行され、`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 `/etc/sysconfig/rc.site` ファイルを新たに作るか既存のものを編集して、`SYSKLOGD_PARMS` 変数を必要な値に設定してください。例えばすべてのパラメーターを無効にする場合は、変数値をヌル値とします。

```
SYSKLOGD_PARMS=
```

詳しくは `man syslogd` を入力して `man` ページを参照してください。

9.6.8. rc.site ファイル

オプションファイル `/etc/sysconfig/rc.site` は、System V の各ブートスクリプトにて自動的に設定される内容を含んでいます。`/etc/sysconfig/` ディレクトリにおける `hostname`、`console`、`clock` の各ファイルにて値の設定を行うこともできます。関係する変数が、これらのファイルと `rc.site` の双方に存在する場合、スクリプトにて指定されたファイル内の値が優先されます。

`rc.site` では、起動時におけるその他の機能をカスタマイズするためのパラメーターも含まれています。変数 `IPROMPT` を設定すると、起動するブートスクリプトを選択することができます。この他のオプションについては、このファイル内にてコメントとして記述されています。このファイルのデフォルト版は以下のとおりです。

```
# rc.site
```

```

# Optional parameters for boot scripts.

# Distro Information
# These values, if specified here, override the defaults
#DISTRO="Linux From Scratch" # The distro name
#DISTRO_CONTACT="lfs-dev@lists.linuxfromscratch.org" # Bug report address
#DISTRO_MINI="LFS" # Short name used in filenames for distro config

# Define custom colors used in messages printed to the screen

# 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

# These values, if specified here, override the defaults
#BRACKET="\033[1;34m" # Blue
#FAILURE="\033[1;31m" # Red
#INFO="\033[1;36m" # Cyan
#NORMAL="\033[0;39m" # Grey
#SUCCESS="\033[1;32m" # Green
#WARNING="\033[1;33m" # Yellow

# Use a colored prefix
# These values, if specified here, override the defaults
#BMPREFIX=" "
#SUCCESS_PREFIX="${SUCCESS} * ${NORMAL} "
#FAILURE_PREFIX="${FAILURE}*****${NORMAL} "
#WARNING_PREFIX="${WARNING} *** ${NORMAL} "

# Manually set the right edge of message output (characters)
# Useful when resetting console font during boot to override
# automatic screen width detection
#COLUMNS=120

# Interactive startup
#IPROMPT="yes" # Whether to display the interactive boot prompt
#itime="3" # The amount of time (in seconds) to display the prompt

# The total length of the distro welcome string, without escape codes
#wlen=$(echo "Welcome to ${DISTRO}" | wc -c )
#welcome_message="Welcome to ${INFO}${DISTRO}${NORMAL}"

# The total length of the interactive string, without escape codes
#ilen=$(echo "Press 'I' to enter interactive startup" | wc -c )
#i_message="Press '${FAILURE}I${NORMAL}' to enter interactive startup"

# Set scripts to skip the file system check on reboot
#FASTBOOT=yes

# Skip reading from the console
#HEADLESS=yes

# Write out fsck progress if yes
#VERBOSE_FSCK=no

# Speed up boot without waiting for settle in udev
#OMIT_UDEV_SETTLE=y

```

```
# Speed up boot without waiting for settle in udev_retry
#OMIT_UDEV_RETRY_SETTLE=yes

# Skip cleaning /tmp if yes
#SKIPTMPCLEAN=no

# For setclock
#UTC=1
#CLOCKPARAMS=

# For consolelog (Note that the default, 7=debug, is noisy)
#LOGLEVEL=7

# For network
#HOSTNAME=mylfs

# Delay between TERM and KILL signals at shutdown
#KILLDELAY=3

# Optional syslogd parameters
#SYSKLOGD_PARMS="-m 0"

# Console parameters
#UNICODE=1
#KEYMAP="de-latin1"
#KEYMAP_CORRECTIONS="euro2"
#FONT="lat0-16 -m 8859-15"
#LEGACY_CHARSET=
```

9.6.8.1. ブートおよびシャットダウンスクリプトのカスタマイズ

LFS のブートスクリプト類により、システムの起動および終了が適正に行われます。ただし `rc.site` ファイルにおいては改善の余地があって、処理性能を向上させたり出力メッセージを調整したりすることができます。種々の設定は、上に示した `/etc/sysconfig/rc.site` ファイルへの変更により実現します。

- ブートスクリプト `udev` の起動中には `udev settle` の呼び出しが行われます。ただこの呼び出しは特定の場合において必要となるものであり、それはシステム上に存在するデバイスに依存します。単純なパーティション設定を行っていて、またイーサネットカードを1つのみ利用している場合には、ブート時に上のコマンドを実行する必要はないかもしれません。このコマンドの実行をスキップする場合は、変数の設定として `OMIT_UDEV_SETTLE=y` を記述します。
- ブートスクリプト `udev_retry` も同様に、デフォルトで `udev settle` を実行します。このコマンドはデフォルトでは、`/var` ディレクトリが個別にマウントされている時にのみ必要となります。それはクロックが `/var/lib/hwclock/adjtime` ファイルを必要とするためです。これ以外にも `udev` の処理を待つことが必要になるケースがありますが、本当に必要になることはまれです。変数の設定として `OMIT_UDEV_RETRY_SETTLE=y` を行えば、コマンドをスキップすることができます。
- デフォルトにおいてファイルシステムのチェックは、何も表示されることなく処理が行われるので、処理が遅延して行われているかのように見えます。 `fsck` による出力を有効とするには、変数の設定を `VERBOSE_FSCK=y` とします。
- 再起動時にはファイルシステムのチェック、つまり `fsck` の実行を完全に行う必要はないと考えられる場合もあります。そうであるなら、ファイル `/fastboot` を生成するか、`/sbin/shutdown -f -r now` というコマンドを実行します。一方、ファイルシステムのチェックを必ず行うのであれば、ファイル `/forcefsck` を生成するか、`shutdown` コマンドの実行において `-f` ではなく `-F` というパラメーターをつける方法があります。

変数の設定として `FASTBOOT=y` を行えば、ブート時において `fsck` を実行しないようにすることができます。この設定を恒常的に行うことは推奨されません。

- 通常 `/tmp` ディレクトリ内にあるファイルは、ブート時にすべて削除されます。ファイル数やディレクトリ数が膨大になっていた場合は、ブート処理が極端に時間を要することにもなります。変数の設定 `SKIPTMPCLEAN=y` を行うと、ファイルの削除が行われなくなります。
- シャットダウン時には `init` プログラムが稼働中のプログラム (`agetty` など) に対して `TERM` シグナルを送信し、一定時間 (デフォルトでは3秒) 待ちます。そして各プロセスに対して `KILL` シグナルを送信して再度待ちます。各プロセスが自身のスクリプト内にてシャットダウンしないようであれば `sendsignals` スクリプトにて上の処理が繰り返

されます。init が起動するまでの時間は、パラメーターにより制御することができます。例えば init の遅延を無くす場合は、シャットダウンまたはリブート時のコマンドに `-t0` パラメーターを与えます。（つまり `/sbin/shutdown -t0 -r now` といったコマンド実行とします。）`sendsignals` スクリプトの遅延を無くすには、パラメーターの設定を `KILLDELAY=0` とします。

9.7. 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さまざまに制御します。`/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。
- キャラクターを英字、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ーマップ (`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
```

キャラクター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ー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、「`Glibc-2.36`」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。

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.utf8」(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 ビットキャラクターセットを用います。もし最上位ビットがセットされたキャラクター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オ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、プログラムによってはサポートしていないものもあります。この問題については <https://www.linuxfromscratch.org/blfs/view/11.2/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。

9.8. `/etc/inputrc` ファイルの生成

`inputrc` ファイルは `readline` ライブラリに対する設定ファイルです。この `Readline` ライブラリは、ユーザーが端末から文字列入力を行う際の編集機能を提供するものです。キーボード入力内容は所定の処理動作に変換され解釈されます。`readline` ライブラリは `bash` をはじめとする各種シェルや他の多くのアプリケーションにおいて利用されています。

ユーザー固有の機能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 8-bit 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
```

9.9. /etc/shells ファイルの生成

`shells` ファイルには、システム上でのログインシェルを記述します。各アプリケーションはこのファイルを参照して、シェルが適切であるかどうかを判別します。各シェルの指定は1行で行い、そのシェルのパスを記述します。パスはルートディレクトリ (/) を基準として記述します。

例えば一般ユーザーが自身のアカウントに対するログインシェルを `chsh` にしようとした場合、`chsh` が `shells` ファイルを参照します。シェルコマンド名が記述されていないならば、その一般ユーザーはシェルの変更ができません。

例えば GDM は `/etc/shells` ファイルが参照できない時には対話インターフェースの設定が出来ません。また FTP デーモンなどは、このファイルに記述されていないシェルを用いてのユーザーアクセスを拒否するのが通常です。こういったアプリケーションのためにこのファイルが必要となります。

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

/bin/sh
/bin/bash

# End /etc/shells
EOF
```

第10章 LFS システムのブート設定

10.1. 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 `/etc/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 のブートローダー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ーを選択できるようにします。

10.2. `/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      nosuid,noexec,nodev 0     0
sysfs         /sys             sysfs     nosuid,noexec,nodev 0     0
devpts        /dev/pts         devpts    gid=5,mode=620    0     0
tmpfs         /run             tmpfs     defaults          0     0
devtmpfs      /dev             devtmpfs  mode=0755,nosuid  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 `sda2`、`sda5`、`ext4` といったものです。上記各行の6項目の記述内容については `man 5 fstab` により確認してください。

MS-DOS や Windows において利用されるファイルシステム（つまり `vfat`、`ntfs`、`smbfs`、`cifs`、`iso9660`、`udf` など）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、特別なマウントオプション「`utf8`」の指定が必要になります。UTF-8 以外のロケールの場合 `iocharset` オプションには、文字ロケールと同じ値を設定することが必要であり、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ーセット定義（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。ただし `iocharset=utf8` というオプション指定によって文字ロケールを UTF-8 とした場合、ファイルシステムの英大文字小文字は区別されるようになります。これを避けるのであれば、`iocharset=utf8` ではなく特別なオプション `utf8` を指定します。`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ようになります。

```
noauto,user,quiet,showexec,codepage=866,iocharset=koi8r
```

`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,codepage=866,utf8
```

`iocharset` オプションは `iso8859-1` に対してのデフォルト設定です。（その場合、ファイルシステムの英大文字小文字は区別されません。）`utf8` オプションは、ファイル名称が UTF-8 ロケール内にて正しく認識されるように、カーネルが UTF-8 ロケールに変換して取り扱うことを指示するものです。

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ー (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。

10.3. Linux-5.19.2

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 - 130.0 SBU (一般的には 12 SBU 程度)
 必要ディスク容量: 1200 - 8800 MB (一般的には 1700 MB 程度)

10.3.1. 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ば、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。

カーネルオプションの設定方法にはいくつかあります。通常は以下に示すように、メニュー形式のインターフェースを通じて行います。

```
make menuconfig
```

追加する make 環境変数の意味:

```
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。\$LC_ALL あるいは \$LC_CTYPE の値を設定することもできます。

```
make menuconfig
```

これは ncurses によるメニュー形式のインターフェースを起動します。これ以外の (グラフィカルな) インターフェースについては make help を入力して確認してください。

カーネルの設定方法に関する一般的な情報が <https://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。 <https://www.linuxfromscratch.org/blfs/view/11.2/longindex.html#kernel-config-index> を参照してください。さらに詳しくカーネルの構築や設定を説明している <http://www.kroah.com/1kn/> もあります。



注記

カーネル設定を行うにあたって、分かりやすいやり方として `make defconfig` を実行する方法があります。これを実行することで基本的な設定がなされ、現在のシステム構成が考慮された、より良い設定が得られるかもしれません。

以下の機能項目についての有効、無効、設定状況を確認してください。不適切である場合にはシステムが正常動作しなかったり起動できなかったりするかもしれません。

```
General setup -->
  [ ] Compile the kernel with warnings as errors [CONFIG_WERROR]
  < > Enable kernel headers through /sys/kernel/kheaders.tar.xz [CONFIG_IKHEADERS]
Device Drivers ---->
  Graphics support ---->
    Frame buffer Devices ---->
      [*] Support for frame buffer devices -----
  Generic Driver Options ---->
    [ ] Support for uevent helper [CONFIG_UEVENT_HELPER]
    [*] Maintain a devtmpfs filesystem to mount at /dev [CONFIG_DEVTMPFS]
    [*] Automount devtmpfs at /dev, after the kernel mounted the rootfs [CONFIG_DEVTMPFS]
```

64 ビットシステムの構築時は、追加機能をいくらか有効にしてください。menuconfig を利用している場合、初めに `CONFIG_PCI_MSI` を有効にして、その後に `CONFIG_IRQ_REMAP`、`CONFIG_X86_X2APIC` を有効にします。こうするのは、依存するオプションが選択されていないと、特定のオプションが現れてこないからです。

```
Processor type and features ---->
  [*] Support x2apic [CONFIG_X86_X2APIC]
Memory Management options ---->
  [ ] Enable userfaultfd() system call [CONFIG_USERFAULTFD]
Device Drivers ---->
  [*] PCI Support ----> [CONFIG_PCI]
    [*] Message Signaled Interrupts (MSI and MSI-X) [CONFIG_PCI_MSI]
  [*] IOMMU Hardware Support ----> [CONFIG_IOMMU_SUPPORT]
  [*] Support for Interrupt Remapping [CONFIG_IRQ_REMAP]
```

システムに特定の機能性が必要になれば、それだけ多くのオプションが必要となります。例えば BLFS パッケージにて必要となるオプションについては BLFS Index of Kernel Settings (<https://www.linuxfromscratch.org/blfs/view/11.2/longindex.html#kernel-config-index>) を参照してください。



注記

ホストが UEFI を利用していて、これを使って LFS システムのブートを行いたい場合は、BLFS ページに従って、カーネル設定を調整する必要があります。

上の設定項目の説明

Compile the kernel with warnings as errors

これを設定すると、カーネル開発者が採用するコンパイラーや設定と異なる場合に、カーネルビルドエラーとなる場合があります。

Enable kernel headers through /sys/kernel/kheaders.tar.xz

これは、カーネルビルドにあたって `cpio` を必要とします。cpio は LFS ではインストールしません。

Support for uevent helper

本項目を有効にすることで、デバイス管理を Udev/Eudev により行ないます。

Maintain a devtmpfs

本項目は、カーネルにより事前登録される自動化デバイスノードを生成します。これは Udev が動作していなくても行われます。Udev はその上で起動し、パーミッション管理やシンボリックリンクの追加を行います。Udev/Eudev を利用する場合には本項目を有効にすることが必要です。

Automount devtmpfs at /dev

これは、カーネルから見たデバイス情報を /dev 上にマウントするものです。init が起動される直前にルートファイルシステムに切り替えられます。

Support x2apic

64 ビット x86 プロセッサの x2APIC モードでのインタラプトコントローラーの実行をサポートします。64 ビット x86 システムにおいてはファームウェアが x2APIC を有効にすることがあります。ファームウェアによって x2APIC が有効である場合、カーネルにおいてこのオプションが無効であると、起動時にパニックを起こします。本オプションには効果がありません。またファームウェアによって x2APIC が無効であった場合、このオプションは影響を及ぼしません。

Enable userfaultfd() system call

このオプションを有効にすると、Linux-5.19.2 において解決されていないセキュリティぜい弱性が悪用される危険があります。ぜい弱性を避けるために、このオプションは無効にしてください。このシステムコールは LFS や BLFS のどこからも利用しません。

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-5.19.2` 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「デバイスとモジュールの扱いについて」や `linux-5.19.2/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.d(5)` も有用です。

カーネル設定においてモジュールの利用を無効にしているのであれば、ここで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。



注意

ホストシステムが独立した `/boot` パーティションを用いている場合はファイルをそこにコピーします。これを簡単に行うために、作業前に (`chroot` 前の) `/boot` をホストの `/mnt/lfs/boot` にバインドしておく方法があります。ホストシステムの `root` ユーザーとなって以下を実行します。

```
mount --bind /boot /mnt/lfs/boot
```

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 `vmlinuz` 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 アーキテクチャーの場合の例です。

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-5.19.2-lfs-11.2
```

`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 -iv System.map /boot/System.map-5.19.2
```

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あ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 -iv .config /boot/config-5.19.2
```

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-5.19.2
cp -r Documentation/* /usr/share/doc/linux-5.19.2
```

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。



注記

カーネルの設定は、BLFS をインストールしていくにつれて、設定を更新していかなければならないことが多々あります。一般にパッケージのソースは削除することが通常ですが、カーネルのソースに関しては、カーネルをもう一度新たにインストールするなら、削除しなくて構いません。

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-5.19.2 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くとした際に、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 (`/usr/include`) にあるヘッダーファイルは、必ず Glibc のコンパイル時のものでなければなりません。つまり「Linux-5.19.2 API ヘッダー」によってインストールされた、健全化 (sanitizing) したものです。したがって生のカーネルヘッダーや他のカーネルにて健全化されたヘッダー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。

10.3.2. Linux モジュールのロード順の設定

たいていの場合 Linux モジュールは自動的にロードされます。しかし中には特定の指示を必要とするものもあります。モジュールをロードするプログラム、`modprobe` または `insmod` は、そのような指示を行う目的で `/etc/modprobe.d/usb.conf` を利用します。USB ドライバー (`ehci_hcd`, `ohci_hcd`, `uhci_hcd`) がモジュールとしてビルドされていた場合には、それらを正しい順でロードしなければならず、そのために `/etc/modprobe.d/usb.conf` ファイルが必要となります。`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
```

10.3.3. Linux の構成

インストールファイル: `config-5.19.2`, `vmlinuz-5.19.2-lfs-11.2`, and `System.map-5.19.2`
 インストールディレクトリ: `/lib/modules`, `/usr/share/doc/linux-5.19.2`

概略説明

`config-5.19.2` カーネルの設定をすべて含みます。
`vmlinuz-5.19.2-lfs-11.2` Linux システムのエンジンです。コンピューターを起動した際には、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ーのハードウェア

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-5.19.2

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。

10.4. GRUB を用いたブートプロセスの設定



注記

UEFI サポートが有効なシステムにおいて UEFI を使って LFS をブートしたい場合は、本ページは読み飛ばしてください。そして BLFS ページ に示されている手順に従って、UEFI に対応するように GRUB 設定を行ってください。

10.4.1. はじめに



警告

GRUB の設定を誤ってしまうと、CD-ROM や USB 起動ドライブのような他のデバイスからもブートできなくなってしまいます。読者の LFS システムをブート可能とするためには、本節の内容は必ずしも必要ではありません。読者が利用している現在のブートローダー、例えば Grub-Legacy, GRUB2, LILO などの設定を修正することが必要かもしれません。

コンピューターが利用不能に（ブート不能に）なってしまいうこともあります。そんな事態に備えてコンピューターを「復旧 (rescue)」するブートディスクの生成を必ず行ってください。ブートデバイスを用意していない場合は作成してください。以降に示す手順を実施するために、必要に応じて BLFS ブックを参照し libisoburn にある **xorriso** をインストールしてください。

```
cd /tmp
grub-mkrescue --output=grub-img.iso
xorriso -as cdrecord -v dev=/dev/cdrw blank=as_needed grub-img.iso
```

10.4.2. GRUB の命名規則

GRUB ではドライブやパーティションに対して (hdm, m) といった書式の命名法を採用しています。n はハードドライブ番号、m はパーティション番号を表します。ハードドライブ番号はゼロから数え始めます。一方パーティション番号は、基本パーティションであれば 1 から、拡張パーティションであれば 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 sda1 は GRUB では (hd0, 1) と表記され、sdb3 は (hd1, 3) と表記されます。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。

10.4.3. 設定作業

GRUB は、ハードディスク上の最初の物理トラックにデータを書き出します。この領域は、どのファイルシステムにも属していません。ここに配置されているプログラムは、ブートパーティションにある GRUB モジュールにアクセスします。モジュールのデフォルト位置は /boot/grub/ です。

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 (200MB 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、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。

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、ブートトラックを構築します。



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當であるならコマンドを実行しないでください。例えばマスターブートレコード (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。

```
grub-install /dev/sda
```



注記

システムが UEFI を通じて起動されている時、grub-install は x86_64-efi ターゲットに対するファイルをインストールしようとします。しかしそのようなファイルは第 8 章にてインストールしていません。その場合は上のコマンドに対して `--target i386-pc` を追加してください。

10.4.4. GRUB 設定ファイルの生成

`/boot/grub/grub.cfg` ファイルを生成します。

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5

insmod ext2
set root=(hd0,2)

menuentry "GNU/Linux, Linux 5.19.2-lfs-11.2" {
    linux /boot/vmlinuz-5.19.2-lfs-11.2 root=/dev/sda2 ro
}
EOF
```



注記

GRUB にとってカーネルファイル群は、配置されるパーティションからの相対位置となります。したがって `/boot` パーティションを別に作成している場合は、上記の `linux` の行から `/boot` の記述を取り除いてください。また `set root` 行でのブートパーティションの指定も、正しく設定する必要があります。



注記

GRUB のパーティション指示子は、(USB サムデバイスといったリムーバブルディスクを含め) ディスクの加除によって変わることがあります。その加除が原因で起動に失敗することがありますが、それは `grub.cfg` において「古い」指示子を用いているからです。こういった問題を避けようとおもったら、パーティション指定にあたって GRUB 指定子を用いずに、パーティションやファイルシステムの UUID を用いることが考えられます。 `lsblk -o UUID,PARTUUID,PATH,MOUNTPOINT` を実行してください。ファイルシステムの UUID が UUID 列に示されます。またパーティションは PARTUUID 列に示されます。そうしたら `set root=(hdx,y)` の記述を `search --set=root --fs-uuid <カーネルがインストールされているファイルシステムの UUID>` に書き換え、同様に `root=/dev/sda2` を `root=PARTUUID=<LFS がビルドされたパーティションの UUID>` に書き換えます。

パーティションの UUID と、そのパーティション内のファイルシステムの UUID は全く異なります。オンラインから得られる情報において、`root=PARTUUID=<パーティション UUID>` ではなく `root=UUID=<ファイルシステム UUID>` を用いるように説明している場合があります。これを行うには `initramfs` が必要であり、これは LFS の範囲を超えるものです。

`/dev` 内のパーティションに対するデバイスノード名も変わります (GRUB 指定子に変更される可能性よりは低いです)。 `/etc/fstab` において記述するデバイスノードへのパスは、たとえば `/dev/sda1` を `PARTUUID=<パーティション UUID>` に置き換えることができます。これによりデバイスノード名が変更になった場合の、潜在的な起動エラーを回避することができます。

GRUB は大変強力なプログラムであり、ブート処理に際しての非常に多くのオプションを提供しています。これにより、各種デバイス、オペレーティングシステム、パーティションタイプに幅広く対応しています。さらにカスタマイズのためのオプションも多く提供されていて、グラフィカルなスプラッシュ画面、サウンド、マウス入力などについてカスタマイズが可能です。オプションの細かな説明は、ここでの手順説明の範囲を超えるため割愛します。



注意

`grub-mkconfig` というコマンドは、設定ファイルを自動的に生成するものです。このコマンドは `/etc/grub.d/` にある一連のスクリプトを利用しており、それまでに設定していた内容は失われることになります。

その一連のスクリプトは、ソースコードを提供しない Linux ディストリビューションにて用いられるのが主であるため、LFS では推奨されません。商用 Linux ディストリビューションをインストールする場合には、それらのスクリプトを実行する、ちょうど良い機会となるはずです。こういった状況ですから、`grub.cfg` のバックアップは忘れずに行うようにしてください。

第11章 作業終了

11.1. 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よう。

`/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ようになります。）以下のコマンドによりこのファイルを生成します。

```
echo 11.2 > /etc/lfs-release
```

インストールシステムの情報を表わした 2 つのファイルがあれば、これからシステムにインストールするパッケージにおいて利用していくことができます。 パッケージはバイナリ形式であっても、ビルドするものであってもかまいません。

1 つめのファイルは Linux Standards Base (LSB) の観点で、あなたのシステムがどのような状況にあるかを示すものです。 これを作成するために以下のコマンドを実行します。

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="11.2"
DISTRIB_CODENAME="<your name here>"
DISTRIB_DESCRIPTION="Linux From Scratch"
EOF
```

2 つめのファイルは、だいたい同じ情報を含むものですが、`systemd` やグラフィカルデスクトップ環境がこれを利用します。 これを作成するために以下のコマンドを実行します。

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="11.2"
ID=lfs
PRETTY_NAME="Linux From Scratch 11.2"
VERSION_CODENAME="<your name here>"
EOF
```

'`DISTRIB_CODENAME`' と '`VERSION_CODENAME`' の両項目に対しては、あなたのシステムを特定できるように適切に設定してください。

11.2. 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ーを取得しますか？ 以下のページ <https://www.linuxfromscratch.org/cgi-bin/lfscounter.php> にて、初めて構築した LFS のバージョンと氏名を登録して下さい。

それではシステムの再起動を行ないましょう。

11.3. 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ーを再起動しますが、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。 これ以降にさまざま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が、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が、以下はその際の構築例です。

- Lynx のようなテキストブラウザをインストールしておけば、仮想端末からでも BLFS ブックを簡単に参照しながらパッケージビルド作業を進めることができます。
- `make-ca` パッケージをインストールしていると、ローカルにおいて信頼されるアンカー証明書を構築できます。 そうすればリモートサーバー（たとえば HTTPS を用いたウェブサイト）が提供する SSL 証明書をシステムが検証できます。
- GPM パッケージをインストールすれば、仮想端末内にてコピーペースト操作を行うことができます。
- ネットワーク環境内にて固定 IP アドレスを用いることが適当ではない場合は、`dhcpcd` パッケージや `dhcpc` パッケージのクライアントモジュール部分を利用することが考えられます。

- `sudo` をインストールすれば、`root` ユーザー以外であっても、パッケージビルドとインストールを容易に行うことができます。
- 利用しやすい GUI 操作を通じてリモート接続を行いたい場合は `openssh` をインストールします。
- インターネット経由により簡単にファイル取得を行うために `wget` をインストールします。
- ワイヤレスのネットワークアクセスポイントに接続する場合は、`wpa_supplicant` をインストールしてください。
- 利用するハードウェア用のカーネルドライバが、それを適切に動作させるために何か別のファームウェアを利用している場合は、`firmwares` をインストールしてください。
- 最後に、以下に示す種々の設定ファイルが適切であるかどうかを確認します。
 - `/etc/bashrc`
 - `/etc/dircolors`
 - `/etc/fstab`
 - `/etc/hosts`
 - `/etc/inputrc`
 - `/etc/profile`
 - `/etc/resolv.conf`
 - `/etc/vimrc`
 - `/root/.bash_profile`
 - `/root/.bashrc`
 - `/etc/sysconfig/ifconfig.eth0`

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
umount -v $LFS/run
umount -v $LFS/proc
umount -v $LFS/sys
```

複数のパーティションを生成していた場合は、メインのパーティションをアンマウントする前に、個々のパーティションをアンマウントします。

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

LFS ファイルシステムそのものをアンマウントします。

```
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 ブートローダーが設定されているはずです。そのメニューには LFS 11.2 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行ってください。

11.4. 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。

- 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ー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。 <https://www.linuxfromscratch.org/blfs/view/11.2/>

- 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 <https://www.linuxfromscratch.org/hints/downloads/files/> 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V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。逆に訳語に適切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
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)
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 コンパイラー 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)
HTML	ハイパーテキスト マークアップ 言語 (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 リレー 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 プロバイダー (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	マスター 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ー (Pre-Compiled Headers)
PCRE	Perl Compatible Regular Expression
PID	プロセス識別子 (Process Identifier)
PTY	仮想端末 (pseudo terminal)
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)
SHA1	Secure-Hash Algorithm 1
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 アレー (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
- Bruce Dubbs <bdubbs@linuxfromscratch.org> - LFS 編集管理者
- Jim Gifford <jim@linuxfromscratch.org> - CLFS プロジェクト共同リーダー
- Pierre Labastie <pierre@linuxfromscratch.org> - BLFS 編集者、ALFS リーダー
- DJ Lucas <dj@linuxfromscratch.org> - LFS、BLFS 編集者
- Ken Moffat <ken@linuxfromscratch.org> - BLFS 編集者
- 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; 2008年まで
- Jean-Philippe Mengual <jmengual@linuxfromscratch.org> - フランスの LFS 翻訳プロジェクト; 2008年~2016年まで
- Julien Lepiller <jlepiller@linuxfromscratch.org> - フランスの LFS 翻訳プロジェクト; 2017年から現在まで
- 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 Baque <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

- Anton Maisak <info@linuxfromscratch.org.ru> - linuxfromscratch.org.ru 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
- 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
- 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
- Bryan Kadzban <bryan@linuxfromscratch.org> - LFS テクニカルライター
- 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
- Randy McMurphy <randy@linuxfromscratch.org> - BLFS プロジェクトリーダー、LFS 編集者
- 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
- Douglas R. Reno <renodr@linuxfromscratch.org> - Systemd 編集者
- Ryan Oliver <ryan@linuxfromscratch.org> - CLFS プロジェクト共同リーダー
- 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、最大で 5 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、対象パッケージのプログラムやライブラリが、実行時にその利用を必要とするパッケージです。3 つめは、1 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。4 つめ以降は、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が。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。

Acl

インストール依存パッケージ:	Attr, Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, Texinfo
実行時依存パッケージ:	Attr, Glibc
テストスイート依存パッケージ:	Automake, Diffutils, Findutils, Libtool
事前インストールパッケージ:	Coreutils, Sed, Tar, Vim
任意依存パッケージ:	なし

Attr

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed, Texinfo
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	Automake, Diffutils, Findutils, Libtool
事前インストールパッケージ:	Acl, Libcap
任意依存パッケージ:	なし

Autoconf

インストール依存パッケージ:	Bash, Coreutils, Grep, M4, Make, Perl, Sed, Texinfo
実行時依存パッケージ:	Bash, Coreutils, Grep, M4, Make, 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
実行時依存パッケージ:	Bash, Coreutils, Grep, M4, 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
実行時依存パッケージ:	Glibc, Ncurses, Readline
テストスイート依存パッケージ:	Expect, Shadow
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

Bc

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Readline
実行時依存パッケージ:	Glibc, Ncurses, Readline
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:	Linux
任意依存パッケージ:	なし

Binutils

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, File, Flex, Gawk, GCC, Glibc, Grep, Make, Perl, Sed, Texinfo, Zlib
実行時依存パッケージ:	Glibc, 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:	Elfutils

Bison

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	Diffutils, Findutils, Flex
事前インストールパッケージ:	Kbd, Tar
任意依存パッケージ:	Doxygen

Bzip2

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, Patch
実行時依存パッケージ:	Glibc
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
任意依存パッケージ:	なし

Check

インストール依存パッケージ:	Gawk, GCC, Grep, Make, Sed, Texinfo
実行時依存パッケージ:	Bash, Gawk
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, Libcap, Make, OpenSSL, Patch, Perl, Sed, Texinfo
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	Diffutils, E2fsprogs, Findutils, Shadow, Util-linux
事前インストールパッケージ:	Bash, Diffutils, Eudev, Findutils, Man-DB
任意依存パッケージ:	Expect, pm, IO::Tty

DejaGNU

インストール依存パッケージ:	Bash, Coreutils, Diffutils, Expect, GCC, Grep, Make, Sed, Texinfo
実行時依存パッケージ:	Expect, Bash
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

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
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	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

E2fsprogs

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Sed, Texinfo, Util-linux
実行時依存パッケージ:	Glibc, Util-linux
テストスイート依存パッケージ:	Procps-ng, Psmisc
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

Eudev

インストール依存パッケージ:	Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Gperf, Make, Sed, Util-linux
実行時依存パッケージ:	Glibc, Kmod, Xz, Util-linux, Zlib
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

Expat

インストール依存パッケージ:	Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed
実行時依存パッケージ:	Glibc
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:	Python, XML::Parser
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
実行時依存パッケージ:	Glibc, 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:	Tk

File

インストール依存パッケージ:	Bash, Binutils, Bzip2, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Xz, Zlib
実行時依存パッケージ:	Glibc, Bzip2, Xz, Zlib
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:	libseccomp

Findutils

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo
実行時依存パッケージ:	Bash, Glibc
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
実行時依存パッケージ:	Bash, Glibc, M4
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:	Binutils, IProute2, Kbd, Kmod, 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, GMP, Grep, Make, MPFR, Patch, Readline, Sed, Texinfo
実行時依存パッケージ:	Bash, Glibc, Mpfr
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:	libsigsegv

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, Zstd
実行時依存パッケージ:	Bash, Binutils, Glibc, Mpc, Python
テストスイート依存パッケージ:	DejaGNU, Expect, Shadow
事前インストールパッケージ:	なし
任意依存パッケージ:	GNAT, ISL

GDBM

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, Sed
実行時依存パッケージ:	Bash, Glibc, Readline
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, Ncurses, Sed, Texinfo
実行時依存パッケージ:	Acl, Bash, Gcc, Glibc
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, Bison
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, Bison, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API ヘッダー, Make, Perl, Python, Sed, Texinfo
実行時依存パッケージ:	なし
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
実行時依存パッケージ:	GCC, Glibc
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

Gperf

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Make
実行時依存パッケージ:	GCC, Glibc
テストスイート依存パッケージ:	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

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
実行時依存パッケージ:	Glibc
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, libsigsegv

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
実行時依存パッケージ:	GCC, Glibc, Perl
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:	ghostscript, UcharDET

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, Xz
実行時依存パッケージ:	Bash, GCC, Gettext, Glibc, Xz, 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

Gzip

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, Texinfo
実行時依存パッケージ:	Bash, Glibc
テストスイート依存パッケージ:	Diffutils, Less
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
実行時依存パッケージ:	なし
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
実行時依存パッケージ:	GCC, Glibc, Ncurses, Readline
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:	Tar
任意依存パッケージ:	なし

Intltool

インストール依存パッケージ:	Bash, Gawk, Glibc, Make, Perl, Sed, XML::Parser
実行時依存パッケージ:	Autoconf, Automake, Bash, Glibc, Grep, Perl, Sed
テストスイート依存パッケージ:	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

IProute2

インストール依存パッケージ:	Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, Libcap, Libelf, Linux API ヘッダー, Zlib
実行時依存パッケージ:	Bash, Coreutils, Glibc, Libcap, Libelf, 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:	Berkeley DB, iptables

Jinja2

インストール依存パッケージ:	MarkupSafe, Python
実行時依存パッケージ:	MarkupSafe, Python
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:	Systemd
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, Check, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, Sed
実行時依存パッケージ:	Bash, Coreutils, Glibc
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

Kmod

インストール依存パッケージ:	Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, OpenSSL, Pkg-config, Sed, Xz, Zlib
実行時依存パッケージ:	Glibc, Xz, 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:	Eudev
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
実行時依存パッケージ:	Glibc, 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:	Gzip
任意依存パッケージ:	PCRE

Libcap

インストール依存パッケージ:	Attr, Bash, Binutils, Coreutils, GCC, Glibc, Perl, Make, Sed
実行時依存パッケージ:	Glibc
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:	IProute2, Shadow
任意依存パッケージ:	Linux-PAM

Libelf

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Make
実行時依存パッケージ:	Glibc, Zlib
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:	IProute2, Linux
任意依存パッケージ:	なし

Libffi

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Make, Sed
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	DejaGnu
事前インストールパッケージ:	Python
任意依存パッケージ:	なし

Libpipeline

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Texinfo
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	Check
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

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
実行時依存パッケージ:	Autoconf, Automake, Bash, Binutils, Coreutils, File, GCC, Glibc, Grep, Make, Sed
テストスイート依存パッケージ:	Autoconf, Automake, 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

インストール依存パッケージ:	Bash, Bc, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Kmod, Libelf, Make, Ncurses, OpenSSL, Perl, Sed
実行時依存パッケージ:	なし
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:	cpio

Linux API Headers

インストール依存パッケージ:	Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Perl, Sed
実行時依存パッケージ:	なし
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
実行時依存パッケージ:	Bash, Glibc
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
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	Perl, Procps-ng
事前インストールパッケージ:	なし
任意依存パッケージ:	Guile

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, Libpipeline, Make, Sed, Xz
実行時依存パッケージ:	Bash, GDBM, Groff, Glibc, Gzip, Less, Libpipeline, Zlib
テストスイート依存パッケージ:	Util-linux
事前インストールパッケージ:	なし
任意依存パッケージ:	libseccomp

Man-Pages

インストール依存パッケージ:	Bash, Coreutils, Make
実行時依存パッケージ:	なし
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

MarkupSafe

インストール依存パッケージ:	Python
実行時依存パッケージ:	Python
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:	Jinja2
任意依存パッケージ:	なし

Meson

インストール依存パッケージ:	Ninja, Python
実行時依存パッケージ:	Python
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:	Systemd
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
実行時依存パッケージ:	Glibc, GMP, MPFR
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
実行時依存パッケージ:	Glibc, GMP
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:	Gawk, 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
実行時依存パッケージ:	Glibc
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-ng, Psmisc, Readline, Texinfo, Util-linux, Vim
任意依存パッケージ:	なし

Ninja

インストール依存パッケージ:	Binutils, Coreutils, GCC, Python
実行時依存パッケージ:	GCC, Glibc
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:	Meson
任意依存パッケージ:	Asciidoc, Doxygen, Emacs, re2c

OpenSSL

インストール依存パッケージ:	Binutils, Coreutils, GCC, Make, Perl
実行時依存パッケージ:	Glibc, Perl
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:	Coreutils, Kmod, Linux
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
実行時依存パッケージ:	Glibc, Patch
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:	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
実行時依存パッケージ:	GDBM, Glibc
テストスイート依存パッケージ:	Iana-Etc, Less, Procps-ng
事前インストールパッケージ:	Autoconf
任意依存パッケージ:	Berkeley DB

Pkg-config

インストール依存パッケージ:	Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Popt, Sed
実行時依存パッケージ:	Glibc
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:	Kmod
任意依存パッケージ:	Glib2

Procps-ng

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses
実行時依存パッケージ:	Glibc
テストスイート依存パッケージ:	DejaGNU
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
実行時依存パッケージ:	Glibc, 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

Python

インストール依存パッケージ:	Bash, Binutils, Coreutils, Expat, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Make, Ncurses, OpenSSL, Sed, Util-linux
実行時依存パッケージ:	Bzip2, Expat, Gdbm, Glibc, Libffi, Ncurses, OpenSSL, Zlib
テストスイート依存パッケージ:	GDB, Valgrind
事前インストールパッケージ:	Ninja
任意依存パッケージ:	Berkeley DB, libnsl, SQLite, and Tk

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
実行時依存パッケージ:	Glibc, 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:	Bash, Bc, Gawk
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
実行時依存パッケージ:	Acl, Attr, Glibc
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:	なし

Shadow

インストール依存パッケージ:	Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Libcap, Make, Sed
実行時依存パッケージ:	Glibc
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:	Coreutils
任意依存パッケージ:	CrackLib, Linux-PAM

Sysklogd

インストール依存パッケージ:	Binutils, Coreutils, GCC, Glibc, Make, Patch
実行時依存パッケージ:	Glibc
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

Systemd

インストール依存パッケージ:	Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Expat, Gawk, GCC, Glibc, Gperf, Grep, Jinja2, Libcap, Meson, Sed, Util-linux, Zstd
実行時依存パッケージ:	Acl, Attr, Glibc, Libcap, Util-linux
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:	btrfs-progs, cURL, cryptsetup, docbook-xml, docbook-xsl-nons, elfutils, Git, gnu-efi, GnuTLS, iptables, kexec-tools, libfido2, libgcrypt, libidn2, Libmicrohttpd, libpwquality, libseccomp, libxkbcommon, libxslt, Linux-PAM, lxml, LZ4, make-ca, pll-kit, PCRE2, Polkit, qemu, qrencode, quota-tools, rsync, Sphinx, tpm2-tss, Valgrind, zsh

Sysvinit

インストール依存パッケージ:	Binutils, Coreutils, GCC, Glibc, Make, and Sed
実行時依存パッケージ:	Glibc
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:	Acl, Attr, Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, Texinfo
実行時依存パッケージ:	Acl, Attr, Bzip2, Glibc, Gzip, Xz
テストスイート依存パッケージ:	Autoconf, 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
実行時依存パッケージ:	Glibc, Zlib
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
実行時依存パッケージ:	Glibc, Ncurses
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

Util-linux

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, Eudev, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Libcap, Make, Ncurses, Sed, Zlib
実行時依存パッケージ:	Glibc, Libcap, Ncurses, Readline, Zlib
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:	Linux-PAM, smartmontools

Vim

インストール依存パッケージ:	Acl, Attr, Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed
実行時依存パッケージ:	Acl, Attr, Glibc, Python, Ncurses, 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:	Xorg, GTK+2, LessTif, Ruby, GPM

wheel

インストール依存パッケージ:	Python
実行時依存パッケージ:	Python
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:	Jinja2
任意依存パッケージ:	なし

XML::Parser

インストール依存パッケージ:	Bash, Binutils, Coreutils, Expat, GCC, Glibc, Make, Perl
実行時依存パッケージ:	Expat, Glibc, Perl
テストスイート依存パッケージ:	Perl
事前インストールパッケージ:	Intltool
任意依存パッケージ:	なし

Xz

インストール依存パッケージ:	Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make
実行時依存パッケージ:	Glibc
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:	Eudev, File, GRUB, Kmod, Man-DB
任意依存パッケージ:	なし

Zlib

インストール依存パッケージ:	Bash, Binutils, Coreutils, GCC, Glibc, Make, Sed
実行時依存パッケージ:	Glibc
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, Kmod, Perl, Util-linux
任意依存パッケージ:	なし

Zstd

インストール依存パッケージ:	Binutils, Coreutils, GCC, Glibc, Gzip, Make, Xz
実行時依存パッケージ:	Glibc
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, Systemd
任意依存パッケージ:	LZ4

付録D ブートスクリプトと sysconfig スクリプト version-20220723

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/bash
#####
# Begin rc
#
# Description : Main Run Level Control Script
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#              : DJ Lucas - dj@linuxfromscratch.org
# Updates      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#              : Pierre Labastie - pierre@linuxfromscratch.org
#
# Version      : LFS 7.0
#
# Notes        : Updates March 24th, 2022: new semantics of S/K files
#              - Instead of testing that S scripts were K scripts in the
#                previous runlevel, test that they were not S scripts
#              - Instead of testing that K scripts were S scripts in the
#                previous runlevel, test that they were not K scripts
#              - S scripts in runlevel 0 or 6 are now run with
#                "script start" (was "script stop" previously).
#####

. /lib/lsb/init-functions

print_error_msg()
{
    log_failure_msg
    # $i is set when called
    MSG="FAILURE:\n\nYou should not be reading this error message.\n\n"
    MSG="${MSG}It means that an unforeseen error took place in\n"
    MSG="${MSG}${i},\n"
    MSG="${MSG}which exited with a return value of ${error_value}.\n"

    MSG="${MSG}If you're able to track this error down to a bug in one of\n"
    MSG="${MSG}the files provided by the ${DISTRO_MINI} book,\n"
    MSG="${MSG}please be so kind to inform us at ${DISTRO_CONTACT}.\n"
    log_failure_msg "${MSG}"

    log_info_msg "Press Enter to continue..."
    wait_for_user
}

check_script_status()
{
    # $i is set when called
    if [ ! -f ${i} ]; then
        log_warning_msg "${i} is not a valid symlink."
        SCRIPT_STAT="1"
    fi
}
```

```

fi

if [ ! -x ${i} ]; then
    log_warning_msg "${i} is not executable, skipping."
    SCRIPT_STAT="1"
fi
}

run()
{
    if [ -z $interactive ]; then
        ${1} ${2}
        return $?
    fi

    while true; do
        read -p "Run ${1} ${2} (Yes/no/continue)? " -n 1 runit
        echo

        case ${runit} in
            c | C)
                interactive=""
                ${i} ${2}
                ret=${?}
                break;
                ;;

            n | N)
                return 0
                ;;

            y | Y)
                ${i} ${2}
                ret=${?}
                break
                ;;

            esac
        done

        return $ret
    }

# Read any local settings/overrides
[ -r /etc/sysconfig/rc.site ] && source /etc/sysconfig/rc.site

DISTRO=${DISTRO:-"Linux From Scratch"}
DISTRO_CONTACT=${DISTRO_CONTACT:-"lfs-dev@lists.linuxfromscratch.org (Registration required)"}
DISTRO_MINI=${DISTRO_MINI:-"LFS"}
IPROMPT=${IPROMPT:-"no"}

# 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 /etc/rc.d/rc${runlevel}.d ]; then
    log_info_msg "/etc/rc.d/rc${runlevel}.d does not exist.\n"
    exit 1
fi

if [ "$runlevel" == "6" -o "$runlevel" == "0" ]; then IPROMPT="no"; fi

# Note: In ${LOGLEVEL:-7}, it is ':' 'dash' '7', not minus 7
if [ "$runlevel" == "S" ]; then
    [ -r /etc/sysconfig/console ] && source /etc/sysconfig/console
    dmesg -n "${LOGLEVEL:-7}"
fi

if [ "${IPROMPT}" == "yes" -a "${runlevel}" == "S" ]; then
    # The total length of the distro welcome string, without escape codes
    wlen=${wlen:-$(echo "Welcome to ${DISTRO}" | wc -c )}
    welcome_message=${welcome_message:-"Welcome to ${INFO}${DISTRO}${NORMAL}"}

    # The total length of the interactive string, without escape codes
    ilen=${ilen:-$(echo "Press 'I' to enter interactive startup" | wc -c )}
    i_message=${i_message:-"Press '${FAILURE}I${NORMAL}' to enter interactive startup"}

    # dcol and icol are spaces before the message to center the message
    # on screen. itime is the amount of wait time for the user to press a key
    wcol=$(( ( ${COLUMNS} - ${wlen} ) / 2 ))
    icol=$(( ( ${COLUMNS} - ${ilen} ) / 2 ))
    itime=${itime:-"3"}

    echo -e "\n\n"
    echo -e "\\033[${wcol}G${welcome_message}"
    echo -e "\\033[${icol}G${i_message}${NORMAL}"
    echo ""
    read -t "${itime}" -n 1 interactive 2>&1 > /dev/null
fi

# Make lower case
[ "${interactive}" == "I" ] && interactive="i"
[ "${interactive}" != "i" ] && interactive=""

# Read the state file if it exists from runlevel S
[ -r /run/interactive ] && source /run/interactive

# Stop all services marked as K, except if marked as K in the previous
# runlevel: it is the responsibility of the script to not try to kill
# a non running service
if [ "${previous}" != "N" ]; then
    for i in $(ls -v /etc/rc.d/rc${runlevel}.d/K* 2> /dev/null)
    do
        check_script_status
        if [ "${SCRIPT_STAT}" == "1" ]; then
            SCRIPT_STAT="0"
            continue
        fi

        suffix=${i#/etc/rc.d/rc${runlevel}.d/K[0-9][0-9]}
        [ -e /etc/rc.d/rc${previous}.d/K[0-9][0-9]$suffix ] && continue

        run ${i} stop
    done
fi

```



```

    error_value=${?}

    if [ "${error_value}" != "0" ]; then print_error_msg; fi
done
fi

if [ "${previous}" == "N" ]; then export IN_BOOT=1; fi

if [ "$runlevel" == "6" -a -n "${FASTBOOT}" ]; then
    touch /fastboot
fi

# Start all services marked as S in this runlevel, except if marked as
# S in the previous runlevel
# it is the responsibility of the script to not try to start an already running
# service
for i in $( ls -v /etc/rc.d/rc${runlevel}.d/S* 2> /dev/null )
do

    if [ "${previous}" != "N" ]; then
        suffix=${i#/etc/rc.d/rc${runlevel}.d/S[0-9][0-9]}
        [ -e /etc/rc.d/rc${previous}.d/S[0-9][0-9]$suffix ] && continue
    fi

    check_script_status
    if [ "${SCRIPT_STAT}" == "1" ]; then
        SCRIPT_STAT="0"
        continue
    fi

    run ${i} start

    error_value=${?}

    if [ "${error_value}" != "0" ]; then print_error_msg; fi
done

# Store interactive variable on switch from runlevel S and remove if not
if [ "$runlevel" == "S" -a "${interactive}" == "i" ]; then
    echo "interactive=\"i\"" > /run/interactive
else
    rm -f /run/interactive 2> /dev/null
fi

# Copy the boot log on initial boot only
if [ "${previous}" == "N" -a "${runlevel}" != "S" ]; then
    cat $BOOTLOG >> /var/log/boot.log

    # Mark the end of boot
    echo "-----" >> /var/log/boot.log

    # Remove the temporary file
    rm -f $BOOTLOG 2> /dev/null
fi

# End rc

```

D.2. /lib/lsb/init-functions

```
#!/bin/sh
#####
#
# Begin /lib/lsb/init-funtions
#
# Description : Run Level Control Functions
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#              : DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
# Notes       : With code based on Matthias Benkmann's simpleinit-msb
#              http://winterdrache.de/linux/newboot/index.html
#
#              The file should be located in /lib/lsb
#
#####

## Environmental setup
# Setup default values for environment
umask 022
export PATH="/bin:/usr/bin:/sbin:/usr/sbin"

## 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

# Use a colored prefix
BMPREFIX=" "
SUCCESS_PREFIX="\${SUCCESS} * \${NORMAL} "
FAILURE_PREFIX="\${FAILURE}*****\${NORMAL} "
WARNING_PREFIX="\${WARNING} *** \${NORMAL} "
SKIP_PREFIX="\${INFO} S \${NORMAL} "

SUCCESS_SUFFIX="\${BRACKET}[\${SUCCESS} OK \${BRACKET}]\${NORMAL} "
FAILURE_SUFFIX="\${BRACKET}[\${FAILURE} FAIL \${BRACKET}]\${NORMAL} "
WARNING_SUFFIX="\${BRACKET}[\${WARNING} WARN \${BRACKET}]\${NORMAL} "
SKIP_SUFFIX="\${BRACKET}[\${INFO} SKIP \${BRACKET}]\${NORMAL} "

BOOTLOG=/run/bootlog
KILLDELAY=3
SCRIPT_STAT="0"

# Set any user specified environment variables e.g. HEADLESS
```

```
[ -r /etc/sysconfig/rc.site ] && . /etc/sysconfig/rc.site

## 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 ))

## 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
CURS_ZERO="\033[0G"

#####
# start_daemon()
# Usage: start_daemon [-f] [-n nicelevel] [-p pidfile] pathname [args...]
#
# Purpose: This runs the specified program as a daemon
#
# Inputs: -f: (force) run the program even if it is already running.
#         -n nicelevel: specify a nice level. See 'man nice(1)'.
#         -p pidfile: use the specified file to determine PIDs.
#         pathname: the complete path to the specified program
#         args: additional arguments passed to the program (pathname)
#
# Return values (as defined by LSB exit codes):
#     0 - program is running or service is OK
#     1 - generic or unspecified error
#     2 - invalid or excessive argument(s)
#     5 - program is not installed
#####
start_daemon()
{
    local force=""
    local nice="0"
    local pidfile=""
    local pidlist=""
    local retval=""

    # Process arguments
    while true
    do
        case "${1}" in
            -f)
                force="1"
                shift 1
                ;;
            -n)

```

```

        nice="${2}"
        shift 2
        ;;

    -p)
        pidfile="${2}"
        shift 2
        ;;

    -*)
        return 2
        ;;

    *)
        program="${1}"
        break
        ;;
esac
done

# Check for a valid program
if [ ! -e "${program}" ]; then return 5; fi

# Execute
if [ -z "${force}" ]; then
    if [ -z "${pidfile}" ]; then
        # Determine the pid by discovery
        pidlist=`pidofproc "${1}"`
        retval="${?}"
    else
        # The PID file contains the needed PIDs
        # Note that by LSB requirement, the path must be given to pidofproc,
        # however, it is not used by the current implementation or standard.
        pidlist=`pidofproc -p "${pidfile}" "${1}"`
        retval="${?}"
    fi

    # Return a value ONLY
    # It is the init script's (or distribution's functions) responsibility
    # to log messages!
    case "${retval}" in

        0)
            # Program is already running correctly, this is a
            # successful start.
            return 0
            ;;

        1)
            # Program is not running, but an invalid pid file exists
            # remove the pid file and continue
            rm -f "${pidfile}"
            ;;

        3)
            # Program is not running and no pidfile exists
            # do nothing here, let start_deamon continue.
            ;;

        *)
            # Others as returned by status values shall not be interpreted

```

```

        # and returned as an unspecified error.
        return 1
        ;;
    esac
fi

# Do the start!
nice -n "${nice}" "${@"}
}

#####
# killproc()
# Usage: killproc [-p pidfile] pathname [signal]
#
# Purpose: Send control signals to running processes
#
# Inputs: -p pidfile, uses the specified pidfile
#         pathname, pathname to the specified program
#         signal, send this signal to pathname
#
# Return values (as defined by LSB exit codes):
#     0 - program (pathname) has stopped/is already stopped or a
#         running program has been sent specified signal and stopped
#         successfully
#     1 - generic or unspecified error
#     2 - invalid or excessive argument(s)
#     5 - program is not installed
#     7 - program is not running and a signal was supplied
#####
killproc()
{
    local pidfile
    local program
    local prefix
    local progname
    local signal="-TERM"
    local fallback="-KILL"
    local nosig
    local pidlist
    local retval
    local pid
    local delay="30"
    local piddead
    local dtime

    # Process arguments
    while true; do
        case "${1}" in
            -p)
                pidfile="${2}"
                shift 2
                ;;

            *)
                program="${1}"
                if [ -n "${2}" ]; then
                    signal="${2}"
                    fallback=""
                else
                    nosig=1
                fi
            fi
        esac
    done
}

```

```

        # Error on additional arguments
        if [ -n "${3}" ]; then
            return 2
        else
            break
        fi
    ;;
esac
done

# Check for a valid program
if [ ! -e "${program}" ]; then return 5; fi

# Check for a valid signal
check_signal "${signal}"
if [ "${?}" -ne "0" ]; then return 2; fi

# Get a list of pids
if [ -z "${pidfile}" ]; then
    # determine the pid by discovery
    pidlist=`pidofproc "${1}"`
    retval="${?}"
else
    # The PID file contains the needed PIDs
    # Note that by LSB requirement, the path must be given to pidofproc,
    # however, it is not used by the current implementation or standard.
    pidlist=`pidofproc -p "${pidfile}" "${1}"`
    retval="${?}"
fi

# Return a value ONLY
# It is the init script's (or distribution's functions) responsibility
# to log messages!
case "${retval}" in

    0)
        # Program is running correctly
        # Do nothing here, let killproc continue.
        ;;

    1)
        # Program is not running, but an invalid pid file exists
        # Remove the pid file.

        procname=${program##*/}

        if [[ -e "/run/${procname}.pid" ]]; then
            pidfile="/run/${procname}.pid"
            rm -f "${pidfile}"
        fi

        # This is only a success if no signal was passed.
        if [ -n "${nosig}" ]; then
            return 0
        else
            return 7
        fi
        ;;

    3)

```

```

# Program is not running and no pidfile exists
# This is only a success if no signal was passed.
if [ -n "${nosig}" ]; then
    return 0
else
    return 7
fi
;;

*)
# Others as returned by status values shall not be interpreted
# and returned as an unspecified error.
return 1
;;
esac

# Perform different actions for exit signals and control signals
check_sig_type "${signal}"

if [ "${?}" -eq "0" ]; then # Signal is used to terminate the program

# Account for empty pidlist (pid file still exists and no
# signal was given)
if [ "${pidlist}" != "" ]; then

# Kill the list of pids
for pid in ${pidlist}; do

kill -0 "${pid}" 2> /dev/null

if [ "${?}" -ne "0" ]; then
# Process is dead, continue to next and assume all is well
continue
else
kill "${signal}" "${pid}" 2> /dev/null

# Wait up to ${delay}/10 seconds to for "${pid}" to
# terminate in 10ths of a second

while [ "${delay}" -ne "0" ]; do
kill -0 "${pid}" 2> /dev/null || piddead="1"
if [ "${piddead}" = "1" ]; then break; fi
sleep 0.1
delay=$(( ${delay} - 1 ))
done

# If a fallback is set, and program is still running, then
# use the fallback
if [ -n "${fallback}" -a "${piddead}" != "1" ]; then
kill "${fallback}" "${pid}" 2> /dev/null
sleep 1
# Check again, and fail if still running
kill -0 "${pid}" 2> /dev/null && return 1
fi
fi
done
fi

# Check for and remove stale PID files.
if [ -z "${pidfile}" ]; then
# Find the basename of $program

```

```

prefix=`echo "${program}" | sed 's/[^/]*$//`
progrname=`echo "${program}" | sed "s@${prefix}@@"`

if [ -e "/run/${progrname}.pid" ]; then
    rm -f "/run/${progrname}.pid" 2> /dev/null
fi
else
    if [ -e "${pidfile}" ]; then rm -f "${pidfile}" 2> /dev/null; fi
fi

# For signals that do not expect a program to exit, simply
# let kill do its job, and evaluate kill's return for value

else # check_sig_type - signal is not used to terminate program
    for pid in ${pidlist}; do
        kill "${signal}" "${pid}"
        if [ "${?}" -ne "0" ]; then return 1; fi
    done
fi
}

#####
# pidofproc()
# Usage: pidofproc [-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
#
# Return values (as defined by LSB status codes):
#     0 - Success (PIDs to stdout)
#     1 - Program is dead, PID file still exists (remaining PIDs output)
#     3 - Program is not running (no output)
#####
pidofproc()
{
    local pidfile
    local program
    local prefix
    local progrname
    local pidlist
    local lpids
    local exitstatus="0"

# Process arguments
while true; do
    case "${1}" in

        -p)
            pidfile="${2}"
            shift 2
            ;;

        *)
            program="${1}"
            if [ -n "${2}" ]; then
                # Too many arguments
                # Since this is status, return unknown
                return 4
            else

```



```

                break
            fi
        ;;
    esac
done

# If a PID file is not specified, try and find one.
if [ -z "${pidfile}" ]; then
    # Get the program's basename
    prefix=`echo "${program}" | sed 's/[^/]*$//`

    if [ -z "${prefix}" ]; then
        procname="${program}"
    else
        procname=`echo "${program}" | sed "s@${prefix}@@"`
    fi

    # If a PID file exists with that name, assume that is it.
    if [ -e "/run/${procname}.pid" ]; then
        pidfile="/run/${procname}.pid"
    fi
fi

# If a PID file is set and exists, use it.
if [ -n "${pidfile}" -a -e "${pidfile}" ]; then

    # Use the value in the first line of the pidfile
    pidlist=`/bin/head -nl "${pidfile}"`
    # This can optionally be written as 'sed 1q' to replace 'head -nl'
    # should LFS move /bin/head to /usr/bin/head
else
    # Use pidof
    pidlist=`pidof "${program}"`
fi

# Figure out if all listed PIDs are running.
for pid in ${pidlist}; do
    kill -0 ${pid} 2> /dev/null

    if [ "${?}" -eq "0" ]; then
        lpids="${lpids}${pid} "
    else
        exitstatus="1"
    fi
done

if [ -z "${lpids}" -a ! -f "${pidfile}" ]; then
    return 3
else
    echo "${lpids}"
    return "${exitstatus}"
fi
}

#####
# statusproc() #
# Usage: statusproc [-p pidfile] pathname #
# # #
# Purpose: This function prints the status of a particular daemon to stdout #
# # #
# Inputs: -p pidfile, use the specified pidfile instead of pidof #

```

```

#      pathname, path to the specified program      #
#      #      #
# Return values:      #
#      0 - Status printed      #
#      1 - Input error. The daemon to check was not specified.      #
#####
statusproc()
{
    local pidfile
    local pidlist

    if [ "${#}" = "0" ]; then
        echo "Usage: statusproc [-p pidfile] {program}"
        exit 1
    fi

    # Process arguments
    while true; do
        case "${1}" in

            -p)
                pidfile="${2}"
                shift 2
                ;;

            *)
                if [ -n "${2}" ]; then
                    echo "Too many arguments"
                    return 1
                else
                    break
                fi
                ;;
        esac
    done

    if [ -n "${pidfile}" ]; then
        pidlist=`pidofproc -p "${pidfile}" @$`
    else
        pidlist=`pidofproc @$`
    fi

    # Trim trailing blanks
    pidlist=`echo "${pidlist}" | sed -r 's/ +$//'`

    base="${1##*/}"

    if [ -n "${pidlist}" ]; then
        /bin/echo -e "${INFO}${base} is running with Process" \
            "ID(s) ${pidlist}.${NORMAL}"
    else
        if [ -n "${base}" -a -e "/run/${base}.pid" ]; then
            /bin/echo -e "${WARNING}${1} is not running but" \
                "/run/${base}.pid exists.${NORMAL}"
        else
            if [ -n "${pidfile}" -a -e "${pidfile}" ]; then
                /bin/echo -e "${WARNING}${1} is not running" \
                    "but ${pidfile} exists.${NORMAL}"
            else
                /bin/echo -e "${INFO}${1} is not running.${NORMAL}"
            fi
        fi
    fi
}

```

```

        fi
    fi
}

#####
# timespec()
#
# Purpose: An internal utility function to format a timestamp
#         a boot log file. Sets the STAMP variable.
#
# Return value: Not used
#####
timespec()
{
    STAMP="$(echo `date +%b %d %T %:z` `hostname`)"
    return 0
}

#####
# log_success_msg()
# Usage: log_success_msg ["message"]
#
# Purpose: Print a successful status message to the screen and
#         a boot log file.
#
# Inputs: $@ - Message
#
# Return values: Not used
#####
log_success_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\033[^a-zA-Z]*.//g'`

    timespec
    /bin/echo -e "${STAMP} ${logmessage} OK" >> ${BOOTLOG}

    return 0
}

log_success_msg2()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"

    echo " OK" >> ${BOOTLOG}

    return 0
}

#####
# log_failure_msg()
# Usage: log_failure_msg ["message"]
#
# Purpose: Print a failure status message to the screen and
#         a boot log file.
#
# Inputs: $@ - Message

```

```

#                                                                                                     #
# Return values: Not used                                                                                   #
#####
log_failure_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}"

    # Strip non-printable characters from log file

    timespec
    logmessage=`echo "${@}" | sed 's/\\033[^a-zA-Z]*.//g'`
    /bin/echo -e "${STAMP} ${logmessage} FAIL" >> ${BOOTLOG}

    return 0
}

log_failure_msg2()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}"

    echo "FAIL" >> ${BOOTLOG}

    return 0
}

#####
# log_warning_msg()                                                                                       #
# Usage: log_warning_msg ["message"]                                                                     #
#                                                                                                           #
# Purpose: Print a warning status message to the screen and                                           #
#           a boot log file.                                                                               #
#                                                                                                           #
# Return values: Not used                                                                                   #
#####
log_warning_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${WARNING_PREFIX}${SET_COL}${WARNING_SUFFIX}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\033[^a-zA-Z]*.//g'`
    timespec
    /bin/echo -e "${STAMP} ${logmessage} WARN" >> ${BOOTLOG}

    return 0
}

log_skip_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${SKIP_PREFIX}${SET_COL}${SKIP_SUFFIX}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\033[^a-zA-Z]*.//g'`
    /bin/echo "SKIP" >> ${BOOTLOG}

    return 0
}

```

```
#####
# log_info_msg()
# Usage: log_info_msg message
#
# Purpose: Print an information message to the screen and
#         a boot log file. Does not print a trailing newline character.
#
# Return values: Not used
#####
log_info_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\033[^a-zA-Z]*.//g'`
    timespec
    /bin/echo -n -e "${STAMP} ${logmessage}" >> ${BOOTLOG}

    return 0
}

log_info_msg2()
{
    /bin/echo -n -e "${@}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\033[^a-zA-Z]*.//g'`
    /bin/echo -n -e "${logmessage}" >> ${BOOTLOG}

    return 0
}

#####
# evaluate_retval()
# Usage: Evaluate a return value and print success or failure as appropriate
#
# Purpose: Convenience function to terminate an info message
#
# Return values: Not used
#####
evaluate_retval()
{
    local error_value="${?}"

    if [ ${error_value} = 0 ]; then
        log_success_msg2
    else
        log_failure_msg2
    fi
}

#####
# check_signal()
# Usage: check_signal [ -{signal} ]
#
# Purpose: Check for a valid signal. This is not defined by any LSB draft,
#         however, it is required to check the signals to determine if the
#         signals chosen are invalid arguments to the other functions.
#
# Inputs: Accepts a single string value in the form of -{signal}
#
#####
```

```

# Return values:
#     0 - Success (signal is valid)
#     1 - Signal is not valid
#####
check_signal()
{
    local valsig

    # Add error handling for invalid signals
    valsig=" -ALRM -HUP -INT -KILL -PIPE -POLL -PROF -TERM -USR1 -USR2"
    valsig="${valsig} -VTALRM -STKFLT -PWR -WINCH -CHLD -URG -TSTP -TTIN"
    valsig="${valsig} -TTOU -STOP -CONT -ABRT -FPE -ILL -QUIT -SEGV -TRAP"
    valsig="${valsig} -SYS -EMT -BUS -XCPU -XFSZ -0 -1 -2 -3 -4 -5 -6 -8 -9"
    valsig="${valsig} -11 -13 -14 -15 "

    echo "${valsig}" | grep -- " ${1} " > /dev/null

    if [ "${?}" -eq "0" ]; then
        return 0
    else
        return 1
    fi
}

#####
# check_sig_type()
# Usage: check_signal [ -{signal} | {signal} ]
#
# Purpose: Check if signal is a program termination signal or a control signal
#           This is not defined by any LSB draft, however, it is required to
#           check the signals to determine if they are intended to end a
#           program or simply to control it.
#
# Inputs: Accepts a single string value in the form or -{signal} or {signal}
#
# Return values:
#     0 - Signal is used for program termination
#     1 - Signal is used for program control
#####
check_sig_type()
{
    local valsig

    # The list of termination signals (limited to generally used items)
    valsig=" -ALRM -INT -KILL -TERM -PWR -STOP -ABRT -QUIT -2 -3 -6 -9 -14 -15 "

    echo "${valsig}" | grep -- " ${1} " > /dev/null

    if [ "${?}" -eq "0" ]; then
        return 0
    else
        return 1
    fi
}

#####
# wait_for_user()
#
# Purpose: Wait for the user to respond if not a headless system
#
#####

```

```

wait_for_user()
{
    # Wait for the user by default
    [ "${HEADLESS=0}" = "0" ] && read ENTER
    return 0
}

#####
# is_true()
#
# Purpose: Utility to test if a variable is true | yes | 1
#
#####
is_true()
{
    [ "$1" = "1" ] || [ "$1" = "yes" ] || [ "$1" = "true" ] || [ "$1" = "y" ] ||
    [ "$1" = "t" ]
}

# End /lib/lsb/init-functions

```

D.3. /etc/rc.d/init.d/mountvirtfs

```

#!/bin/sh
#####
# Begin mountvirtfs
#
# Description : Ensure proc, sysfs, run, and dev are mounted
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          mountvirtfs
# Required-Start:    $first
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:     S
# Default-Stop:
# Short-Description: Mounts various special fs needed at start
# Description:       Mounts /sys and /proc virtual (kernel) filesystems.
#                   Mounts /run (tmpfs) and /dev (devtmpfs).
#                   This is done only if they are not already mounted.
#                   with the kernel config proposed in the book, dev
#                   should be automatically mounted by the kernel.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        # Make sure /run is available before logging any messages
        if ! mountpoint /run >/dev/null; then

```

```

    mount /run || failed=1
fi

mkdir -p /run/lock /run/shm
chmod 1777 /run/shm /run/lock

log_info_msg "Mounting virtual file systems: ${INFO}/run"

if ! mountpoint /proc >/dev/null; then
    log_info_msg2 " ${INFO}/proc"
    mount -o nosuid,noexec,nodev /proc || failed=1
fi

if ! mountpoint /sys >/dev/null; then
    log_info_msg2 " ${INFO}/sys"
    mount -o nosuid,noexec,nodev /sys || failed=1
fi

if ! mountpoint /dev >/dev/null; then
    log_info_msg2 " ${INFO}/dev"
    mount -o mode=0755,nosuid /dev || failed=1
fi

ln -sf /run/shm /dev/shm

(exit ${failed})
evaluate_retval
exit $failed
;;

*)
echo "Usage: ${0} {start}"
exit 1
;;

esac

# End mountvirtfs

```

D.4. /etc/rc.d/init.d/modules

```

#!/bin/sh
#####
# Begin modules
#
# Description : Module auto-loading script
#
# Authors      : Zack Winkles
#                DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          modules
# Required-Start:    mountvirtfs
# Should-Start:
# Required-Stop:
# Should-Stop:

```



```

# Default-Start:      S
# Default-Stop:
# Short-Description:  Loads required modules.
# Description:        Loads modules listed in /etc/sysconfig/modules.
# X-LFS-Provided-By:  LFS
### END INIT INFO

# Assure that the kernel has module support.
[ -e /proc/modules ] || exit 0

. /lib/lsb/init-functions

case "${1}" in
    start)
        # Exit if there's no modules file or there are no
        # valid entries
        [ -r /etc/sysconfig/modules ] || exit 0
        grep -E -qv '^(#|)' /etc/sysconfig/modules || exit 0

        log_info_msg "Loading modules:"

        # 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, passing any arguments provided.
            modprobe ${module} ${args} >/dev/null

            # Print the module name if successful, otherwise take note.
            if [ $? -eq 0 ]; then
                log_info_msg2 " ${module}"
            else
                failedmod="${failedmod} ${module}"
            fi
        done < /etc/sysconfig/modules

        # Print a message about successfully loaded modules on the correct line.
        log_success_msg2

        # Print a failure message with a list of any modules that
        # may have failed to load.
        if [ -n "${failedmod}" ]; then
            log_failure_msg "Failed to load modules:${failedmod}"
            exit 1
        fi
        ;;
    *)
        echo "Usage: ${0} {start}"
        exit 1
        ;;
esac

exit 0

```

```
# End modules
```

D.5. /etc/rc.d/init.d/udev

```
#!/bin/sh
#####
# Begin udev
#
# Description : Udev cold-plugging script
#
# Authors      : Zack Winkles, Alexander E. Patrakov
#                DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          udev $time
# Required-Start:    localnet
# Should-Start:      modules
# Required-Stop:
# Should-Stop:
# Default-Start:     S
# Default-Stop:
# Short-Description: Populates /dev with device nodes.
# Description:       Mounts a tempfs on /dev and starts the udevd daemon.
#                    Device nodes are created as defined by udev.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Populating /dev with device nodes... "
        if ! grep -q '[[[:space:]]sysfs' /proc/mounts; then
            log_failure_msg2
            msg="FAILURE:\n\nUnable to create "
            msg="${msg}devices without a SysFS filesystem\n\n"
            msg="${msg}After you press Enter, this system "
            msg="${msg}will be halted and powered off.\n\n"
            log_info_msg "$msg"
            log_info_msg "Press Enter to continue..."
            wait_for_user
            /etc/rc.d/init.d/halt stop
        fi

        # Start the udev daemon to continually watch for, and act on,
        # uevents
        /sbin/udev --daemon

        # Now traverse /sys in order to "coldplug" devices that have
        # already been discovered
        /sbin/udevadm trigger --action=add      --type=subsystems
        /sbin/udevadm trigger --action=add      --type=devices
        /sbin/udevadm trigger --action=change  --type=devices

        # Now wait for udevd to process the uevents we triggered
```

```

if ! is_true "$OMIT_UDEV_SETTLE"; then
    /sbin/udevadm settle
fi

# If any LVM based partitions are on the system, ensure they
# are activated so they can be used.
if [ -x /sbin/vgchange ]; then /sbin/vgchange -a y >/dev/null; fi

log_success_msg2
;;

*)
echo "Usage ${0} {start}"
exit 1
;;
esac

exit 0

# End udev

```

D.6. /etc/rc.d/init.d/swap

```

#!/bin/sh
#####
# Begin swap
#
# Description : Swap Control Script
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          swap
# Required-Start:    udev
# Should-Start:      modules
# Required-Stop:     localnet
# Should-Stop:       $local_fs
# Default-Start:     S
# Default-Stop:      0 6
# Short-Description: Mounts and unmounts swap partitions.
# Description:       Mounts and unmounts swap partitions defined in
#                   /etc/fstab.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Activating all swap files/partitions..."
        swapon -a
        evaluate_retval
        ;;

```

```

stop)
    log_info_msg "Deactivating all swap files/partitions..."
    swapoff -a
    evaluate_retval
    ;;

restart)
    ${0} stop
    sleep 1
    ${0} start
    ;;

status)
    log_success_msg "Retrieving swap status."
    swapon -s
    ;;

*)
    echo "Usage: ${0} {start|stop|restart|status}"
    exit 1
    ;;
esac

exit 0

# End swap

```

D.7. /etc/rc.d/init.d/setclock

```

#!/bin/sh
#####
# Begin setclock
#
# Description : Setting Linux Clock
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:
# Required-Start:
# Should-Start:      modules
# Required-Stop:
# Should-Stop:       $syslog
# Default-Start:     S
# Default-Stop:
# Short-Description: Stores and restores time from the hardware clock
# Description:       On boot, system time is obtained from hwclock. The
#                   hardware clock can also be set on shutdown.
#
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

[ -r /etc/sysconfig/clock ] && . /etc/sysconfig/clock

```

```

case "${UTC}" in
    yes|true|1)
        CLOCKPARAMS="${CLOCKPARAMS} --utc"
        ;;

    no|false|0)
        CLOCKPARAMS="${CLOCKPARAMS} --localtime"
        ;;

esac

case ${1} in
    start)
        hwclock --hctosys ${CLOCKPARAMS} >/dev/null
        ;;

    stop)
        log_info_msg "Setting hardware clock..."
        hwclock --systohc ${CLOCKPARAMS} >/dev/null
        evaluate_retval
        ;;

    *)
        echo "Usage: ${0} {start|stop}"
        exit 1
        ;;

esac

exit 0

```

D.8. /etc/rc.d/init.d/checkfs

```

#!/bin/sh
#####
# Begin checkfs
#
# Description : File System Check
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               A. Luebke - luebke@users.sourceforge.net
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
# 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
#

```

```
#####
### BEGIN INIT INFO
# Provides:          checkfs
# Required-Start:    udev swap
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:     S
# Default-Stop:
# Short-Description: Checks local filesystems before mounting.
# Description:       Checks local filesystems before mounting.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        if [ -f /fastboot ]; then
            msg="/fastboot found, will omit "
            msg="${msg} file system checks as requested.\n"
            log_info_msg "${msg}"
            exit 0
        fi

        log_info_msg "Mounting root file system in read-only mode... "
        mount -n -o remount,ro / >/dev/null

        if [ ${?} != 0 ]; then
            log_failure_msg2
            msg="\n\nCannot check root "
            msg="${msg}filesystem because it could not be mounted "
            msg="${msg}in read-only mode.\n\n"
            msg="${msg}After you press Enter, this system will be "
            msg="${msg}halted and powered off.\n\n"
            log_failure_msg "${msg}"

            log_info_msg "Press Enter to continue..."
            wait_for_user
            /etc/rc.d/init.d/halt stop
        else
            log_success_msg2
        fi

        if [ -f /forcefsck ]; then
            msg="/forcefsck found, forcing file"
            msg="${msg} system checks as requested."
            log_success_msg "${msg}"
            options="-f"
        else
            options=""
        fi

        log_info_msg "Checking file systems..."
        # Note: -a option used to be -p; but this fails e.g. on fsck.minix
        if is_true "$VERBOSE_FSCK"; then
            fsck ${options} -a -A -C -T
        else
            fsck ${options} -a -A -C -T >/dev/null
        fi
    esac
done
```

```

error_value=${?}

if [ "${error_value}" = 0 ]; then
    log_success_msg2
fi

if [ "${error_value}" = 1 ]; then
    msg="\nWARNING:\n\nFile system errors "
    msg="${msg}were found and have been corrected.\n"
    msg="${msg}          You may want to double-check that "
    msg="${msg}everything was fixed properly."
    log_warning_msg "$msg"
fi

if [ "${error_value}" = 2 -o "${error_value}" = 3 ]; then
    msg="\nWARNING:\n\nFile system errors "
    msg="${msg}were found and have been been "
    msg="${msg}corrected, but the nature of the "
    msg="${msg}errors require this system to be rebooted.\n\n"
    msg="${msg}After you press enter, "
    msg="${msg}this system will be rebooted\n\n"
    log_failure_msg "$msg"

    log_info_msg "Press Enter to continue..."
    wait_for_user
    reboot -f
fi

if [ "${error_value}" -gt 3 -a "${error_value}" -lt 16 ]; then
    msg="\nFAILURE:\n\nFile system errors "
    msg="${msg}were encountered that could not be "
    msg="${msg}fixed automatically.\nThis system "
    msg="${msg}cannot continue to boot and will "
    msg="${msg}therefore be halted until those "
    msg="${msg}errors are fixed manually by a "
    msg="${msg}System Administrator.\n\n"
    msg="${msg}After you press Enter, this system will be "
    msg="${msg}halted and powered off.\n\n"
    log_failure_msg "$msg"

    log_info_msg "Press Enter to continue..."
    wait_for_user
    /etc/rc.d/init.d/halt stop
fi

if [ "${error_value}" -ge 16 ]; then
    msg="FAILURE:\n\nUnexpected failure "
    msg="${msg}running fsck. Exited with error "
    msg="${msg} code: ${error_value}.\n"
    log_info_msg $msg
    exit ${error_value}
fi

exit 0
;;
*)
echo "Usage: ${0} {start}"
exit 1
;;
esac

```

```
# End checkfs
```

D.9. /etc/rc.d/init.d/mountfs

```
#!/bin/sh
#####
# Begin mountfs
#
# Description : File System Mount Script
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          $local_fs
# Required-Start:    udev checkfs
# Should-Start:      modules
# Required-Stop:     localnet
# Should-Stop:
# Default-Start:     S
# Default-Stop:      0 6
# Short-Description: Mounts/unmounts local filesystems defined in /etc/fstab.
# Description:       Remounts root filesystem read/write and mounts all
#                   remaining local filesystems defined in /etc/fstab on
#                   start. Remounts root filesystem read-only and unmounts
#                   remaining filesystems on stop.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Remounting root file system in read-write mode..."
        mount --options remount,rw / >/dev/null
        evaluate_retval

        # Remove fsck-related file system watermarks.
        rm -f /fastboot /forcefsck

        # Make sure /dev/pts exists
        mkdir -p /dev/pts

        # This will mount all filesystems that do not have _netdev in
        # their option list. _netdev denotes a network filesystem.

        log_info_msg "Mounting remaining file systems..."
        failed=0
        mount --all --test-opts no_netdev >/dev/null || failed=1
        evaluate_retval
        exit $failed
        ;;

    stop)

```



```

# Don't unmount virtual file systems like /run
log_info_msg "Unmounting all other currently mounted file systems..."
# Ensure any loop devies are removed
losetup -D
umount --all --detach-loop --read-only \
    --types notmpfs,nosysfs,nodevtmpfs,noproc,nodevpts >/dev/null
evaluate_retval

# Make sure / is mounted read only (umount bug)
mount --options remount,ro /

# Make all LVM volume groups unavailable, if appropriate
# This fails if swap or / are on an LVM partition
#if [ -x /sbin/vgchange ]; then /sbin/vgchange -an > /dev/null; fi
if [ -r /etc/mdadm.conf ]; then
    log_info_msg "Mark arrays as clean..."
    mdadm --wait-clean --scan
    evaluate_retval
fi
;;

*)
echo "Usage: ${0} {start|stop}"
exit 1
;;

esac

# End mountfs

```

D.10. /etc/rc.d/init.d/udev_retry

```

#!/bin/sh
#####
# Begin udev_retry
#
# Description : Udev cold-plugging script (retry)
#
# Authors      : Alexander E. Patrakov
#                DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#                Bryan Kadzban -
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          udev_retry
# Required-Start:    udev
# Should-Start:      $local_fs cleanfs
# Required-Stop:
# Should-Stop:
# Default-Start:     S
# Default-Stop:
# Short-Description: Replays failed uevents and creates additional devices.
# Description:       Replays any failed uevents that were skipped due to
#                    slow hardware initialization, and creates those needed
#                    device nodes
#
# X-LFS-Provided-By: LFS
### END INIT INFO

```

```

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Retrying failed uevents, if any..."

        # As of udev-186, the --run option is no longer valid
        #rundir=$(/sbin/udevadm info --run)
        rundir=/run/udev
        # From Debian: "copy the rules generated before / was mounted
        # read-write":

        for file in ${rundir}/tmp-rules--*; do
            dest=${file##*tmp-rules--}
            [ "$dest" = '*' ] && break
            cat $file >> /etc/udev/rules.d/$dest
            rm -f $file
        done

        # Re-trigger the uevents that may have failed,
        # in hope they will succeed now
        /bin/sed -e 's/#.*$//' /etc/sysconfig/udev_retry | /bin/grep -v '^$' | \
        while read line ; do
            for subsystem in $line ; do
                /sbin/udevadm trigger --subsystem-match=$subsystem --action=add
            done
        done

        # Now wait for udevd to process the uevents we triggered
        if ! is_true "$OMIT_UDEV_RETRY_SETTLE"; then
            /sbin/udevadm settle
        fi

        evaluate_retval
        ;;

    *)
        echo "Usage ${0} {start}"
        exit 1
        ;;
esac

exit 0

# End udev_retry

```

D.11. /etc/rc.d/init.d/cleanfs

```

#!/bin/sh
#####
# Begin cleanfs
#
# Description : Clean file system
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0

```

```

#
#####

### BEGIN INIT INFO
# Provides:          cleanfs
# Required-Start:    $local_fs
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:     S
# Default-Stop:
# Short-Description: Cleans temporary directories early in the boot process.
# Description:       Cleans temporary directories /run, /var/lock, and
#                    optionally, /tmp. cleanfs also creates /run/utmp
#                    and any files defined in /etc/sysconfig/createfiles.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

# Function to create files/directory on boot.
create_files()
{
    # Input to file descriptor 9 and output to stdin (redirection)
    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
                            ;;
                    *)
                        log_warning_msg "\nUnknown device type: ${dtype}"
                        ;;
                    esac
                ;;
            *)
                log_warning_msg "\nUnknown type: ${type}"
            ;;
        esac
    done
}

```

```

        continue
        ;;
    esac

    # Set up the permissions, too.
    chown ${usr}:${grp} "${name}"
    chmod ${perm} "${name}"
fi
done

# Close file descriptor 9 (end redirection)
exec 0>&9 9>&-
return 0
}

case "${1}" in
start)
    log_info_msg "Cleaning file systems:"

    if [ "${SKIPTMPCLEAN}" = "" ]; then
        log_info_msg2 " /tmp"
        cd /tmp &&
        find . -xdev -mindepth 1 ! -name lost+found -delete || failed=1
    fi

    > /run/utmp

    if grep -q '^utmp:' /etc/group ; then
        chmod 664 /run/utmp
        chgrp utmp /run/utmp
    fi

    (exit ${failed})
    evaluate_retval

    if grep -E -qv '^(#|$)' /etc/sysconfig/createfiles 2>/dev/null; then
        log_info_msg "Creating files and directories... "
        create_files      # Always returns 0
        evaluate_retval
    fi

    exit $failed
    ;;
*)
    echo "Usage: ${0} {start}"
    exit 1
    ;;
esac

# End cleanfs

```

D.12. /etc/rc.d/init.d/console

```

#!/bin/sh
#####
# Begin console
#
# Description : Sets keymap and screen font
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org

```

```

#           Alexander E. Patrakov
#           DJ Lucas - dj@linuxfromscratch.org
# Update    : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version   : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          console
# Required-Start:    $local_fs
# Should-Start:     udev_retry
# Required-Stop:
# Should-Stop:
# Default-Start:    S
# Default-Stop:
# Short-Description: Sets up a localised console.
# Description:       Sets up fonts and language settings for the user's
#                   local as defined by /etc/sysconfig/console.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

# Native English speakers probably don't have /etc/sysconfig/console at all
[ -r /etc/sysconfig/console ] && . /etc/sysconfig/console

failed=0

case "${1}" in
    start)
        # See if we need to do anything
        if [ -z "${KEYMAP}" ] && [ -z "${KEYMAP_CORRECTIONS}" ] &&
            [ -z "${FONT}" ] && [ -z "${LEGACY_CHARSET}" ] &&
            ! is_true "${UNICODE}"; then
            exit 0
        fi

        # There should be no bogus failures below this line!
        log_info_msg "Setting up Linux console..."

        # 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.

```

```

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
[ "${use_fb}" == "1" ] || [ -z "${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

exit $failed
;;

*)
echo "Usage:  ${0} {start}"
exit 1
;;

esac

# End console

```

D.13. /etc/rc.d/init.d/localnet

```

#!/bin/sh
#####
# Begin localnet
#
# Description : Loopback device
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          localnet
# Required-Start:    mountvirtfs
# Should-Start:      modules

```

```

# Required-Stop:
# Should-Stop:
# Default-Start:      S
# Default-Stop:      0 6
# Short-Description:  Starts the local network.
# Description:        Sets the hostname of the machine and starts the
#                     loopback interface.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions
[ -r /etc/sysconfig/network ] && . /etc/sysconfig/network
[ -r /etc/hostname ] && HOSTNAME=`cat /etc/hostname`

case "${1}" in
    start)
        log_info_msg "Bringing up the loopback interface..."
        ip addr add 127.0.0.1/8 label lo dev lo
        ip link set lo up
        evaluate_retval

        log_info_msg "Setting hostname to ${HOSTNAME}..."
        hostname ${HOSTNAME}
        evaluate_retval
        ;;

    stop)
        log_info_msg "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

exit 0

# End localnet

```

D.14. /etc/rc.d/init.d/sysctl

```

#!/bin/sh
#####
# Begin sysctl
#
# Description : File uses /etc/sysctl.conf to set kernel runtime

```

```

#           parameters
#
# Authors   : Nathan Coulson (nathan@linuxfromscratch.org)
#           : Matthew Burgess (matthew@linuxfromscratch.org)
#           : DJ Lucas - dj@linuxfromscratch.org
# Update    : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version   : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          sysctl
# Required-Start:    mountvirtfs
# Should-Start:      console
# Required-Stop:
# Should-Stop:
# Default-Start:     S
# Default-Stop:
# Short-Description: Makes changes to the proc filesystem
# Description:       Makes changes to the proc filesystem as defined in
#                   /etc/sysctl.conf. See 'man sysctl(8)'.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        if [ -f "/etc/sysctl.conf" ]; then
            log_info_msg "Setting kernel runtime parameters..."
            sysctl -q -p
            evaluate_retval
        fi
        ;;

    status)
        sysctl -a
        ;;

    *)
        echo "Usage: ${0} {start|status}"
        exit 1
        ;;
esac

exit 0

# End sysctl

```

D.15. /etc/rc.d/init.d/sysklogd

```

#!/bin/sh
#####
# Begin sysklogd
#
# Description : Sysklogd loader
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#             : DJ Lucas - dj@linuxfromscratch.org

```



```

# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          $syslog
# Required-Start:    $first localnet
# Should-Start:
# Required-Stop:     $local_fs
# Should-Stop:       sendsignals
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts kernel and system log daemons.
# Description:       Starts kernel and system log daemons.
#                    /etc/fstab.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Starting system log daemon..."
        parms=${SYSKLOGD_PARMS-'-m 0'}
        start_daemon /sbin/syslogd $parms
        evaluate_retval

        log_info_msg "Starting kernel log daemon..."
        start_daemon /sbin/klogd
        evaluate_retval
        ;;

    stop)
        log_info_msg "Stopping kernel log daemon..."
        killproc /sbin/klogd
        evaluate_retval

        log_info_msg "Stopping system log daemon..."
        killproc /sbin/syslogd
        evaluate_retval
        ;;

    reload)
        log_info_msg "Reloading system log daemon config file..."
        pid=`pidofproc syslogd`
        kill -HUP "${pid}"
        evaluate_retval
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;

    status)
        statusproc /sbin/syslogd
        statusproc klogd
        ;;

```

```

*)
    echo "Usage: ${0} {start|stop|reload|restart|status}"
    exit 1
    ;;
esac

exit 0

# End sysklogd

```

D.16. /etc/rc.d/init.d/network

```

#!/bin/sh
#####
# Begin network
#
# Description : Network Control Script
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#               Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpflaming@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

### BEGIN INIT INFO
# Provides:          $network
# Required-Start:    $local_fs localnet swap
# Should-Start:      $syslog firewalld iptables nftables
# Required-Stop:     $local_fs localnet swap
# Should-Stop:       $syslog firewalld iptables nftables
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts and configures network interfaces.
# Description:       Starts and configures network interfaces.
# X-LFS-Provided-By: LFS
### END INIT INFO

case "${1}" in
    start)
        # if the default route exists, network is already configured
        if ip route | grep -q "^default"; then return 0; fi
        # Start all network interfaces
        for file in /etc/sysconfig/ifconfig.*
        do
            interface=${file##*/ifconfig.}

            # Skip if $file is * (because nothing was found)
            if [ "${interface}" = "*" ]; then continue; fi

            /sbin/ifup ${interface}
        done
        ;;
    stop)
        # Unmount any network mounted file systems

```

```

umount --all --force --types nfs,cifs,nfs4

# Reverse list
net_files=""
for file in /etc/sysconfig/ifconfig.*
do
    net_files="${file} ${net_files}"
done

# Stop all network interfaces
for file in ${net_files}
do
    interface=${file##*/ifconfig.}

    # Skip if $file is * (because nothing was found)
    if [ "${interface}" = "*" ]; then continue; fi

    # See if interface exists
    if [ ! -e /sys/class/net/${interface} ]; then continue; fi

    # Is interface UP?
    ip link show $interface 2>/dev/null | grep -q "state UP"
    if [ $? -ne 0 ]; then continue; fi

    /sbin/ifdown ${interface}
done
;;

restart)
    ${0} stop
    sleep 1
    ${0} start
    ;;

*)
    echo "Usage: ${0} {start|stop|restart}"
    exit 1
    ;;
esac

exit 0

# End network

```

D.17. /etc/rc.d/init.d/sendsignals

```

#!/bin/sh
#####
# Begin sendsignals
#
# Description : Sendsignals Script
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#              : DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
#####

```

```

### BEGIN INIT INFO
# Provides:          sendsignals
# Required-Start:
# Should-Start:
# Required-Stop:    $local_fs swap localnet
# Should-Stop:
# Default-Start:
# Default-Stop:    0 6
# Short-Description: Attempts to kill remaining processes.
# Description:      Attempts to kill remaining processes.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    stop)
        omit=$(pidof mdmon)
        [ -n "$omit" ] && omit="-o $omit"

        log_info_msg "Sending all processes the TERM signal..."
        killall5 -15 $omit
        error_value=${?}

        sleep ${KILLDELAY}

        if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
            log_success_msg
        else
            log_failure_msg
        fi

        log_info_msg "Sending all processes the KILL signal..."
        killall5 -9 $omit
        error_value=${?}

        sleep ${KILLDELAY}

        if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
            log_success_msg
        else
            log_failure_msg
        fi
        ;;
    *)
        echo "Usage: ${0} {stop}"
        exit 1
        ;;
esac

exit 0

# End sendsignals

```

D.18. /etc/rc.d/init.d/reboot

```

#!/bin/sh
#####

```

```

# Begin reboot
#
# Description : Reboot Scripts
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#              : DJ Lucas - dj@linuxfromscratch.org
# Updates      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#              : Pierre Labastie - pierre@linuxfromscratch.org
#
# Version      : LFS 7.0
#
# Notes        : Update March 24th, 2022: change "stop" to "start".
#              : Add the $last facility to Required-start
#
#####

### BEGIN INIT INFO
# Provides:          reboot
# Required-Start:    $last
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:     6
# Default-Stop:
# Short-Description: Reboots the system.
# Description:       Reboots the System.
# X-LFS-Provided-By: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Restarting system..."
        reboot -d -f -i
        ;;

    *)
        echo "Usage: ${0} {start}"
        exit 1
        ;;
esac

# End reboot

```

D.19. /etc/rc.d/init.d/halt

```

#!/bin/sh
#####
# Begin halt
#
# Description : Halt Script
#
# Authors      : Gerard Beekmans - gerard@linuxfromscratch.org
#              : DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#              : Pierre Labastie - pierre@linuxfromscratch.org
#
# Version      : LFS 7.0

```

```

#
# Notes      : Update March 24th, 2022: change "stop" to "start".
#            Add the $last facility to Required-start
#
#####

### BEGIN INIT INFO
# Provides:          halt
# Required-Start:    $last
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:     0
# Default-Stop:
# Short-Description: Halts the system.
# Description:       Halts the System.
# X-LFS-Provided-By: LFS
### END INIT INFO

case "${1}" in
    start)
        halt -d -f -i -p
        ;;

    *)
        echo "Usage: {start}"
        exit 1
        ;;
esac

# End halt

```

D.20. /etc/rc.d/init.d/template

```

#!/bin/sh
#####
# Begin scriptname
#
# Description :
#
# Authors      :
#
# Version      : LFS x.x
#
# Notes       :
#
#####

### BEGIN INIT INFO
# Provides:          template
# Required-Start:
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:
# Default-Stop:
# Short-Description:
# Description:
# X-LFS-Provided-By:
### END INIT INFO

```

```

. /lib/lsb/init-functions

case "${1}" in
    start)
        log_info_msg "Starting..."
        # if it is possible to use start_daemon
        start_daemon fully_qualified_path
        # if it is not possible to use start_daemon
        # (command to start the daemon is not simple enough)
        if ! pidofproc daemon_name_as_reported_by_ps >/dev/null; then
            command_to_start_the_service
        fi
        evaluate_retval
        ;;

    stop)
        log_info_msg "Stopping..."
        # if it is possible to use killproc
        killproc fully_qualified_path
        # if it is not possible to use killproc
        # (the daemon shouldn't be stopped by killing it)
        if pidofproc daemon_name_as_reported_by_ps >/dev/null; then
            command_to_stop_the_service
        fi
        evaluate_retval
        ;;

    restart)
        ${0} stop
        sleep 1
        ${0} start
        ;;

    *)
        echo "Usage: ${0} {start|stop|restart}"
        exit 1
        ;;
esac

exit 0

# End scriptname

```

D.21. /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 its own line, and any options that you want
# passed to the module should follow it. The line delimiter is either

```

```
# a space or a tab.
#####

# End /etc/sysconfig/modules
```

D.22. /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 device
#                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.23. /etc/sysconfig/udev-retry

```
#####
# Begin /etc/sysconfig/udev_retry
#
# Description : udev_retry script configuration
#
# Authors      :
#
# Version      : 00.00
#
# Notes        : Each subsystem that may need to be re-triggered after mountfs
#                runs should be listed in this file. Probable subsystems to be
#                listed here are rtc (due to /var/lib/hwclock/adjtime) and sound
#                (due to both /var/lib/alsa/asound.state and /usr/sbin/alsactl).
#                Entries are whitespace-separated.
#####

rtc

# End /etc/sysconfig/udev_retry
```


D.24. /sbin/ifup

```
#!/bin/sh
#####
# Begin /sbin/ifup
#
# Description : Interface Up
#
# Authors      : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpflaming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
#
# Version      : LFS 7.7
#
# Notes       : The IFCONFIG variable is passed to the SERVICE script
#               in the /lib/services directory, to indicate what file the
#               service should source to get interface specifications.
#
#####
up()
{
    log_info_msg "Bringing up the ${1} interface..."

    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
        log_failure_msg "Interface ${IFACE} doesn't exist."
        exit 1
    fi

    evaluate_retval
}

RELEASE="7.7"

USAGE="Usage: $0 [ -hV ] [--help] [--version] interface"
VERSTR="LFS ifup, version ${RELEASE}"

while [ $# -gt 0 ]; do
    case "$1" in
        --help | -h)      help="y"; break ;;
        --version | -V)   echo "${VERSTR}"; exit 0 ;;
        -*)               echo "ifup: ${1}: invalid option" >&2
                          echo "${USAGE}" >& 2
                          exit 2 ;;
        *)               break ;;
    esac
done
```

```

if [ -n "$help" ]; then
    echo "${VERSTR}"
    echo "${USAGE}"
    echo
    cat << HERE_EOF
ifup is used to bring up a network interface. The interface
parameter, e.g. eth0 or eth0:2, must match the trailing part of the
interface specifications file, e.g. /etc/sysconfig/ifconfig.eth0:2.

HERE_EOF
    exit 0
fi

file=/etc/sysconfig/ifconfig.${1}

# Skip backup files
[ "${file}" = "${file%}"~" ] || exit 0

. /lib/lsb/init-functions

if [ ! -r "${file}" ]; then
    log_failure_msg "Unable to bring up ${1} interface! ${file} is missing or cannot be accessed"
    exit 1
fi

. $file

if [ "$IFACE" = "" ]; then
    log_failure_msg "Unable to bring up ${1} interface! ${file} does not define an interface [IF"
    exit 1
fi

# Do not process this service if started by boot, and ONBOOT
# is not set to yes
if [ "${IN_BOOT}" = "1" -a "${ONBOOT}" != "yes" ]; then
    exit 0
fi

# Bring up the interface
if [ "$VIRTINT" != "yes" ]; then
    up ${IFACE}
fi

for S in ${SERVICE}; do
    if [ ! -x "/lib/services/${S}" ]; then
        MSG="\nUnable to process ${file}. Either "
        MSG="${MSG}the SERVICE '${S}' was not present "
        MSG="${MSG}or cannot be executed."
        log_failure_msg "$MSG"
        exit 1
    fi
done

if [ "${SERVICE}" = "wpa" ]; then log_success_msg; fi

# Create/configure the interface
for S in ${SERVICE}; do
    IFCONFIG=${file} /lib/services/${S} ${IFACE} up
done

```

```

# Set link up virtual interfaces
if [ "${VIRTINT}" == "yes" ]; then
    up ${IFACE}
fi

# Bring up any additional interface components
for I in $INTERFACE_COMPONENTS; do up $I; done

# Set MTU if requested. Check if MTU has a "good" value.
if test -n "${MTU}"; then
    if [[ ${MTU} =~ ^[0-9]+$ ]] && [[ $MTU -ge 68 ]]; then
        for I in $IFACE $INTERFACE_COMPONENTS; do
            ip link set dev $I mtu $MTU;
        done
    else
        log_info_msg2 "Invalid MTU $MTU"
    fi
fi

# Set the route default gateway if requested
if [ -n "${GATEWAY}" ]; then
    if ip route | grep -q default; then
        log_warning_msg "Gateway already setup; skipping."
    else
        log_info_msg "Adding default gateway ${GATEWAY} to the ${IFACE} interface..."
        ip route add default via ${GATEWAY} dev ${IFACE}
        evaluate_retval
    fi
fi

# End /sbin/ifup

```

D.25. /sbin/ifdown

```

#!/bin/bash
#####
# Begin /sbin/ifdown
#
# Description : Interface Down
#
# Authors      : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpflaming@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
# Notes        : the IFCONFIG variable is passed to the scripts found
#               in the /lib/services directory, to indicate what file the
#               service should source to get interface specifications.
#
#####
RELEASE="7.0"

USAGE="Usage: $0 [ -hV ] [--help] [--version] interface"
VERSTR="LFS ifdown, version ${RELEASE}"

while [ $# -gt 0 ]; do
    case "$1" in
        --help | -h)    help="y"; break ;;

```

```

--version | -V) echo "${VERSTR}"; exit 0 ;;

*)
    echo "ifup: ${1}: invalid option" >&2
    echo "${USAGE}" >& 2
    exit 2 ;;

*)
    break ;;
esac
done

if [ -n "$help" ]; then
    echo "${VERSTR}"
    echo "${USAGE}"
    echo
    cat << HERE_EOF
ifdown is used to bring down a network interface. The interface
parameter, e.g. eth0 or eth0:2, must match the trailing part of the
interface specifications file, e.g. /etc/sysconfig/ifconfig.eth0:2.

HERE_EOF
    exit 0
fi

file=/etc/sysconfig/ifconfig.${1}

# Skip backup files
[ "${file}" = "${file%*"~"}" ] || exit 0

. /lib/lsb/init-functions

if [ ! -r "${file}" ]; then
    log_warning_msg "${file} is missing or cannot be accessed."
    exit 1
fi

. ${file}

if [ "$IFACE" = "" ]; then
    log_failure_msg "${file} does not define an interface [IFACE]."
    exit 1
fi

# We only need to first service to bring down the interface
S=`echo ${SERVICE} | cut -f1 -d " "`

if ip link show ${IFACE} > /dev/null 2>&1; then
    if [ -n "${S}" -a -x "/lib/services/${S}" ]; then
        IFCONFIG=${file} /lib/services/${S} ${IFACE} down
    else
        MSG="Unable to process ${file}. Either "
        MSG="${MSG}the SERVICE variable was not set "
        MSG="${MSG}or the specified service cannot be executed."
        log_failure_msg "$MSG"
        exit 1
    fi
else
    log_warning_msg "Interface ${1} doesn't exist."
fi

# Leave the interface up if there are additional interfaces in the device

```

```

link_status=`ip link show ${IFACE} 2>/dev/null`

if [ -n "${link_status}" ]; then
    if [ "$(echo "${link_status}" | grep UP)" != "" ]; then
        if [ "$(ip addr show ${IFACE} | grep 'inet ')" == "" ]; then
            log_info_msg "Bringing down the ${IFACE} interface..."
            ip link set ${IFACE} down
            evaluate_retval
        fi
    fi
fi

# End /sbin/ifdown

```

D.26. /lib/services/ipv4-static

```

#!/bin/sh
#####
# Begin /lib/services/ipv4-static
#
# Description : IPV4 Static Boot Script
#
# Authors      : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpflaming@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

. /lib/lsb/init-functions
. ${IFCONFIG}

if [ -z "${IP}" ]; then
    log_failure_msg "\nIP variable missing from ${IFCONFIG}, cannot continue."
    exit 1
fi

if [ -z "${PREFIX}" -a -z "${PEER}" ]; then
    log_warning_msg "\nPREFIX variable missing from ${IFCONFIG}, assuming 24."
    PREFIX=24
    args="${args} ${IP}/${PREFIX}"
elif [ -n "${PREFIX}" -a -n "${PEER}" ]; then
    log_failure_msg "\nPREFIX and PEER both specified in ${IFCONFIG}, cannot continue."
    exit 1
elif [ -n "${PREFIX}" ]; then
    args="${args} ${IP}/${PREFIX}"
elif [ -n "${PEER}" ]; then
    args="${args} ${IP} peer ${PEER}"
fi

if [ -n "${LABEL}" ]; then
    args="${args} label ${LABEL}"
fi

if [ -n "${BROADCAST}" ]; then
    args="${args} broadcast ${BROADCAST}"

```

```

fi
case "${2}" in
  up)
    if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP}/)" = "" ]; then
      log_info_msg "Adding IPv4 address ${IP} to the ${1} interface..."
      ip addr add ${args} dev ${1}
      evaluate_retval
    else
      log_warning_msg "Cannot add IPv4 address ${IP} to ${1}.  Already present."
    fi
    ;;
  down)
    if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP}/)" != "" ]; then
      log_info_msg "Removing IPv4 address ${IP} from the ${1} interface..."
      ip addr del ${args} dev ${1}
      evaluate_retval
    fi

    if [ -n "${GATEWAY}" ]; then
      # Only remove the gateway if there are no remaining ipv4 addresses
      if [ "$(ip addr show ${1} 2>/dev/null | grep 'inet ')" != "" ]; then
        log_info_msg "Removing default gateway..."
        ip route del default
        evaluate_retval
      fi
    fi
    ;;
  *)
    echo "Usage: ${0} [interface] {up|down}"
    exit 1
  ;;
esac

# End /lib/services/ipv4-static

```

D.27. /lib/services/ipv4-static-route

```

#!/bin/sh
#####
# Begin /lib/services/ipv4-static-route
#
# Description : IPV4 Static Route Script
#
# Authors      : Kevin P. Fleming - kp Fleming@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update       : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version      : LFS 7.0
#
#####

. /lib/lsb/init-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 "
    ;;

    (*)
        log_failure_msg "Unknown route type ({TYPE}) in ${IFCONFIG}, cannot continue."
        exit 1
    ;;
esac

if [ -n "${GATEWAY}" ]; then
    MSG="The GATEWAY variable cannot be set in ${IFCONFIG} for static routes.\n"
    log_failure_msg "$MSG Use STATIC_GATEWAY only, cannot continue"
    exit 1
fi

if [ -n "${need_ip}" ]; then
    if [ -z "${IP}" ]; then
        log_failure_msg "IP variable missing from ${IFCONFIG}, cannot continue."
        exit 1
    fi

    if [ -z "${PREFIX}" ]; then
        log_failure_msg "PREFIX variable missing from ${IFCONFIG}, cannot continue."
        exit 1
    fi

    args="{args} ${IP}/${PREFIX}"
    desc="{desc}${IP}/${PREFIX}"
fi

if [ -n "${need_gateway}" ]; then
    if [ -z "${STATIC_GATEWAY}" ]; then
        log_failure_msg "STATIC_GATEWAY variable missing from ${IFCONFIG}, cannot continue."
        exit 1
    fi
    args="{args} via ${STATIC_GATEWAY}"
fi

if [ -n "${SOURCE}" ]; then
    args="{args} src ${SOURCE}"
fi

case "${2}" in
    up)
        log_info_msg "Adding '${desc}' route to the ${1} interface..."

```

```
    ip route add ${args} dev ${1}
    evaluate_retval
;;

down)
    log_info_msg "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 /lib/services/ipv4-static-route
```


付録E Udev 設定ルール

本付録にて udev ルールを列記します。インストール手順は「Eudev-3.2.11」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=="ipp[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

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.
- 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, MERCHANTABILITY, 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

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 © 1999-2022 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

- Acl: 119
- Attr: 118
- Autoconf: 151
- Automake: 152
- Bash: 139
 - ツール: 53
- Bash: 139
 - ツール: 53
- Bc: 105
- Binutils: 111
 - ツール, 1回め: 40
 - ツール, 2回め: 66
- Binutils: 111
 - ツール, 1回め: 40
 - ツール, 2回め: 66
- Binutils: 111
 - ツール, 1回め: 40
 - ツール, 2回め: 66
- Bison: 137
 - ツール: 75
- Bison: 137
 - ツール: 75
- ブートスクリプト: 210
 - 利用方法: 219
- ブートスクリプト: 210
 - 利用方法: 219
- Bzip2: 97
- Check: 168
- Coreutils: 164
 - ツール: 54
- Coreutils: 164
 - ツール: 54
- DejaGNU: 110
- Diffutils: 169
 - ツール: 55
- Diffutils: 169
 - ツール: 55
- E2fsprogs: 201
- Eudev: 190
 - 設定: 190
- Eudev: 190
 - 設定: 190
- Expat: 143
- Expect: 109
- File: 102
 - ツール: 56
- File: 102
 - ツール: 56
- Findutils: 171
 - ツール: 57
- Findutils: 171
 - ツール: 57
- Flex: 106
- Gawk: 170
 - ツール: 58
- Gawk: 170
 - ツール: 58
- GCC: 125
 - ツール, 1回め: 42
 - ツール, 2回め: 67
 - ツール, libstdc++ 1 回め: 48
- GCC: 125
 - ツール, 1回め: 42
 - ツール, 2回め: 67
 - ツール, libstdc++ 1 回め: 48
- GCC: 125
 - ツール, 1回め: 42
 - ツール, 2回め: 67
 - ツール, libstdc++ 1 回め: 48
- GCC: 125
 - ツール, 1回め: 42
 - ツール, 2回め: 67
 - ツール, libstdc++ 1 回め: 48
- GDBM: 141
- Gettext: 135
 - ツール: 74
- Gettext: 135
 - ツール: 74
- Glibc: 89
 - ツール: 45
- Glibc: 89
 - ツール: 45
- GMP: 114
- Gperf: 142
- Grep: 138
 - ツール: 59
- Grep: 138
 - ツール: 59
- Groff: 172
- GRUB: 174
- Gzip: 176
 - ツール: 60
- Gzip: 176
 - ツール: 60
- Iana-Etc: 88
- Inetutils: 144
- Intltool: 150
- IPRoute2: 177
- Kbd: 179
- Kmod: 155
- Less: 146
- Libcap: 120
- Libelf: 157
- libffi: 158
- Libpipeline: 181
- Libtool: 140
- Linux: 233
 - ツール, API ヘッダー: 44
- Linux: 233
 - ツール, API ヘッダー: 44
- M4: 104
 - ツール: 50
- M4: 104
 - ツール: 50
- Make: 182
 - ツール: 61

Make: 182
 ツール: 61
 Man-DB: 191
 Man-pages: 87
 Meson: 163
 MPC: 117
 MPFR: 116
 Ncurses: 130
 ツール: 51
 Ncurses: 130
 ツール: 51
 Ninja: 162
 OpenSSL: 153
 Patch: 183
 ツール: 62
 Patch: 183
 ツール: 62
 Perl: 147
 ツール: 76
 Perl: 147
 ツール: 76
 Pkgconfig: 129
 Procps-ng: 194
 Psmisc: 134
 Python: 159
 一時的: 77
 Python: 159
 一時的: 77
 rc.site: 224
 Readline: 103
 Sed: 133
 ツール: 63
 Sed: 133
 ツール: 63
 Shadow: 121
 設定: 122
 Shadow: 121
 設定: 122
 Sysklogd: 204
 設定: 204
 Sysklogd: 204
 設定: 204
 Sysvinit: 205
 設定: 220
 Sysvinit: 205
 設定: 220
 Tar: 184
 ツール: 64
 Tar: 184
 ツール: 64
 Tcl: 107
 Texinfo: 185
 一時的: 78
 Texinfo: 185
 一時的: 78
 Udev
 利用方法: 212
 Util-linux: 196
 ツール: 79
 Util-linux: 196
 ツール: 79

Vim: 187
 wheel: 161
 XML::Parser: 149
 Xz: 99
 ツール: 65
 Xz: 99
 ツール: 65
 Zlib: 96
 zstd: 101

プログラム

[: 164, 165
 2to3: 159
 accessdb: 191, 192
 aclocal: 152, 152
 aclocal-1.16: 152, 152
 addftinfo: 172, 172
 addpart: 196, 197
 addr2line: 111, 112
 afmtodit: 172, 172
 agetty: 196, 197
 apropos: 191, 192
 ar: 111, 112
 as: 111, 112
 attr: 118, 118
 autoconf: 151, 151
 autoheader: 151, 151
 autom4te: 151, 151
 automake: 152, 152
 automake-1.16: 152, 152
 autopoint: 135, 135
 autoreconf: 151, 151
 autoscan: 151, 151
 autoupdate: 151, 151
 awk: 170, 170
 b2sum: 164, 165
 badblocks: 201, 202
 base64: 164, 165, 164, 165
 base64: 164, 165, 164, 165
 basename: 164, 165
 basenc: 164, 165
 bash: 139, 139
 bashbug: 139, 139
 bc: 105, 105
 bison: 137, 137
 blkdiscard: 196, 197
 blkid: 196, 197
 blkzone: 196, 197
 blockdev: 196, 197
 bootlogd: 205, 205
 bridge: 177, 177
 bunzip2: 97, 98
 bzip2: 97, 98
 bzip2: 97, 98
 bzdiff: 97, 98
 bzegrep: 97, 98
 bzfgrep: 97, 98
 bzgrep: 97, 98
 bzip2: 97, 98
 bzip2recover: 97, 98

bzless: 97, 98
 bzmores: 97, 98
 c++: 125, 128
 c++filt: 111, 112
 cal: 196, 197
 capsh: 120, 120
 captinfo: 130, 131
 cat: 164, 165
 catman: 191, 192
 cc: 125, 128
 cfdisk: 196, 197
 chacl: 119, 119
 chage: 121, 123
 chattr: 201, 202
 chcon: 164, 165
 chcpu: 196, 197
 checkmk: 168, 168
 chem: 172, 172
 chfn: 121, 123
 chgpasswd: 121, 123
 chgrp: 164, 165
 chmem: 196, 197
 chmod: 164, 165
 choom: 196, 197
 chown: 164, 165
 chpasswd: 121, 123
 chroot: 164, 165
 chrt: 196, 197
 chsh: 121, 123
 chvt: 179, 180
 cksum: 164, 165
 clear: 130, 131
 cmp: 169, 169
 col: 196, 197
 colcrt: 196, 197
 colrm: 196, 197
 column: 196, 197
 comm: 164, 165
 compile_et: 201, 202
 corelist: 147, 148
 cp: 164, 165
 cpan: 147, 148
 cpp: 125, 128
 csplit: 164, 165
 ctrlaltdel: 196, 197
 ctstat: 177, 177
 cut: 164, 165
 c_rehash: 153, 153
 date: 164, 165
 dc: 105, 105
 dd: 164, 165
 deallocvt: 179, 180
 debugfs: 201, 202
 dejagnu: 110, 110
 delpart: 196, 197
 depmod: 155, 155
 df: 164, 165
 diff: 169, 169
 diff3: 169, 169
 dir: 164, 165
 dircolors: 164, 166
 dirname: 164, 166
 dmesg: 196, 197
 dnsdomainname: 144, 144
 du: 164, 166
 dumpe2fs: 201, 202
 dumpkeys: 179, 180
 e2freefrag: 201, 202
 e2fsck: 201, 202
 e2image: 201, 202
 e2label: 201, 202
 e2mmpstatus: 201, 202
 e2scrub: 201, 202
 e2scrub_all: 201, 202
 e2undo: 201, 202
 e4crypt: 201, 202
 e4defrag: 201, 202
 echo: 164, 166
 egrep: 138, 138
 eject: 196, 197
 elfedit: 111, 112
 enc2xs: 147, 148
 encguess: 147, 148
 env: 164, 166
 envsubst: 135, 135
 eqn: 172, 172
 eqn2graph: 172, 172
 ex: 187, 188
 expand: 164, 166
 expect: 109, 109
 expiry: 121, 123
 expr: 164, 166
 factor: 164, 166
 faillog: 121, 123
 fallocate: 196, 197
 false: 164, 166
 fdisk: 196, 197
 fgconsole: 179, 180
 fgrep: 138, 138
 file: 102, 102
 filefrag: 201, 202
 findcore: 196, 197
 find: 171, 171
 findfs: 196, 197
 findmnt: 196, 197
 flex: 106, 106
 flex++: 106, 106
 flock: 196, 197
 fmt: 164, 166
 fold: 164, 166
 free: 194, 194
 fsck: 196, 197
 fsck.cramfs: 196, 198
 fsck.ext2: 201, 202
 fsck.ext3: 201, 202
 fsck.ext4: 201, 202
 fsck.minix: 196, 198
 fsfreeze: 196, 198
 fstab-decode: 205, 205
 fstrim: 196, 198
 ftp: 144, 145
 fuser: 134, 134

g++: 125, 128
gawk: 170, 170
gawk-5.1.1: 170, 170
gcc: 125, 128
gc-ar: 125, 128
gc-nm: 125, 128
gc-ranlib: 125, 128
gcov: 125, 128
gcov-dump: 125, 128
gcov-tool: 125, 128
gdbmtool: 141, 141
gdbm_dump: 141, 141
gdbm_load: 141, 141
gdifffmk: 172, 172
gencat: 89, 93
genl: 177, 177
getcap: 120, 120
getconf: 89, 94
getent: 89, 94
getfacl: 119, 119
getfattr: 118, 118
getkeycodes: 179, 180
getopt: 196, 198
getpcaps: 120, 120
getsubids: 121, 123
gettext: 135, 135
gettext.sh: 135, 135
gettextize: 135, 135
glilypond: 172, 172
gpasswd: 121, 123
gperf: 142, 142
gperl: 172, 172
gpinyin: 172, 172
gprof: 111, 112
gprofng: 111, 112
grap2graph: 172, 172
grep: 138, 138
grn: 172, 172
grodvi: 172, 172
groff: 172, 172
groffer: 172, 172
grog: 172, 172
grolbp: 172, 173
grolj4: 172, 173
gropdf: 172, 173
grops: 172, 173
grotty: 172, 173
groupadd: 121, 123
groupdel: 121, 123
groupmems: 121, 123
groupmod: 121, 123
groups: 164, 166
grpck: 121, 123
grpconv: 121, 123
grpunconv: 121, 123
grub-bios-setup: 174, 174
grub-editenv: 174, 174
grub-file: 174, 174
grub-fstest: 174, 174
grub-glue-efi: 174, 174
grub-install: 174, 174
grub-kbdcomp: 174, 175
grub-macbless: 174, 175
grub-menulst2cfg: 174, 175
grub-mkconfig: 174, 175
grub-mkimage: 174, 175
grub-mklayout: 174, 175
grub-mknetdir: 174, 175
grub-mkpasswd-pbkdf2: 174, 175
grub-mkrelpath: 174, 175
grub-mkrescue: 174, 175
grub-mkstandalone: 174, 175
grub-ofpathname: 174, 175
grub-probe: 174, 175
grub-reboot: 174, 175
grub-render-label: 174, 175
grub-script-check: 174, 175
grub-set-default: 174, 175
grub-setup: 174, 175
grub-syslinux2cfg: 174, 175
gunzip: 176, 176
gzexe: 176, 176
gzip: 176, 176
h2ph: 147, 148
h2xs: 147, 148
halt: 205, 205
head: 164, 166
hexdump: 196, 198
hostid: 164, 166
hostname: 144, 145
hpftodit: 172, 173
hwclock: 196, 198
i386: 196, 198
iconv: 89, 94
iconvconfig: 89, 94
id: 164, 166
idle3: 159
ifcfg: 177, 177
ifconfig: 144, 145
ifnames: 151, 151
ifstat: 177, 177
indxbib: 172, 173
info: 185, 185
infocmp: 130, 131
infotocap: 130, 131
init: 205, 205
insmod: 155, 155
install: 164, 166
install-info: 185, 185
instmodsh: 147, 148
intltool-extract: 150, 150
intltool-merge: 150, 150
intltool-prepare: 150, 150
intltool-update: 150, 150
intltoolize: 150, 150
ionice: 196, 198
ip: 177, 177
ipcmk: 196, 198
ipcrm: 196, 198
ipcs: 196, 198
irqtop: 196, 198
isosize: 196, 198

join: 164, 166
 json_pp: 147, 148
 kbdfinfo: 179, 180
 kbdrate: 179, 180
 kbd_mode: 179, 180
 kill: 196, 198
 killall: 134, 134
 killall5: 205, 205
 klogd: 204, 204
 kmod: 155, 155
 last: 196, 198
 lastb: 196, 198
 lastlog: 121, 123
 ld: 111, 112
 ld.bfd: 111, 112
 ld.gold: 111, 112
 ldattach: 196, 198
 ldconfig: 89, 94
 ldd: 89, 94
 lddlibc4: 89, 94
 less: 146, 146
 lessecho: 146, 146
 lesskey: 146, 146
 lex: 106, 106
 lexgrog: 191, 192
 lfskernel-5.19.2: 233, 236
 libasan: 125, 128
 libatomic: 125, 128
 libcc1: 125, 128
 libnetcfg: 147, 148
 libtool: 140, 140
 libtoolize: 140, 140
 link: 164, 166
 linux32: 196, 198
 linux64: 196, 198
 lkbib: 172, 173
 ln: 164, 166
 lnstat: 177, 177
 loadkeys: 179, 180
 loadunimap: 179, 180
 locale: 89, 94
 localedef: 89, 94
 locate: 171, 171
 logger: 196, 198
 login: 121, 123
 logname: 164, 166
 logoutd: 121, 123
 logsave: 201, 202
 look: 196, 198
 lookbib: 172, 173
 losetup: 196, 198
 ls: 164, 166
 lsattr: 201, 202
 lsblk: 196, 198
 lscpu: 196, 198
 lsipc: 196, 198
 lsirq: 196, 198
 lslocks: 196, 198
 lslogins: 196, 198
 lsmem: 196, 198
 lsmod: 155, 155
 lsns: 196, 198
 lto-dump: 125, 128
 lzcat: 99, 99
 lzcmp: 99, 99
 lzdiff: 99, 99
 lzegrep: 99, 99
 lzfgrep: 99, 99
 lzgrep: 99, 99
 lzless: 99, 99
 lzma: 99, 99
 lzmadec: 99, 99
 lzmainfo: 99, 99
 lzmore: 99, 99
 m4: 104, 104
 make: 182, 182
 makedb: 89, 94
 makeinfo: 185, 185
 man: 191, 192
 man-recode: 191, 192
 mandb: 191, 192
 manpath: 191, 192
 mapscrn: 179, 180
 mcookie: 196, 198
 md5sum: 164, 166
 msg: 196, 198
 meson: 163, 163
 mkdir: 164, 166
 mke2fs: 201, 202
 mkfifo: 164, 166
 mkfs: 196, 198
 mkfs.bfs: 196, 198
 mkfs.cramfs: 196, 198
 mkfs.ext2: 201, 202
 mkfs.ext3: 201, 202
 mkfs.ext4: 201, 202
 mkfs.minix: 196, 198
 mklost+found: 201, 202
 mknod: 164, 166
 mkswap: 196, 198
 mktemp: 164, 166
 mk_cmds: 201, 202
 mmroff: 172, 173
 modinfo: 155, 155
 modprobe: 155, 155
 more: 196, 198
 mount: 196, 198
 mountpoint: 196, 199
 msgattrib: 135, 135
 msgcat: 135, 135
 msgcmp: 135, 135
 msgcomm: 135, 135
 msgconv: 135, 135
 msgen: 135, 135
 msgexec: 135, 135
 msgfilter: 135, 135
 msgfmt: 135, 135
 msggrep: 135, 136
 msginit: 135, 136
 msgmerge: 135, 136
 msgunfmt: 135, 136
 msguniq: 135, 136

mtrace: 89, 94
 mv: 164, 166
 namei: 196, 199
 ncursesw6-config: 130, 131
 neqn: 172, 173
 newgidmap: 121, 123
 newgrp: 121, 123
 newuidmap: 121, 123
 newusers: 121, 123
 ngettext: 135, 136
 nice: 164, 166
 ninja: 162, 162
 nl: 164, 166
 nm: 111, 112
 nohup: 164, 166
 nologin: 121, 123
 nproc: 164, 166
 nroff: 172, 173
 nscd: 89, 94
 nsenter: 196, 199
 nstat: 177, 178
 numfmt: 164, 166
 objcopy: 111, 112
 objdump: 111, 112
 od: 164, 166
 openssl: 153, 154
 openvt: 179, 180
 partx: 196, 199
 passwd: 121, 123
 paste: 164, 166
 patch: 183, 183
 pathchk: 164, 166
 pcprofiledump: 89, 94
 pdfmom: 172, 173
 pdfroff: 172, 173
 pdftexi2dvi: 185, 185
 peekfd: 134, 134
 perl: 147, 148
 perl5.36.0: 147, 148
 perlbug: 147, 148
 perldoc: 147, 148
 perlivp: 147, 148
 perlthanks: 147, 148
 pfbtops: 172, 173
 pgrep: 194, 194
 pic: 172, 173
 pic2graph: 172, 173
 piconv: 147, 148
 pidof: 194, 194
 ping: 144, 145
 ping6: 144, 145
 pinky: 164, 166
 pip3: 159
 pivot_root: 196, 199
 pkg-config: 129, 129
 pkill: 194, 194
 pl2pm: 147, 148
 pldd: 89, 94
 pmap: 194, 194
 pod2html: 147, 148
 pod2man: 147, 148
 pod2texi: 185, 186
 pod2text: 147, 148
 pod2usage: 147, 148
 podchecker: 147, 148
 podselect: 147, 148
 post-grohtml: 172, 173
 poweroff: 205, 205
 pr: 164, 166
 pre-grohtml: 172, 173
 preconv: 172, 173
 printenv: 164, 166
 printf: 164, 166
 prlimit: 196, 199
 prove: 147, 148
 prtstat: 134, 134
 ps: 194, 194
 psfaddtable: 179, 180
 psfgettable: 179, 180
 psfstriptime: 179, 180
 psfxtable: 179, 180
 pslog: 134, 134
 pstree: 134, 134
 pstree.x11: 134, 134
 ptar: 147, 148
 ptardiff: 147, 148
 ptargrep: 147, 148
 ptx: 164, 166
 pwait: 194, 194
 pwck: 121, 123
 pwconv: 121, 123
 pwd: 164, 166
 pwdx: 194, 194
 pwunconv: 121, 123
 pydoc3: 159
 python3: 159
 ranlib: 111, 112
 readelf: 111, 112
 readlink: 164, 167
 readprofile: 196, 199
 realpath: 164, 167
 reboot: 205, 205
 recode-sr-latin: 135, 136
 refer: 172, 173
 rename: 196, 199
 renice: 196, 199
 reset: 130, 131
 resize2fs: 201, 202
 resizepart: 196, 199
 rev: 196, 199
 rkfill: 196, 199
 rm: 164, 167
 rmdir: 164, 167
 rmmmod: 155, 155
 roff2dvi: 172, 173
 roff2html: 172, 173
 roff2pdf: 172, 173
 roff2ps: 172, 173
 roff2text: 172, 173
 roff2x: 172, 173
 routef: 177, 178
 routel: 177, 178

rtacct: 177, 178
 rtcwake: 196, 199
 rtmon: 177, 178
 rtpr: 177, 178
 rtstat: 177, 178
 runcon: 164, 167
 runlevel: 205, 205
 runtest: 110, 110
 rview: 187, 189
 rvim: 187, 189
 script: 196, 199
 scriptlive: 196, 199
 scriptreplay: 196, 199
 sdiff: 169, 169
 sed: 133, 133
 seq: 164, 167
 setarch: 196, 199
 setcap: 120, 120
 setfacl: 119, 119
 setfattr: 118, 118
 setfont: 179, 180
 setkeycodes: 179, 180
 settled: 179, 180
 setmetamode: 179, 180
 setsid: 196, 199
 setterm: 196, 199
 setvtrgb: 179, 180
 sfdisk: 196, 199
 sg: 121, 123
 sh: 139, 139
 shasum: 164, 167
 sha224sum: 164, 167
 sha256sum: 164, 167
 sha384sum: 164, 167
 sha512sum: 164, 167
 shasum: 147, 148
 showconsolefont: 179, 180
 showkey: 179, 180
 shred: 164, 167
 shuf: 164, 167
 shutdown: 205, 205
 size: 111, 112
 slabtop: 194, 194
 sleep: 164, 167
 sln: 89, 94
 soelim: 172, 173
 sort: 164, 167
 sotruss: 89, 94
 splain: 147, 148
 split: 164, 167
 sprof: 89, 94
 ss: 177, 178
 stat: 164, 167
 stdbuf: 164, 167
 strings: 111, 112
 strip: 111, 113
 stty: 164, 167
 su: 121, 123
 sulogin: 196, 199
 sum: 164, 167
 swapon: 196, 199
 swapoff: 196, 199
 swapon: 196, 199
 switch_root: 196, 199
 sync: 164, 167
 sysctl: 194, 194
 syslogd: 204, 204
 tabs: 130, 131
 tac: 164, 167
 tail: 164, 167
 talk: 144, 145
 tar: 184, 184
 taskset: 196, 199
 tbl: 172, 173
 tc: 177, 178
 tclsh: 107, 108
 tclsh8.6: 107, 108
 tee: 164, 167
 telinit: 205, 205
 telnet: 144, 145
 test: 164, 167
 texi2dvi: 185, 186
 texi2pdf: 185, 186
 texi2any: 185, 186
 texindex: 185, 186
 tfmtodit: 172, 173
 tftp: 144, 145
 tic: 130, 131
 timeout: 164, 167
 tload: 194, 194
 toe: 130, 132
 top: 194, 194
 touch: 164, 167
 tput: 130, 132
 tr: 164, 167
 traceroute: 144, 145
 troff: 172, 173
 true: 164, 167
 truncate: 164, 167
 tset: 130, 132
 tsort: 164, 167
 tty: 164, 167
 tune2fs: 201, 202
 tzselect: 89, 94
 uclampset: 196, 199
 udevadm: 190, 190
 udevd: 190, 190
 ul: 196, 199
 umount: 196, 199
 uname: 164, 167
 uname26: 196, 199
 uncompress: 176, 176
 unexpand: 164, 167
 unicode_start: 179, 180
 unicode_stop: 179, 180
 uniq: 164, 167
 unlink: 164, 167
 unlzma: 99, 99
 unshare: 196, 199
 unxz: 99, 99
 updatedb: 171, 171
 uptime: 194, 194

useradd: 121, 123
 userdel: 121, 123
 usermod: 121, 123
 users: 164, 167
 utmpdump: 196, 199
 uuid: 196, 199
 uuidgen: 196, 199
 uuidparse: 196, 199
 vdir: 164, 167
 vi: 187, 189
 view: 187, 189
 vigr: 121, 123
 vim: 187, 189
 vimdiff: 187, 189
 vimtutor: 187, 189
 vipw: 121, 124
 vmstat: 194, 194
 w: 194, 195
 wall: 196, 199
 watch: 194, 195
 wc: 164, 167
 wdctl: 196, 199
 whatis: 191, 192
 wheel: 161
 whereis: 196, 199
 who: 164, 167
 whoami: 164, 167
 wipefs: 196, 199
 x86_64: 196, 199
 xargs: 171, 171
 xgettext: 135, 136
 xmlwf: 143, 143
 xsubpp: 147, 148
 xtrace: 89, 94
 xxd: 187, 189
 xz: 99, 99
 xzcat: 99, 99
 xzcmp: 99, 99
 xzdec: 99, 99
 xzdiff: 99, 100
 xzegrep: 99, 100
 xzfgrep: 99, 100
 xzgrep: 99, 100
 xzless: 99, 100
 xzmore: 99, 100
 yacc: 137, 137
 yes: 164, 167
 zcat: 176, 176
 zcmp: 176, 176
 zdiff: 176, 176
 zdump: 89, 94
 zegrep: 176, 176
 zfgrep: 176, 176
 zforce: 176, 176
 zgrep: 176, 176
 zic: 89, 94
 zipdetails: 147, 148
 zless: 176, 176
 zmore: 176, 176
 znew: 176, 176
 zramctl: 196, 199

zstd: 101, 101
 zstdgrep: 101, 101
 zstdless: 101, 101

ライブラリ

Expat: 149, 149
 ld-2.36.so: 89, 94
 libacl: 119, 119
 libanl: 89, 94
 libasprintf: 135, 136
 libattr: 118, 118
 libbdf: 111, 113
 libblkid: 196, 200
 libBrokenLocale: 89, 94
 libbz2: 97, 98
 libc: 89, 94
 libcap: 120, 120
 libcheck: 168, 168
 libcom_err: 201, 202
 libcrypt: 89, 94
 libcrypto.so: 153, 154
 libctf: 111, 113
 libctf-nobfd: 111, 113
 libcursesw: 130, 132
 libc_malloc_debug: 89, 94
 libdl: 89, 94
 libe2p: 201, 203
 libelf: 157, 157
 libexpat: 143, 143
 libexpect-5.45.4: 109, 109
 libext2fs: 201, 203
 libfdisk: 196, 200
 libffi: 158
 libfl: 106, 106
 libformw: 130, 132
 libg: 89, 94
 libgcc: 125, 128
 libgcov: 125, 128
 libgdbm: 141, 141
 libgdbm_compat: 141, 141
 libgettextlib: 135, 136
 libgettextpo: 135, 136
 libgettextsrc: 135, 136
 libgmp: 114, 115
 libgmpxx: 114, 115
 libgomp: 125, 128
 libhistory: 103, 103
 libitm: 125, 128
 libkmod: 155
 liblsan: 125, 128
 libltdl: 140, 140
 liblto_plugin: 125, 128
 liblzma: 99, 100
 libm: 89, 94
 libmagic: 102, 102
 libman: 191, 193
 libmandb: 191, 193
 libmcheck: 89, 94
 libmemusage: 89, 94
 libmenuw: 130, 132

libmount: 196, 200
 libmpc: 117, 117
 libmpfr: 116, 116
 libmvec: 89, 94
 libncursesw: 130, 132
 libnsl: 89, 94
 libnss_*: 89, 94
 libopcodes: 111, 113
 libpanelw: 130, 132
 libpcprofile: 89, 94
 libpipeline: 181
 libproc-2: 194, 195
 libpsx: 120, 120
 libpthread: 89, 94
 libquadmath: 125, 128
 libreadline: 103, 103
 libresolv: 89, 95
 librt: 89, 95
 libsmartcols: 196, 200
 libss: 201, 203
 libssl.so: 153, 154
 libssp: 125, 128
 libstdbuf: 164, 167
 libstdc++: 125, 128
 libstdc++fs: 125, 128
 libsubid: 121, 124
 libsupc++: 125, 128
 libtcl8.6.so: 107, 108
 libtclstub8.6.a: 107, 108
 libtextstyle: 135, 136
 libthread_db: 89, 95
 libtsan: 125, 128
 libubsan: 125, 128
 libudev: 190, 190
 libutil: 89, 95
 libuuid: 196, 200
 liby: 137, 137
 libz: 96, 96
 libzstd: 101, 101
 preloadable_libintl: 135, 136

スクリプト

checkfs: 210, 210
 cleanfs: 210, 210
 console: 210, 210
 設定: 222
 console: 210, 210
 設定: 222
 File creation at boot
 設定: 224
 functions: 210, 210
 halt: 210, 210
 hostname
 設定: 218
 ifdown: 210, 210
 ifup: 210, 210
 ipv4-static: 210, 210
 localnet: 210, 210
 /etc/hosts: 218
 localnet: 210, 210

/etc/hosts: 218
 modules: 210, 210
 mountfs: 210, 210
 mountvirtfs: 210, 210
 network: 210, 210
 /etc/hosts: 218
 設定: 217
 network: 210, 210
 /etc/hosts: 218
 設定: 217
 network: 210, 210
 /etc/hosts: 218
 設定: 217
 rc: 210, 210
 reboot: 210, 210
 sendsignals: 210, 210
 setclock: 210, 210
 設定: 221
 setclock: 210, 210
 設定: 221
 swap: 210, 210
 sysctl: 210, 210
 syslogd: 210, 210
 設定: 224
 syslogd: 210, 210
 設定: 224
 template: 210, 211
 udev: 210, 211
 udev_retry: 210, 211
 dwp: 111, 112

その他

/boot/config-5.19.2: 233, 236
 /boot/System.map-5.19.2: 233, 237
 /dev/*: 69
 /etc/fstab: 231
 /etc/group: 71
 /etc/hosts: 218
 /etc/inittab: 220
 /etc/inputrc: 228
 /etc/ld.so.conf: 93
 /etc/lfs-release: 241
 /etc/localtime: 92
 /etc/lsb-release: 241
 /etc/modprobe.d/usb.conf: 236
 /etc/nsswitch.conf: 92
 /etc/os-release: 241
 /etc/passwd: 71
 /etc/profile: 227
 /etc/protocols: 88
 /etc/resolv.conf: 218
 /etc/services: 88
 /etc/syslog.conf: 204
 /etc/udev: 190, 190
 /etc/udev/hwdb.bin: 190
 /etc/vimrc: 188
 /run/utmp: 71
 /usr/include/asm-generic/*.h: 44, 44
 /usr/include/asm/*.h: 44, 44
 /usr/include/drm/*.h: 44, 44

/usr/include/linux/*.h: 44, 44
/usr/include/misc/*.h: 44, 44
/usr/include/mtd/*.h: 44, 44
/usr/include/rdma/*.h: 44, 44
/usr/include/scsi/*.h: 44, 44
/usr/include/sound/*.h: 44, 44
/usr/include/video/*.h: 44, 44
/usr/include/xen/*.h: 44, 44
/var/log/btmp: 71
/var/log/lastlog: 71
/var/log/wtmp: 71
/etc/shells: 229
man ページ: 87, 87