Linux From Scratch

Version r12.4-29-systemd

2025/10/01 公開

製作： Gerard Beekmans
編集総括： Bruce Dubbs
編集： Douglas R. Reno
編集： DJ Lucas
日本語訳： 松山 道夫

Linux From Scratch: Version r12.4-29-systemd: 2025/10/01 公開

: 製作： Gerard Beekmans, 編集総括： Bruce Dubbs, 編集： Douglas R. Reno, 編集： DJ Lucas, 、 日本語訳： 松山 道夫

目次

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ったら、いくらでも変えたら良いし、そうすることがむしろ当たり前となったのです。

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がありませんでした。個々のディストリビューションは優れたもので、それぞれを見てみれば正しいものです。ただこれは正しいとか間違っているとかの問題ではなく、個人的な趣味の問題へと変化しています。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、逆にすべての強力さを合わせ持つものです。当初は気の遠くなる思いがしていましたが、そのアイデアは今も持ち続けています。

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

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

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ちます。そしてあなたが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、たいていのものは使わないですし、その内容もよく分からないものです。それらのプログラムはハードウェアリソースを無駄に占有することになります。今日のハードドライブや CPU のことを考えたら、リソース消費は大したことはないと思うかもしれません。しかし問題がなくなったとしても、サイズの制限だけは気にかける必要があることでしょう。例えばブータブル CD、USB スティック、組み込みシステムなどのことを思い浮かべてください。そういったものに対して LFS は有用なものとなるでしょう。

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

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

LFS を構築する理由はさまざまですから、すべてを列記することはできません。学習こそ、理由を突き詰める重要な手段です。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。）

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% 早くなるだけです（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 を用いるのでも充分です。一方 BLFS にあるパッケージの中には、ビルド時や実行時に 4 GB 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 ビットシステムを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 ビットシステムです。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 の教育的で最小限のものとする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、ひょっとすると役に立たないかもしれません。 フォーラムにおいて基本的な知識を尋ねたとしたら、誰も回答してくれないでしょう。（そうするよりも LFS に取り掛かる前に以下のような情報をよく読んでください。）

LFS システムの構築作業に入る前に、以下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）https://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) https://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つの仕様から構成されます。 コア (Core)、デスクトップ (Desktop)、言語 (Languages)、画像処理 (Imaging) です。 コアとデスクトップの中には、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しの仕様も含んでいます。 LFS では前節にて示したように、IA32 (32 ビット x86) や AMD64 (x86_64) アーキテクチャーに対応する LSB 仕様への適合を目指しています。

注記

このような要求に対しては異論のある方も多いでしょう。 LSB の目的は、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「一から」クリアしていくように構築することも可能です。 ただし LFS ブック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, GCC, Gettext, Glibc, Grep, Gzip, M4, Man-DB, Procps, Psmisc, Sed, Shadow, Systemd, Tar, Util-linux, Zlib
LSB デスクトップ:	なし
LSB 言語:	Perl
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), BLFS Bash Startup Files, Cpio, Ed, Fcrontab, 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, GLU, Icon-naming-utils, Libjpeg-turbo, Libxml2, Mesa, Pango, Xdg-utils, Xorg
LSB 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、あるいはオプションで提供されているコンポーネントで LSB 要求に従うもの

LSB コア:	install_initd, **libcrypt.so.1** (LFS の Libxcrypt パッケージにおける任意実施の手順により提供), **libncurses.so.5** (LFS の Ncurses パッケージにおける任意実施の手順により提供), **libncursesw.so.5** (ただし **libncursesw.so.6** は LFS の Ncurses パッケージにより提供)

LSB デスクトップ:	libgdk-x11-2.0.so（ただし libgdk-3.so は BLFS GTK+-3 パッケージが提供），libgtk-x11-2.0.so（ただし libgtk-3.so と libgtk-4.so はBLFS GTK+-3 と GTK-4 パッケージが提供），libpng12.so（ただし libpng16.so は BLFS Libpng パッケージが提供），libQt*.so.4（ただし libQt6*.so.6 は BLFS Qt6 パッケージが提供），libtiff.so.4（ただし libtiff.so.6 は BLFS Libtiff パッケージが提供）
LSB 言語:	/usr/bin/python（LSB は Python2 を要求しているが LFS と BLFS は Python3 のみを提供）
LSB 画像処理:	なし
LSB Gtk3、LSB グラフィックス (試用):	libpng15.so（ただし libpng16.so は BLFS Libpng パッケージが提供）

各パッケージを用いる理由

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、きめ細かくさまざま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て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 プログラムをビルドする際に、これを必要とするものがあります。

- Bzip2

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

- Coreutils

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

- D-Bus

  このパッケージはメッセージバスシステムを実装しています。 これはアプリケーション間での通信手段を容易にするものです。

- 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。

- Expat

  このパッケージは比較的小規模の XML 解析ライブラリを生成します。 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 コンパイラーコレクションパッケージです。 C コンパイラーと C++ コンパイラーを含みます。また LFS ではビルドしないコンパイラーも含まれています。

- 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。 Systemd 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。

- Jinja2

  このパッケージは、テキストテンプレート処理を行う Python モジュールです。 Systemd のビルドに必要となります。

- 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生成します。 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。 多くのパッケージでは、ページング出力を行うためにこれを利用しています。

- 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ー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。

- Libxcrypt

  このパッケージは **libcrypt** ライブラリを提供するものであり、さまざまなパッケージ（代表的なものとして Shadow）がパスワードのハッシュ処理のために必要としています。 これは Glibc における、かつての **libcrypt** 実装を置き換えるものです。

- 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ーを提供するものであり、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。

- MarkupSafe

  このパッケージは、HTML/XHTML/XML 内の文字列を安全に処理するための Python モジュールです。 Jinja2 がこのパッケージを必要としています。

- Man-DB

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

- Man-pages

  このパッケージは Linux の基本的な man ページを提供します。

- 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エ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ける時間を、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。

- Pcre2

This package provides a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5.

- 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。

- Pkgconf

  このパッケージは、開発用ライブラリに対するコンパイラーフラグやリンカーフラグを設定するためのプログラムを提供します。 このプログラムは、pkg-config の単純な置き換えプログラムとして利用することができます。 そもそもこのプログラムは、数多くのパッケージによるシステム構築に必要となるものです。 元々の Pkg-config パッケージに比べて活発に開発されており、処理速度も若干早くなっています。

- 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、多くのパッケージが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。

- Sqlite

  このパッケージは、サーバーレスで設定不要のトランザクション型 SQL データベースエンジンを提供します。

- Systemd

  このパッケージは SysVinit の代替として、init プログラムなど数種のプログラムにより、システム起動やシステム制御を実現します。 多くの Linux ディストリビューションにおいてもよく利用されています。

- 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。 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。

- 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。

- 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。

- 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。 本書を（PDF ではなく）HTML 形式で読んでいる場合、そのテキストは青字で表示されているはずです。

固定幅フォントは `/etc/ld.so.conf` のようなファイル名を示す際にも用います。

注記

ブラウザーの設定において、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、`Il1` 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** が表示されることになります。 特定のマニュアルページを見たい場合は man 5 passwd といった入力を行う必要があります。 マニュアルページが複数あるケースはまれですので、普通は man <プログラム名> と入力するだけで十分です。 LFS ブックにおけるこの表記はハイパーリンクとしています。 その表記をクリックすると Arch Linux man ページ が提供する man ページを開きます。

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

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/systemd/ 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、このリストを読み、https://www.linuxfromscratch.org/lfs/advisories/ にアクセスしてください。 LFS のビルド作業を進めていく上では、各セクションに対するセキュリティアドバイスの内容に従って、修正作業を適用してください。 さらに LFS システムを、現実にデスクトップやサーバーシステムとして利用としている場合は、アドバイスを常に確認してセキュリティフィックスを適用するようにしてください。 これは LFS システムを構築した後であっても同様です。

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 r12.4-29-systemd の日本語版 20251002 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 GitHub 内に開発の場を設け https://lfsbookja.github.io/lfsbookja-doc/ja.index.html 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、処理に必要となるツール類の詳細が示されています。 それらのツール類はすべて BLFS にてインストールする対象となっていますので、興味のある方は参照してください。
- 日本語訳にあたっては、原文にて「地の文」として表現されている文章を日本語化しています。 逆に各手順におけるコマンド説明（四角の枠囲いで示されている箇所）は、日本語化の対象とはしていません。 コマンド類や設定記述が英単語で行われるわけですから、これは当たり前のことです。 ただ厳密に言えば、その四角の枠囲いの中でシェルのコメント書きが含まれる場合があり、これは日本語化せずそのまま表記しています。

日本語版における注意点

日本語版 LFS ブックを参照頂く際には、以下の点に注意してください。

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

第I部 はじめに

第1章 はじめに

1.1. LFS をどうやって作るか

LFS システムは、既にインストールされている Linux ディストリビューション (Debian、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ー、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を含めておく必要があります。

注記

Linux ディストリビューションのインストールには、さまざまな方法がありますが、デフォルトインストールでは、普通は LFS システムの構築には適していません。 商用ディストリビューションにおける設定方法に関しては https://www.linuxfromscratch.org/hints/downloads/files/partitioning-for-lfs.txt を参照してください。

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 構築に必要となるパッケージとパッチについて説明します。 これらをダウンロードして新たなファイルシステム内に保存します。 第 4 章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。 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 章において本格的な 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、いつも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生成します。 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。 本章に示す作業をすべて実施すれば、新たな LFS システムを起動することが出来ます。

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

1.2. 前版からの変更点

以下に示すのは、前版から変更されているパッケージです。

アップグレード:

-
- Coreutils-9.8
- Expat-2.7.3
- Iana-Etc-20250926
- Kbd-2.9.0
- Linux-6.16.9
- MarkupSafe-3.0.3

- Meson-1.9.1
- OpenSSL-3.5.4
- Pcre2-10.46
- Systemd-258
- Tcl-8.6.17
- Util-linux-2.41.2
- Vim-9.1.1806

追加:

-
- Coreutils-9.8-i18n-2.patch

削除:

-
- Coreutils-9.7-i18n-1.patch
- Coreutils-9.7-upstream_fix-1.patch

1.3. 変更履歴

本書は Linux From Scratch ブック、バージョン r12.4-29-systemd、2025/10/01 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す。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

- 2025-10-01
  - [renodr] - systemd-258 (SysV 向け udev を含む) へのアップデート。 #5791 を Fix に。
  - [bdubbs] - vim-9.1.1806 へのアップデート。 #4500 にて言及。
  - [bdubbs] - iana-etc-20250926 へのアップデート。 #5006 にて言及。
  - [bdubbs] - coreutils-9.8 へのアップデート。 #5795 を Fix に。
  - [bdubbs] - expat-2.7.3 (セキュリティリリース) へのアップデート。 #5792 を Fix に。
  - [bdubbs] - linux-6.16.9 へのアップデート。 #5796 を Fix に。
  - [bdubbs] - markupsafe-3.0.3 へのアップデート。 #5801 を Fix に。
  - [bdubbs] - meson-1.9.1 へのアップデート。 #5797 を Fix に。
  - [bdubbs] - openssl-3.5.4 (セキュリティアップデート) へのアップデート。 #5793 を Fix に。
  - [bdubbs] - util-linux-2.41.2 へのアップデート。 #5798 を Fix に。
- 2025-09-15
  - [bdubbs] - vim-9.1.1754 へのアップデート。 #4500 にて言及。
  - [bdubbs] - iana-etc-20250826 へのアップデート。 #5006 にて言及。
  - [bdubbs] - tcl8.6.17 へのアップデート。 #5781 を Fix に。
  - [bdubbs] - pcre2-10.46 へのアップデート。 #5790 を Fix に。
  - [bdubbs] - meson-1.9.0 へのアップデート。 #5788 を Fix に。
  - [bdubbs] - linux-6.16.7 へのアップデート。 #5787 を Fix に。
  - [bdubbs] - kbd-2.9.0 へのアップデート。 #5789 を Fix に。
- 2025-09-03
  - [bdubbs] - sqlite-3.50.4 パッケージの追加。 #5784 を Fix に。
  - [bdubbs] - pcre2-10.45 パッケージの追加。 #5782 を Fix に。
  - [bdubbs] - 第 3 章にて、カーネルの base+patch の利用方法の説明を追加。 #5785 を Fix に。
- 2025-09-01

- [bdubbs] - LFS-12.4 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 r12.4-29-systemd 日本語版（バージョン20251002）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。

「r12.4-XXX」という表記は、オリジナル LFS ブック GIT 管理ソースの連番号を意味します。 また a2bf74a9b などの表記は、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を意味します。

変更履歴

- 2025-10-02
  - [matsuand] - r12.4-29 (5abealab6) までの対応。
- 2025-10-01
  - [matsuand] - r12.4-28 (454448912) までの対応。
- 2025-09-21
  - [matsuand] - r12.4-22 (215e914e5) までの対応。
- 2025-09-16
  - [matsuand] - r12.4-20 (158d47f83) までの対応。
- 2025-09-02
  - [matsuand] - r12.4-15 (09d0e04da) までの対応。
- 2025-09-01
  - [matsuand] - LFS 12.4 対応。

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 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 ページに答えが見つからなかった場合には、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 において、チャネル名 #lfs-support により提供しています。

1.5.4. ミラーサイト

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

注記

LFS の手順に従って特定のパッケージをビルドした際に、何かの問題が発生した場合、いきなりアップストリームのサポートチャンネルへ問題を投稿することは是非お止めください。 その前にまずは 「情報源」 に示されている LFS サポートチャンネルでの議論を行ってください。 いきなりアップストリームの保守担当者に投稿したところで、その担当者は LFS のビルド手順についてほぼ理解はしていないため、非効率なやり方となります。 たとえアップストリームの問題であったとしても、LFS コミュニティを経由すれば、アップストリームが本当に必要とする情報のみを抜き出して適切な報告を上げるお役に立てるはずです。

アップストリームのサポートチャンネルに直接質問を上げることがどうしても必要となった場合でも、多くのアップストリームプロジェクトにおいては、サポートチャンネルとバグトラッカーは別々に運用されている点に注意してください。 「バグ」報告に質問を行うことは不適切とされて、そのアップストリームプロジェクトの開発担当者に迷惑をかけるだけかもしれません。

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。 そこに回答が示されていなかったなら、問題の本質部分を見極めてください。 トラブルシューティングとして以下のヒントが有用かもしれません。 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、簡単な状況説明に加えて、尋ねたい内容に合わせて以下の基本的情報も含めてください。

- お使いの LFS ブックのバージョン。（本書の場合 r12.4-29-systemd）
- 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問題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、関連する情報はすべて含めてください。 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 -D ALIASPATH=\"/mnt/lfs/usr/share/locale:.\"
-D LOCALEDIR=\"/mnt/lfs/usr/share/locale\"
-D LIBDIR=\"/mnt/lfs/usr/lib\"
-D INCLUDEDIR=\"/mnt/lfs/usr/include\" -D HAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

たいていの方は、上のような場合に終わりの数行しか示してくれません。

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

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

2.2.1. ハードウェア

LFS 編集者としては、システム CPU は最低でも 4 コア、メモリ容量は最低でも 8 GB を推奨しています。 この要件を満たさない古いシステムであっても、動くかもしれません。 しかしパッケージのビルド時間は、本書に示すものよりも極端に長くなるかもしれません。

2.2.2. ソフトウェア

ホストシステムには以下に示すソフトウェアが必要であり、それぞれに示されているバージョン以降である必要があります。 最近の Linux ディストリビューションを利用するなら、あまり問題にはならないはずです。 ディストリビューションによっては、ソフトウェアのヘッダ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 (2.45 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-8.1
- 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-5.4 と C++ コンパイラーである g++ (15.2.0 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-5.4

  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す。 こうすると古いカーネルに対する対応コードが無効となり、コンパイルした glibc が若干早く、また軽量になります。 2024 年 12 月時点、カーネル開発者によってサポートされる、もっとも古いカーネルバージョンは 5.4 です。 5.4 よりも古いカーネルリリースであっても、サードパーティチームによってサポートされているものもあります。 ただしそういったものは、公式のカーネルリリースとは認められません。 詳しくは https://kernel.org/category/releases.html を参照してください。

  ホストシステムのカーネルバージョンが 5.4 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 5.4 以上のバージョンのカーネルを提供しているかを調べることです。 提供していれば、それをインストールします。 もしそれがない場合や、あったとしてもそれをインストールしたくない場合、カーネルをご自身でコンパイルする必要があります。 カーネルのコンパイルと (ホストシステムが GRUB を利用しているとして) ブートローダーの設定方法については第 10 章 を参照してください。

  本書では、ホストカーネルが UNIX 98 疑似端末 (PTY) をサポートしていることが必要です。 これは Linux 5.4 またはそれ以降のカーネルを利用するデスクトップ向け、あるいはサーバー向けのディストリビューションにとって利用できなければなりません。 独自のホストカーネルを利用している場合には、カーネル設定において CONFIG_UNIX98_PTYS が y であることを確認してください。
- 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-5.0
- 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。 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、以下のコマンドを実行して確認することができます。

```
cat > version-check.sh << "EOF"
#!/bin/bash
# A script to list version numbers of critical development tools

# If you have tools installed in other directories, adjust PATH here AND
# in ~lfs/.bashrc (section 4.4) as well.

LC_ALL=C
PATH=/usr/bin:/bin

bail() { echo "FATAL: $1"; exit 1; }
grep --version > /dev/null 2> /dev/null || bail "grep does not work"
sed '' /dev/null || bail "sed does not work"
sort   /dev/null || bail "sort does not work"

ver_check()
{
   if ! type -p $2 &>/dev/null
   then
     echo "ERROR: Cannot find $2 ($1)"; return 1;
   fi
   v=$($2 --version 2>&1 | grep -E -o '[0-9]+\.[0-9\.]+[a-z]*' | head -n1)
   if printf '%s\n' $3 $v | sort --version-sort --check &>/dev/null
   then
     printf "OK:    %-9s %-6s >= $3\n" "$1" "$v"; return 0;
   else
     printf "ERROR: %-9s is TOO OLD ($3 or later required)\n" "$1";
     return 1;
   fi
}

ver_kernel()
{
   kver=$(uname -r | grep -E -o '^[0-9\.]+')
   if printf '%s\n' $1 $kver | sort --version-sort --check &>/dev/null
   then
     printf "OK:    Linux Kernel $kver >= $1\n"; return 0;
   else
     printf "ERROR: Linux Kernel ($kver) is TOO OLD ($1 or later required)\n" "$kver";
     return 1;
   fi
}

# Coreutils first because --version-sort needs Coreutils >= 7.0
ver_check Coreutils      sort     8.1 || bail "Coreutils too old, stop"
```

```
ver_check Bash           bash     3.2
ver_check Binutils       ld       2.13.1
ver_check Bison          bison    2.7
ver_check Diffutils      diff     2.8.1
ver_check Findutils      find     4.2.31
ver_check Gawk           gawk     4.0.1
ver_check GCC            gcc      5.4
ver_check "GCC (C++)"    g++      5.4
ver_check Grep           grep     2.5.1a
ver_check Gzip           gzip     1.3.12
ver_check M4             m4       1.4.10
ver_check Make           make     4.0
ver_check Patch          patch    2.5.4
ver_check Perl           perl     5.8.8
ver_check Python         python3  3.4
ver_check Sed            sed      4.1.5
ver_check Tar            tar      1.22
ver_check Texinfo        texi2any 5.0
ver_check Xz             xz       5.0.0
ver_kernel 5.4

if mount | grep -q 'devpts on /dev/pts' && [ -e /dev/ptmx ]
then echo "OK:    Linux Kernel supports UNIX 98 PTY";
else echo "ERROR: Linux Kernel does NOT support UNIX 98 PTY"; fi

alias_check() {
   if $1 --version 2>&1 | grep -qi $2
   then printf "OK:    %-4s is $2\n" "$1";
   else printf "ERROR: %-4s is NOT $2\n" "$1"; fi
}
echo "Aliases:"
alias_check awk GNU
alias_check yacc Bison
alias_check sh Bash

echo "Compiler check:"
if printf "int main(){}" | g++ -x c++ -
then echo "OK:    g++ works";
else echo "ERROR: g++ does NOT work"; fi
rm -f a.out

if [ "$(nproc)" = "" ]; then
   echo "ERROR: nproc is not available or it produces empty output"
else
   echo "OK: nproc reports $(nproc) logical cores are available"
fi
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。 つまり作業途中に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コマンド実行します。 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。 これを行わなかった場合、パッケージインストールがホストに対して行われてしまい、利用不能になってしまうリスクがあります。
- 全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。
- 「仮想カーネルファイルシステムの準備」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。 chroot 環境に入った後は、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ので、スワップ（swap）領域をパーティションとして設けるのが普通です。 この領域へは利用頻度が低いデータを移すことで、アクティブな処理プロセスがより多くのメモリを確保できるようにカーネルが制御します。 swap パーティションは、LFS システムのものとホストシステムのものを共有することもできます。 その場合は新しいパーティションを作る必要はありません。

ディスクのパーティション生成は cfdisk コマンドや fdisk コマンドを使って行います。 コマンドラインオプションにはパーティションを生成するハードディスク名を指定します。 例えば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て不適当なファイル配置を生み出すため、余計なディスク消費を発生させます。

2.4.1.1. 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っても妥当なところでしょう。 それだけあれば LFS 構築も、また BLFS においてもおそらく十分なはずです。 実験的に複数パーティションを設けるとしても、これだけのサイズで十分です。

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 のハイバーネーション（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 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 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ことが必要です。 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 コマンドを用いた場合は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ことだけです。

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。 それでもこれを生成する場合、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、KDE や Texlive 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 - systemd はデフォルトで tmpfs をマウントします。 この動作を上書きしたい場合は 「/tmp の tmpfs としての生成抑止」 に従って LFS システムを設定してください。
- /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

パーティションとは、ディスクドライブ上の一定数のセクターの集まりのことです。 これはパーティションテーブルにおいて、その境界設定によって定められます。 オペレーティングシステムがファイルを保存するパーティションを利用できるように、そのパーティションはフォーマットしておかなければなりません。 そこにはラベル、ディレクトリブロック、データブロック、目的となるファイル位置へのインデックススキームといったものが含まれます。 ファイルシステムは、OS がパーティションの空き容量を管理できるようにしています。 また新規ファイル生成時や既存ファイルの拡張時に必要となるセクターの確保や、ファイル削除によって生み出された未使用データセグメントの再利用なども可能にします。 さらにデータ冗長性やエラー回復のためのサポート機能も提供しています。

LFS では Linux カーネルが認識できるファイルシステムであれば何でも利用できます。 最も標準的なものは ext3 や ext4 です。 ファイルシステムを正しく選ぶことは、実は難しいことです。 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
: ファイルシステムに用いられている 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 まで）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, 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、さらに多くのことは https://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 と umask 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。 他の名前にしても構いません。 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
```

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

ファイルモード生成マスク（umask）を 022 に設定します。 ホストディストロがこれとは違う値をデフォルトとしている場合を考慮して行うものです。

```
umask 022
```

umask を 022 に設定しておけば、新たに生成するファイルやディレクトリは、所有者のみが書き込み可能となり、他ユーザーは読み込みおよび（ディレクトリについての）検索についてのみ可能となります。（これはシステムコール open(2) がこのデフォルトモードを利用する前提としており、新規ファイルのモードは 644、新規ディレクトリは 755 となります。）LFS システムにおいて、デフォルト値を緩めすぎればセキュリティモードを生み出すことになり、逆に厳しすぎれば、ビルドや LFS システム利用自体に不都合な問題を引き起こすことにもなります。

注意

$LFS がセットされていて、umask は 022 に設定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。

umask が正しく設定されているかどうかを以下により確認してください。

```
umask
```

出力結果は 022 または 0022 となっているはずです。（先頭にゼロがつくかどうかは、ホストディストロによります。）

上の 2 つのコマンドの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、umask には 022 を設定してください。

注記

LFS 変数と umask を確実に設定しておくために、ローカルの .bash_profile および /root/.bash_profile に上記変数を export コマンド、mask 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。 .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 に上記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、”if”テストを利用して残りの .bashrc を実行しないようにしているものがあります。 非対話形式を利用する場合は、そのテストの直前に上記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、ホストシステムからアクセスできるようにパーティションをマウントします。 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をマウントするものとします。

厳密に言うと「パーティションはマウントできません」。 マウントできるのは、そのパーティション内に埋め込まれているファイルシステムです。 ただし 1 つのパーティションに複数のファイルシステムを収めることはできないので、パーティションとそこに関連づいたファイルシステムのことを、同一のものとして表現するわけです。

以下のコマンドによって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。

（LFS システムにおいて新たに生成されるファイルシステムのルートディレクトリである）**$LFS** ディレクトリに対して、その所有者とパーミッションモードをそれぞれ **root** および **755** に設定します。 ホストディストロがこれとは違う値をデフォルトとしている場合を考慮して行うものです。

```
chown root:root $LFS
chmod 755 $LFS
```

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。 例えば **/etc/fstab** ファイルに以下のような行を追加します。

```
/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。

こうして新たな LFS パーティションが整いました。 次はパッケージのダウンロードです。

第3章 パッケージとパッチ

3.1. はじめに

この章では基本的な Linux システム構築のためにダウンロードするべきパッケージの一覧を示します。 各パッケージのバージョンは動作が確認されているものを示しており、本書ではこれに基づいて説明します。 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、両者を同様のファイル名、場合によっては完全に一致したファイル名で提供している場合があります。 ただしリリース tarball の中には、リポジトリスナップショットの内容に加えて、リポジトリには保存されていない重要なファイル（たとえば autoconf によって生成される configure スクリプトなど）を含む場合があります。 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（https://www.google.com/）を使って検索してみてください。 たいていのパッケージを見つけ出すことが出来るはずです。 それでも見つけられなかったら https://www.linuxfromscratch.org/lfs/packages.html#packages から入手してください。

重要

次のページでは ftp.gnu.org に存在する重要なパッケージをいくつも一覧に示しています。 このサイトは対象パッケージの標準的な配布サイトであるため、長期にわたって DDOS（distributed denial of services）攻撃にさらされています。 詳しくは slashdot ニュース を参照してください。

ftp.gnu.org に対しての別の選択として、そのミラーサイトを利用する方法があります。 ミラーサイトの一覧は https://www.gnu.org/prep/ftp.en.html に示されています。

以降に示している wget リストを利用する予定であれば、このファイルを修正して必要なミラーサイトを利用するようにしてください。

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-systemd 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-systemd 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-systemd --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-systemd 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-systemd 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-systemd 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|www\.kernel\.org/pub/linux/|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-systemd
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-systemd と wget-list-systemd.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-systemd --continue --directory-prefix=$LFS/sources
wget -N --input-file=wget-list-systemd.orig --continue --directory-prefix=$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。

パッケージとパッチを非 root ユーザーによってダウンロードした場合、各ファイルはそのユーザーが所有します。 ファイルシステムは、UID によって所有者を記録しますが、ホスト上の一般ユーザーの UID は LFS 内には割り当てられていません。 したがって各ファイルは、最終の LFS システムにおいて、名前付けられていない UID によって所有されたまま残ります。 LFS システムに存在する自身のユーザーに対して、同じ UID を割り当てるつもりがないのであれば、各ファイルの所有者を root に変更することで、この状況を解消してください。

```
chown root:root $LFS/sources/*
```

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、古いリリースソ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、ぜい弱性のあるパッケージを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 (2.3.2) - 363 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-2.3.2.tar.xz
MD5 sum: 590765dee95907dbc3c856f7255bd669

• Attr (2.5.2) - 484 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-2.5.2.tar.gz
MD5 sum: 227043ec2f6ca03c0948df5517f9c927

• Autoconf (2.72) - 1,360 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-2.72.tar.xz
MD5 sum: 1be79f7106ab6767f18391c5e22be701

• Automake (1.18.1) - 1,614 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-1.18.1.tar.xz
MD5 sum: cea31dbf1120f890cbf2a3032cfb9a68

• Bash (5.3) - 11,089 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-5.3.tar.gz
MD5 sum: 977c8c0c5ae6309191e7768e28ebc951

• Bc (7.0.3) - 464 KB:
ホームページ: https://github.com/gavinhoward
ダウンロード: https://github.com/gavinhoward/bc/releases/download/7.0.3/bc-7.0.3.tar.xz
MD5 sum: ad4db5a0eb4fdbb3f6813be4b6b3da74

• Binutils (2.45) - 27,216 KB:
ホームページ: https://www.gnu.org/software/binutils/
ダウンロード: https://sourceware.org/pub/binutils/releases/binutils-2.45.tar.xz
MD5 sum: dee5b4267e0305a99a3c9d6131f45759

• 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

• Coreutils (9.8) - 6,089 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-9.8.tar.xz
MD5 sum: b2e687b6e664b9dd76581836c5c3e782

• D-Bus (1.16.2) - 1,090 KB:
ホームページ: https://www.freedesktop.org/wiki/Software/dbus
ダウンロード: https://dbus.freedesktop.org/releases/dbus/dbus-1.16.2.tar.xz
MD5 sum: 97832e6f0a260936d28536e5349c22e5

• 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 (3.12) - 1,894 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-3.12.tar.xz
MD5 sum: d1b18b20868fb561f77861cd90b05de4

• E2fsprogs (1.47.3) - 9,851 KB:
ホームページ: https://e2fsprogs.sourceforge.net/
ダウンロード: https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.47.3/e2fsprogs-1.47.3.tar.gz
MD5 sum: 113d7a7ee0710d2a670a44692a35fd2e

• Elfutils (0.193) - 11,695 KB:
ホームページ: https://sourceware.org/elfutils/
ダウンロード: https://sourceware.org/ftp/elfutils/0.193/elfutils-0.193.tar.bz2
MD5 sum: ceefa052ded950a4c523688799193a44

• Expat (2.7.3) - 493 KB:
ホームページ: https://libexpat.github.io/
ダウンロード: https://github.com/libexpat/libexpat/releases/download/R_2_7_3/expat-2.7.3.tar.xz
MD5 sum: 423975a2a775ff32f12c53635b463a91

• 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 (5.46) - 1,283 KB:
ホームページ: https://www.darwinsys.com/file/
ダウンロード: https://astron.com/pub/file/file-5.46.tar.gz
MD5 sum: 459da2d4b534801e2e2861611d823864

• Findutils (4.10.0) - 2,189 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-4.10.0.tar.xz
MD5 sum: 870cfd71c07d37ebe56f9f4aaf4ad872

• 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

• Flit-core (3.12.0) - 53 KB:
ホームページ: https://pypi.org/project/flit-core/
ダウンロード: https://pypi.org/packages/source/f/flit-core/flit_core-3.12.0.tar.gz
MD5 sum: c538415c1f27bd69cbbbf3cdd5135d39

• Gawk (5.3.2) - 3,662 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-5.3.2.tar.xz
MD5 sum: b7014650c5f45e5d4837c31209dc0037

• GCC (15.2.0) - 98,688 KB:
ホームページ: https://gcc.gnu.org/
ダウンロード: https://ftp.gnu.org/gnu/gcc/gcc-15.2.0/gcc-15.2.0.tar.xz
MD5 sum: b861b092bf1af683c46a8aa2e689a6fd

• GDBM (1.26) - 1,198 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-1.26.tar.gz
MD5 sum: aaa600665bc89e2febb3c7bd90679115

• Gettext (0.26) - 9,926 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-0.26.tar.xz
MD5 sum: 8e14e926f088e292f5f2bce95b81d10e

• Glibc (2.42) - 19,464 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-2.42.tar.xz
MD5 sum: 23c6f5a27932b435cae94e087cb8b1f5

注記

Glibc の開発者は Git ブランチ を管理しており、そこには Glibc-2.42 に有用と思われるパッチを含んでいますが、それは残念ながら Glibc-2.42 のリリース以降に開発されたものに限ります。 LFS 編集者は、そのブランチにセキュリティフィックスが加えられた際には、セキュリティアドバイザリーを発表することにしています。 ただしセキュリティ以外で新規追加されたパッチに関しては、何も作業は行いません。 したがって各パッチは自分で確認するようにし、また重要であると思われる場合は各自でそのパッチを適用してください。

• GMP (6.3.0) - 2,046 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-6.3.0.tar.xz
MD5 sum: 956dc04e864001a9c22429f761f2c283

• Gperf (3.3) - 1,789 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-3.3.tar.gz
MD5 sum: 31753b021ea78a21f154bf9eecb8b079

• Grep (3.12) - 1,874 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-3.12.tar.xz
MD5 sum: 5d9301ed9d209c4a88c8d3a6fd08b9ac

• Groff (1.23.0) - 7,259 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-1.23.0.tar.gz
MD5 sum: 5e4f40315a22bb8a158748e7d5094c7d

• GRUB (2.12) - 6,524 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-2.12.tar.xz
MD5 sum: 60c564b1bdc39d8e43b3aab4bc0fb140

• Gzip (1.14) - 865 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-1.14.tar.xz
MD5 sum: 4bf5a10f287501ee8e8ebe00ef62b2c2

• Iana-Etc (20250926) - 593 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/20250926/iana-etc-20250926.tar.gz
MD5 sum: 437a3e9f4a420244c90db4ab20e713b6

• Inetutils (2.6) - 1,724 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-2.6.tar.xz
MD5 sum: 401d7d07682a193960bcdecafd03de94

• 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 (6.16.0) - 910 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/iproute2-6.16.0.tar.xz
MD5 sum: 80e1f91bf59d572acc15d5c6eb4f3e7c

• Jinja2 (3.1.6) - 240 KB:
ホームページ: https://jinja.palletsprojects.com/en/3.1.x/
ダウンロード: https://pypi.org/packages/source/J/Jinja2/jinja2-3.1.6.tar.gz
MD5 sum: 66d4c25ff43d1deaf9637ccda523dec8

• Kbd (2.9.0) - 1,492 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-2.9.0.tar.xz
MD5 sum: 7be7c6f658f5fb9512e2c490349a8eeb

• Kmod (34.2) - 434 KB:
ホームページ: https://github.com/kmod-project/kmod
ダウンロード: https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-34.2.tar.xz
MD5 sum: 36f2cc483745e81ede3406fa55e1065a

• Less (679) - 857 KB:
ホームページ: https://www.greenwoodsoftware.com/less/
ダウンロード: https://www.greenwoodsoftware.com/less/less-679.tar.gz
MD5 sum: 0386dc14f6a081a94dfb4c2413864eed

• Libcap (2.76) - 195 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-2.76.tar.xz
MD5 sum: 449ade7d620b5c4eeb15a632fbaa4f74

• Libffi (3.5.2) - 1,390 KB:
ホームページ: https://sourceware.org/libffi/
ダウンロード: https://github.com/libffi/libffi/releases/download/v3.5.2/libffi-3.5.2.tar.gz
MD5 sum: 92af9efad4ba398995abf44835c5d9e9

• Libpipeline (1.5.8) - 1,046 KB:
ホームページ: https://libpipeline.nongnu.org/
ダウンロード: https://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.8.tar.gz
MD5 sum: 17ac6969b2015386bcb5d278a08a40b5

• Libtool (2.5.4) - 1,033 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-2.5.4.tar.xz
MD5 sum: 22e0a29df8af5fdde276ea3a7d351d30

• Libxcrypt (4.4.38) - 612 KB:
ホームページ: https://github.com/besser82/libxcrypt/
ダウンロード: https://github.com/besser82/libxcrypt/releases/download/v4.4.38/libxcrypt-4.4.38.tar.xz
MD5 sum: 1796a5d20098e9dd9e3f576803c83000

• Linux (6.16.9) - 149,102 KB:
ホームページ: https://www.kernel.org/
ダウンロード: https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.16.9.tar.xz
MD5 sum: feb0a3d5ecf5a4628aed7d9f8f7ab3f6

注記

Linux カーネルはかなり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。

たとえば linux-6.16.9 の場合、以下をダウンロードすることになります。
• https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.16.tar.xz
• https://www.kernel.org/pub/linux/kernel/v6.x/patch-6.16.9.xz
そして 「Linux-6.16.9 API ヘッダー」 と 「Linux-6.16.9」 においてカーネルを伸長 (解凍) します。 パッケージディレクトリに移動したら **xzcat ../patch-6.16.9.xz | patch -Np1** のコマンドによってパッチを当てます。

このとき別の新たなバージョンのカーネルが必要であったなら、その新たなバージョン向けのパッチのみを用います。 ただしその新たなバージョンがマイナーバージョンであったなら、そのマイナーバージョン向けのパッチすべてと、必要となるパッチの両方をダウンロードすることになります。

• Lz4 (1.10.0) - 379 KB:
ホームページ: https://lz4.org/
ダウンロード: https://github.com/lz4/lz4/releases/download/v1.10.0/lz4-1.10.0.tar.gz
MD5 sum: dead9f5f1966d9ae56e1e32761e4e675

• M4 (1.4.20) - 1,997 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/m4-1.4.20.tar.xz
MD5 sum: 6eb2ebed5b24e74b6e890919331d2132

• Make (4.4.1) - 2,300 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-4.4.1.tar.gz
MD5 sum: c8469a3713cbbe04d955d4ae4be23eeb

• Man-DB (2.13.1) - 2,061 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-2.13.1.tar.xz
MD5 sum: b6335533cbeac3b24cd7be31fdee8c83

• Man-pages (6.15) - 1,817 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-6.15.tar.xz
MD5 sum: 16f68d70139dd2bbcae4102be4705753

• MarkupSafe (3.0.3) - 79 KB:
ホームページ: https://palletsprojects.com/p/markupsafe/
ダウンロード: https://pypi.org/packages/source/M/MarkupSafe/markupsafe-3.0.3.tar.gz
MD5 sum: 13a73126d25afa72a1ff0daed072f5fe

• Meson (1.9.1) - 4,964 KB:
ホームページ: https://mesonbuild.com
ダウンロード: https://github.com/mesonbuild/meson/releases/download/1.9.1/meson-1.9.1.tar.gz
MD5 sum: 19e0a1091cec23d369dd77d852844195

• MPC (1.3.1) - 756 KB:
ホームページ: https://www.multiprecision.org/
ダウンロード: https://ftp.gnu.org/gnu/mpc/mpc-1.3.1.tar.gz
MD5 sum: 5c9bc658c9fd0f940e8e3e0f09530c62

• MPFR (4.2.2) - 1,471 KB:
ホームページ: https://www.mpfr.org/
ダウンロード: https://ftp.gnu.org/gnu/mpfr/mpfr-4.2.2.tar.xz
MD5 sum: 7c32c39b8b6e3ae85f25156228156061

• Ncurses (6.5-20250809) - 3,703 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/current/ncurses-6.5-20250809.tgz
MD5 sum: 679987405412f970561cc85e1e6428a2

• Ninja (1.13.1) - 286 KB:
ホームページ: https://ninja-build.org/
ダウンロード: https://github.com/ninja-build/ninja/archive/v1.13.1/ninja-1.13.1.tar.gz
MD5 sum: c35f8f55f4cf60f1a916068d8f45a0f8

• OpenSSL (3.5.4) - 51,944 KB:
ホームページ: https://www.openssl-library.org/
ダウンロード: https://github.com/openssl/openssl/releases/download/openssl-3.5.4/openssl-3.5.4.tar.gz
MD5 sum: 570a7ab371147b6ba72c6d0fed93131f

• Packaging (25.0) - 162 KB:
ホームページ: https://pypi.org/project/packaging/
ダウンロード: https://files.pythonhosted.org/packages/source/p/packaging/packaging-25.0.tar.gz
MD5 sum: ab0ef21ddebe09d1803575120d3f99f8

• Patch (2.8) - 886 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-2.8.tar.xz
MD5 sum: 149327a021d41c8f88d034eab41c039f

• Pcre2 (10.46) - 1,988 KB:
ホームページ: https://github.com/PCRE2Project/pcre2/
ダウンロード: https://github.com/PCRE2Project/pcre2/releases/download/pcre2-10.46/pcre2-10.46.tar.bz2
MD5 sum: 641f99b635ebb9332a9b6a8ce8e2f3cf

• Perl (5.42.0) - 14,084 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-5.42.0.tar.xz
MD5 sum: 7a6950a9f12d01eb96a9d2ed2f4e0072

• Pkgconf (2.5.1) - 321 KB:
ホームページ: https://github.com/pkgconf/pkgconf
ダウンロード: https://distfiles.ariadne.space/pkgconf/pkgconf-2.5.1.tar.xz
MD5 sum: 3291128c917fdb8fccd8c9e7784b643b

• Procps (4.0.5) - 1,483 KB:
ホームページ: https://gitlab.com/procps-ng/procps/
ダウンロード: https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-4.0.5.tar.xz
MD5 sum: 90803e64f51f192f3325d25c3335d057

• Psmisc (23.7) - 423 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-23.7.tar.xz
MD5 sum: 53eae841735189a896d614cba440eb10

• Python (3.13.7) - 22,236 KB:
ホームページ: https://www.python.org/
ダウンロード: https://www.python.org/ftp/python/3.13.7/Python-3.13.7.tar.xz
MD5 sum: 256cdb3bbf45cdce7499e52ba6c36ea3

• Python Documentation (3.13.7) - 10,183 KB:
ダウンロード: https://www.python.org/ftp/python/doc/3.13.7/python-3.13.7-docs-html.tar.bz2
MD5 sum: b84c0d81b2758398bb7f5b7411d3d908

• Readline (8.3) - 3,340 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-8.3.tar.gz
MD5 sum: 25a73bfb2a3ad7146c5e9d4408d9f6cd

• Sed (4.9) - 1,365 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-4.9.tar.xz
MD5 sum: 6aac9b2dbafcd5b7a67a8a9bcb8036c3

• Setuptools (80.9.0) - 1,290 KB:
ホームページ: https://pypi.org/project/setuptools/
ダウンロード: https://pypi.org/packages/source/s/setuptools/setuptools-80.9.0.tar.gz
MD5 sum: 82e1d67883b713f9493659b50d13b436

• Shadow (4.18.0) - 2,293 KB:
ホームページ: https://github.com/shadow-maint/shadow/
ダウンロード: https://github.com/shadow-maint/shadow/releases/download/4.18.0/shadow-4.18.0.tar.xz
MD5 sum: 30ef46f54363db1d624587be68794ef2

• Sqlite (3500400) - 3,099 KB:
ホームページ: https://sqlite.org
ダウンロード: https://sqlite.org/2025/sqlite-autoconf-3500400.tar.gz
MD5 sum: d74bbdca4ab1b2bd46d3b3f8dbb0f3db

• Sqlite Documentation (3500400) - 5,943 KB:
ホームページ: https://sqlite.org
ダウンロード: https://anduin.linuxfromscratch.org/LFS/sqlite-doc-3500400.tar.xz
MD5 sum: 63a62af5b35913459954e6e66876f2b8

注記
Linux From Scratch チームでは sqlite ドキュメントの zip ファイルを使って、ドキュメント用の tarball を独自に作り出しています。 これは不要な依存関係を持たないようにするためです。

• Systemd (258) - 16,580 KB:
ホームページ: https://systemd.io
ダウンロード: https://github.com/systemd/systemd/archive/v258/systemd-258.tar.gz
MD5 sum: 490c1c2bbe5b576ff9ca2848fbd31507

• Systemd Man Pages (258) - 768 KB:
ホームページ: https://systemd.io
ダウンロード: https://anduin.linuxfromscratch.org/LFS/systemd-man-pages-258.tar.xz
MD5 sum: 688045e26e6121dd8f7b9abf788e47d9

注記
Linux From Scratch チームは、systemd ソースにおいて提供される man ページの tarball を独自に生成しています。 これは、不要な依存関係を取り除くためです。

• Tar (1.35) - 2,263 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-1.35.tar.xz
MD5 sum: a2d8042658cfd8ea939e6d911eaf4152

• Tcl (8.6.17) - 11,450 KB:
ホームページ: https://tcl.sourceforge.net/
ダウンロード: https://downloads.sourceforge.net/tcl/tcl8.6.17-src.tar.gz
MD5 sum: 1ec3444533f54d0f86cd120058e15e48

• Tcl Documentation (8.6.17) - 1,170 KB:
ダウンロード: https://downloads.sourceforge.net/tcl/tcl8.6.17-html.tar.gz
MD5 sum: 60c71044e723b0db5f21be82929f3534

• Texinfo (7.2) - 6,259 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-7.2.tar.xz
MD5 sum: 11939a7624572814912a18e76c8d8972

• Time Zone Data (2025b) - 454 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/tzdata2025b.tar.gz
MD5 sum: ad65154c48c74a9b311fe84778c5434f

• Util-linux (2.41.2) - 9,388 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/v2.41/util-linux-2.41.2.tar.xz
MD5 sum: a2a3281ce76821c4bc28794fdf9d3994

• Vim (9.1.1806) - 18,368 KB:
ホームページ: https://www.vim.org
ダウンロード: https://github.com/vim/vim/archive/v9.1.1806/vim-9.1.1806.tar.gz
MD5 sum: e72f31be182f1ccf4b66bef46ac1e60e

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 (0.46.1) - 54 KB:
ホームページ: https://pypi.org/project/wheel/
ダウンロード: https://pypi.org/packages/source/w/wheel/wheel-0.46.1.tar.gz
MD5 sum: 65e09ee84af36821e3b1e9564aa91bd5

• XML::Parser (2.47) - 276 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-2.47.tar.gz
MD5 sum: 89a8e82cfd2ad948b349c0a69c494463

• Xz Utils (5.8.1) - 1,428 KB:
ホームページ: https://tukaani.org/xz
ダウンロード: https://github.com//tukaani-project/xz/releases/download/v5.8.1/xz-5.8.1.tar.xz
MD5 sum: cf5e1feb023d22c6bdaa30e84ef3abe3

• Zlib (1.3.1) - 1,478 KB:
ホームページ: https://zlib.net/
ダウンロード: https://zlib.net/fossils/zlib-1.3.1.tar.gz
MD5 sum: 9855b6d802d7fe5b7bd5b196a2271655

• Zstd (1.5.7) - 2,378 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/v1.5.7/zstd-1.5.7.tar.gz
MD5 sum: 780fc1896922b1bc52a4e90980cdda48

全パッケージのサイズ合計: 約 595 MB

3.3. 必要なパッチ

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

日本語訳情報

各パッチに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/development/bzip2-1.0.8-install_docs-1.patch
MD5 sum: 6a5ac7e89b791aae556de0f745916f7f

• Coreutils Internationalization Fixes Patch - 128 KB:
ダウンロード: https://www.linuxfromscratch.org/patches/lfs/development/coreutils-9.8-i18n-2.patch
MD5 sum: c800540039fb0707954197486b1bde70

• Expect GCC15 Patch - 12 KB:
ダウンロード: https://www.linuxfromscratch.org/patches/lfs/development/expect-5.45.4-gcc15-1.patch
MD5 sum: 0ca4d6bb8d572fbcdb13cb36cd34833e

• Glibc FHS Patch - 2.8 KB:
ダウンロード: https://www.linuxfromscratch.org/patches/lfs/development/glibc-2.42-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/development/kbd-2.9.0-backspace-1.patch
MD5 sum: f75cca16a38da6caa7d52151f7136895

全パッチの合計サイズ: 約 156.4 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、一連のディレクトリを作ります（ここには一時的なツールをインストールしていきます）。 一般ユーザーを生成して、このユーザーが利用す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「SBUs」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

本節では最終的な Linux システムを構成する各種部品を LFS ファイルシステムに追加します。 はじめに行うのは、限定的なディレクトリの生成です。 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のコマンドを実行します。

```
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。 root ユーザーのまま、ここでそのディレクトリを生成します。

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

注記

LFS の編集者は /usr/lib64 ディレクトリは意図的に利用しないこととしました。 ツールチェーンにおいてはこのディレクトリを利用しないように、手順をいくつか進めています。 何らかの理由によってこのディレクトリが出てきたとしたら（ビルド手順を誤っていた、LFS 構築後にバイナリーパッケージをインストールしたような場合には）、システムが壊れる場合があります。 したがってこのディレクトリが用いられていないことを常に確認してください。

4.3. LFS ユーザーの追加

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

lfs にログインする、あるいは非 root ユーザーから lfs に切り替える場合には、lfs に対してパスワードを設定しておくことが必要です （この反対に root ユーザーにログインしている状態から lfs にユーザー切り替えを行う場合には、パスワードは必要ありません）。 root ユーザーにおいて以下のコマンドを実行して、パスワードの設定を行います。

```
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{,/*},var,etc,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 su 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な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 ユーザーにより起動するシェルを開始します。 これは、仮想コンソール上から lfs によってログインして実現します。 あるいは以下のユーザー切り替えコマンドを実行します。

```
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、あるいは su コマンドとそのオプション「-」を使って lfs に切り替えた時、起動されるシェル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。

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

umask の設定については、すでに 「変数 $LFS と umask の設定」 において説明しています。

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、クロスコンパイル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、クロスコンパイラー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ーは、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ー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 ユーザーを必要としなくなったら、（必要に応じて）/etc/bash.bashrc を元に戻してください。

なお 「Bash-5.3」 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。

最新のシステムは複数プロセッサー（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ー数がいくつなのかを make プログラムの実行時に、コマンドラインオプション引数として、あるいは環境変数として指定します。 Intel コア i9-13900K プロセッサーは 8 P コア（P は performance の意味）、および 16 E コア（E は efficiency の意味）を持ちます。 1 つの P は同時に 2 つのスレッド実行が可能であり、Linux カーネルからは 2 つの論理コアとして扱われます。 したがって合計で 32 の論理コアを持つことになります。 明示的にその全コアを利用するには、make が 32 個のビルドジョブまで生成できるようにすることです。 これには make に対して *-j32* オプションを与えます。

```
make -j32
```

あるいは環境変数 MAKEFLAGS を用います。 この変数の設定値は make が自動的にコマンドラインオプションとして利用します。

```
export MAKEFLAGS=-j32
```

重要

-j オプションに数値を与えずに make コマンドに受け渡したり、あるいは MAKEFLAGS に設定することはやめてください。 それを行ってしまうと make に対して無限のビルドジョブ生成を行わせるものとなり、システムの安定性を損なう問題が発生します。

第 5 章 および 第 6 章 におけるパッケージのビルドに対して、利用可能な論理コアをすべて利用するように、ここで .bashrc にて MAKEFLAGS を設定します。

```
cat >> ~/.bashrc << "EOF"
export MAKEFLAGS=-j$(nproc)
EOF
```

$(nproc) 部分は、論理コアすべてでなく利用したい論理コア数を設定してください。

一時的なツールを構築する準備の最後として、bash シェルが、今作り出したユーザープロファイルを読み込むようにします。

```
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（gcc）の場合、処理性能の高いシステムでも 5 分はかかります。 それが性能の低いシステムとなると数日はかかるかもしれません! 本書では処理時間を正確に示すのでなく、標準ビルド単位（Standard Build Unit; SBU）を用いることにします。

SBU の測定は以下のようにします。 最初にコンパイルするのは 第 5 章における binutils です。 このパッケージを 1 コアのシステムによってコンパイルするの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を基準にして表現します。

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 4 分であったとすると、そのパッケージは およそ 18 分かかることを意味しています。 幸いにも、たいていのパッケージは 1 SBU よりもコンパイル時間は短いものです。

コンパイル時間というものは、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。

最新のシステムの場合、マザーボードにシステムクロック速度の制御機能があります。 これは powerprofilesctl などのコマンドを使って制御します。 LFS では利用できないものですが、ホストディストロでは利用できるものかもしれません。 LFS の構築を終えた後に BLFS power-profiles-daemon に示される手順に従えば、システムにその機能を追加することができます。 どのパッケージのビルド時間を調べる際であっても、システムの電源プロファイルセットは最大のパフォーマンス（最大消費電力）を発揮するように設定することが推奨されます。 これを行っていないと SBU の測定値は極端に不適切なものとなります。 binutils 1回め や他パッケージのビルドの際には、システムがさまざまな処理反応を示すためです。 測定するパッケージに対して同一プロファイルを用いていたとしても、極端に測定値が不適切となる場合もあることに留意しておいてください。 これはビルド処理開始の際にシステムがアイドル状態となっていると、システムの反応がより遅くなるためです。 電源プロファイルを「performance」に設定しておけば、この問題は解消します。 さらに LFS ビルドは明らかに早くなるはずです。

power-profiles-daemon が利用可能である場合は powerprofilesctl set performance コマンドを実行して `performance` プロファイルを選択します。 ディストロの中にはプロファイルの管理に powerprofilesctl ではなく tuned-adm コマンドを利用しているものがあります。 その場合は tuned-adm profile throughput-performance コマンドを実行して `throughput-performance` プロファイルを選択します。

注記

上のようにして複数プロセッサーが利用されると、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、単一プロセッサー処理を行ってエラーメッセージを分析してください。

ここに示す時間は（1 コアで処理を行う binutils 1回め を除き）4 コア（-j4）を使用した場合に基づいています。 また第 8 章では、特に断りがない限り、パッケージの縮退テストの実行時間も含めています。

4.6. テストスイートについて

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

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

注記

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に意味がありません。。 各テストプログラムはクロスコンパイラー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実行することができないためです。

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/development/ 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ー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ーを使って、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ーをビルドします。 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）、最終システムを構築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。 各コマンドが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。 どれだけ熱心にビルド作業を終わらせているとしても、ただ単に書かれている内容を入力する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 ユーティリティー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ー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っていないまま、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。

GCC の 2 回めはクロスツールチェーンを崩すものと捉えられています。 しかしこれはバグと扱うべきものではありません。 というのも GCC の 2 回めというのは、本書においてクロスコンパイルを行う最後のパッケージです。 そしてこの GCC 2 回めのビルド以降に、クロスコンパイルを本当に行う必要のあるパッケージが現れるまでは、この状況を「直したくない」のです。

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。 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同一ではありません。

ターゲット（target）
コンパイラーにおいてのみ用いられます。 コンパイラー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。） コンパイラー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、コンパイラーは 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ーがありません。 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 向けにコンパイラー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、クロスコンパイラー cc1 をビルド。
2	A	B	C	マシン A 上の cc1 を使い、クロスコンパイラー cc2 をビルド。
3	B	C	C	マシン B 上の cc2 を使い、コンパイラー 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になります。 たとえば ccC においてテストスイートを実行するには、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ーです。 つまりこのコンパイラーは、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 というコンパイラーは、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ーのこをネイティブ コンパイラーと呼びます。

LFS におけるクロスコンパイラーの実装方法

注記

本書におけるクロスコンパイルパッケージは、すべて autoconf ベースのビルドシステムを利用しています。この autoconf ベースのビルドシステムは、システムのタイプとして cpu-vendor-kernel-os という形式のシステムトリプレット（system triplet）を用いています。 ベンダー項目はたいていは正しくないため、autoconf では無視されます。

よく考えてみると、4 つの項目からなる名前なのに、どうして「3 つの組（triplet)」というのでしょう。 カーネルと OS の各項目は、元は「システム（system)」項目に由来しています。 したがって 3 つの項目からなる書式が、今も有効に扱われるシステムもあります。 たとえば `x86_64-unknown-freebsd` です。 異なる 2 システム間で同一カーネルを共有することも不可能ではありませんが、それにしても同一トリプレットとするには、あまりにも差異がありすぎます。 たとえば携帯端末で動作する Android は ARM64 サーバー上で動作する Ubuntu とでは、確かに同一の CPU タイプ（ARM64）であり、同一カーネル（Linux）を使っているとは言っても、明らかに違います。

エミュレーション層を設けない限り、携帯端末上にサーバー向け実行モジュールを起動することはできません。 この逆の場合も同様です。 そこで「システム（system)」フィールドは kernel と os に分けられ、システムを明確に指定するようになりました。 たとえば上の例における Android システムは `aarch64-unknown-linux-android` と指定され、Ubuntu システムは `aarch64-unknown-linux-gnu` と指定されるようになります。

「トリプレット」 という名前が辞書に残っただけです。 システム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ーであれば、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。 ほとんどの Linux システムでは、より簡単に gcc -dumpmachine コマンドを実行すれば、同様の情報を得ることができます。

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 パッケージ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、Glibc wiki ページ を見てみることです。

クロスコンパイルには以下の 2 つのキーポイントがあります。

- マシンコードを生成して処理するのが「ホスト」上である場合は、クロスツールチェーンを用いなければなりません。 「ビルド」システムにあるネイティブなツールチェーンは、「ビルド」システム上において実行することができるマシンコードを生成するため、まだ呼び出される場合があるかもしれません。 たとえばビルドシステムがネイティブツールチェーンを使って、ジェネレーターをコンパイルしたとします。 そのジェネレーターを使って C ソースファイルを生成し、最終的にクロスツールチェーンを使って C ソースファイルをコンパイルするかもしれません。 これによって生成されるコードは「ホスト」上で実行可能となります。

  autoconf ベースのビルドシステムを使った場合、この状況は *`--host`* スイッチを用いることで「ホスト」トリプレットを確実に設定することができます。 このスイッチを使えば、ビルドシステムは `<ホストのトリプレット>` というプリフィックスのついたツールチェーンコンポーネントを利用して、「ホスト」用のマシンコードを生成します。 具体的にコンパイラーは `<ホストのトリプレット>`-gcc となり、readelf ツールは `<ホストのトリプレット>`-readelf となります。

- 「ホスト」上において実行する目的で生成されたマシンコードを、ビルドシステム上で実行してはなりません。 たとえばネイティブにユーティリティーをビルドする際に、その man ページは *`--help`* スイッチを使ってユーティリティーを実行させ、その出力を得ようとするでしょう。 しかしクロスコンパイルを行っていた場合には、このユーティリティーを「ビルド」上で動作させることに失敗するため、出力は得られません。 特に x86 CPU 上において（エミュレーターはないものとして）ARM64 マシンコードが実行できないのは当然です。

autoconf ベースのビルドシステムを使った場合、この状況は「クロスコンパイルモード」を用いて実現可能であり、「ホスト」向けマシンコードの実行に必要となるオプション機能は、ビルド時には無効化されます。 「ホスト」のトリプレットが明示的に指定された場合は、configure スクリプトが「ホスト」向けにコンパイルされたダミープログラムの実行に失敗した場合、あるいは「ビルド」トリプレットが *--build* スイッチにより指定され、それが「ホスト」鳥プレととは異なる場合にのみ、「クロスコンパイルモード」が有効になります。

LFS の一時システム向けのパッケージをクロスコンパイルするため、LFS_TGT 変数においてシステムトリプレットの ”vendor” フィールドを ”lfs” として若干修正します。 そして「ホスト」トリプレットを *--host* を通じて指定することで、クロスツールチェーンが LFS 一時システムにおいて稼働するマシンコードを生成実行するようにします。 また「クロスコンパイルモード」を有効にすることも必要です。 「ホスト」向けマシンコード、すなわち LFS 一時システム向けのマシンコードは、現在の CPU 上で実行できるわけですが、「ビルド」（ホストディストロ）上では利用できないライブラリを参照するかもしれませんし、参照できたとしても、存在しなかったり異なった定義となってしまうようなコードやデータの参照が起きるかもしれません。 LFS 一時システム向けのパッケージをクロスコンパイルする際には、configure スクリプトがその問題を判別するダミープログラムのことを信用することはできません。 つまり（ホストディストロが Musl のように libc とは異なる実装をしていない限り）ダミープログラムは、ホストディストロの libc が提供する可能性のある、libc の機能の一部を利用するに過ぎないからです。 したがってよくできたプログラムが失敗するようであっても、処理には成功するのかもしれません。 そこで「ビルド」トリプレットの指定は明示的に行わなければならず、これによって「クロスコンパイルモード」を有効にします。 設定する値は普通のデフォルト値です。 つまり config.guess の出力から得られる、元々のシステムトリプレットです。 「クロスコンパイルモード」 は上で述べたように明示的な指定によって有効になります。

クロスリンカーとクロスコンパイラーをビルドする際には *--with-sysroot* オプションを使います。 これは「ホスト」が必要とするファイルの検索ディレクトリを指示するためです。 こうしておけば 第 6 章 においてビルドするそれ以外のプログラムが、「ビルド」上のライブラリにリンクすることをほぼ防げます。 ここで「ほぼ」という表現を使ったのは、コンパイラーの「互換性」ラッパーである libtool と、autoconf ベースのビルドシステム向けのリンカーは、よくできたものであるがために、リンカーに対して「ビルド」システム内のライブラリを検索するようなオプションを与えてしまいます。 このことを避けるために libtool アーカイブ（.la）ファイルは削除するものとし、Binutils に同梱されている古い libtool は修正するようにしています。

段階	ビルド	ホスト	ターゲット	作業
1	pc	pc	lfs	pc 上の cc-pc を使い、クロスコンパイラー ccl をビルド。
2	pc	lfs	lfs	pc 上の ccl を使い、クロスコンパイラー cc-lfs をビルド。
3	lfs	lfs	lfs	lfs 上の cc-lfs を使い cc-lfs の再ビルド（とおそらくテスト）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。

話はまだまだあります。 C 言語というと単にコンパイラーがあるだけではなく、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（別の選択肢として ”musl” があります）。 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ー ccl を使うということです。 しかしコンパイラーには内部ライブラリというものがあって、アセンブラー命令セットだけでは利用できない複

雑なサブルーチンが含まれます。 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。 このようなニワ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 ccl をビルドします。 この ccl 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。 その後に、この低機能なコンパイラーを使って 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 の機能がいくつか欠如しています。

上の段落における結論は以下のようになります。 グレードの落ちた libgcc を使っている以上、ccl 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ーです。 すでに述べているように cc-lfs は pc（ホストディストロ）上で実行することはできません。 それはライブラリー、コード、データのいずれかが利用できないにも関わらず、「ビルド」（ホストディストロ）上において必要となることがあるからです。 そこで gcc の第 2 段階においてはライブラリ検索パスを上書きして、新たに再ビルドされた libgcc に対してリンクされる libstdc++ を使ってビルドするようにします。 つまりはデグレードしたビルドです。 こうすることで、再ビルドした libstdc++ は完全に機能することになります。

第 8 章（つまり「3 回め」）において、LFS システムに必要なパッケージがすべてビルドされます。 それまでの章において、特定のパッケージがたとえ LFS システムにインストールされていても、再ビルドをし続けます。 そのようにしてパッケージを再ビルドする最大の理由は、そのパッケージを安定させるためです。 完全に仕上がった LFS システムに、どれかの LFS パッケージを再インストールしたとしたら、その際にインストールされる内容は、第 8 章 において初めてインストールされるものと、全く同一でなければなりません。 第 6 章 や 第 7 章 においてインストールする一時的なパッケージでは、この要件を満たしません。 なぜならそういった任意機能に対しては、依存パッケージがない、クロスコンパイルモードである、といった理由により利用できないからです。 さらにパッケージ再ビルドのもう一つの理由は、テストスイートを実行するためです。

その他の手順詳細

クロスコンパイラーは、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ーは、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ー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で、そこではどの機能が利用可能または利用不能であるかが確認されます。 ただ重要なのは binutils を一番初めにビルドするという点だけではありません。 gcc や glibc の configure が正しく処理されなかったとすると、ツールチェーンがわずかながらも不完全な状態で生成されてしまいます。 この状態は、すべてのビルド作業を終えた最後になって、大きな不具合となって現れてくることになります。 テストスイートを実行することが欠かせません。 これを実行しておけば、この先に行う多くの作業に入る前に不備があることが分かるからです。

Binutils はアセンブラー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 $LFS_TGT-ld --verbose | grep SEARCH を実行すると、検索するライブラリのパスとその検索順を示してくれます。（この例は見て分かるように **lfs** ユーザーでログインしている場合のみ実行することができます。 この後にもう一度このページに戻ってきたときには、$LFS_TGT-ld を単に ld と置き換えてください。）

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。（上でも述べたように、後でこのページに戻ってきたときは $LFS_TGT- の部分を取り除いてください。）

gcc からさらに詳細な情報を知りたいときは、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。 たとえば $LFS_TGT-gcc -v ***example.c*** と（あるいは後にここに戻ってきたときは $LFS_TGT- 部分を除いて）入力すると、プリプロセッサー、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 がインクルードヘッダーを検索する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。 これはクロスコンパイルを行う最初のパッケージになります。 ここでは ***--host=$LFS_TGT*** というオプションを使います。 これを行うことによって、ビルドシステムが **$LFS_TGT-** というプリフィックスのついたツール類を用いるようにするためです。 さらに ***--build=$(../scripts/config.guess)*** オプションを指定することによって、上で述べたような「クロスコンパイルモード」を有効にします。 環境変数 **DESTDIR** はインストール先を LFS ファイルシステムとするために利用します。

すでに述べたように、標準 C++ ライブラリはこの後にコンパイルします。 そして 第 6 章 では、自己依存性を持ったプログラムをビルドできるように、その依存性を無視するためにクロスコンパイル行っていきます。 これらのパッケージにおける手順は glibc の手順と同等です。

第 6 章 の最後には、LFS のネイティブコンパイラーをインストールします。 はじめに DESTDIR ディレクトリを使って binutils 2 回めをビルドし、他のプログラムにおいても同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、不必要なライブラリは省略します。

第 7 章での chroot による環境下では、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

注意

LFS を開発してきた中では、パッケージの更新やそれに伴う新機能に適応するために、本書内の手順を順次修正しています。 LFS ブックのバージョンが異なっているにもかかわらず、その手順を混同してしまうと、些細なエラーにつながります。 こういった問題は、一般的には LFS ブックの前バージョンに対して作り出したスクリプトを、そのまま再利用した結果として起こります。 スクリプトの再利用は是非行わないでください。 仮に何らかの理由があって前バージョン向けのスクリプトを再利用する場合であっても、最新バージョンの LFS ブック向けにそのスクリプトの更新を十分確認して行ってください。

パッケージをビルドしていくにあたって、理解しておくべき内容を以下に示します:

- 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ますが、同じパッケージを二度ビルドする場合であっても、パッチを必要としない場合があります。 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更新されていないものもあります。 重大な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

ビルド作業の概要を示します。

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. `/mnt/lfs/sources/` 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 からソースコードを抽出する際には tar コマンド以外による方法は用いないでください。 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、ソースツリー内のタイムスタンプを壊しかねません。 そうな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. ビルドが終了したら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ー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-2.45 - 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: 678 MB

5.2.1. クロスコンパイル版 Binutils のインストール

注記

全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    \
             --enable-new-dtags  \
             --enable-default-hash-style=gnu
```

configure オプションの意味

注記

他のパッケージとは違って、以下に示すオプションは ./configure --help の実行によってすべて表示されるわけではありません。 たとえば --with-sysroot オプションを確認するには ld/configure --help を実行する必要があります。 一度にすべてのオプションを確認したい場合は ./configure --help=recursive を実行します。

--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した場合に、ビルドが中断することがないようにします。

--enable-new-dtags
これは実行ファイルや共有ライブラリに埋め込むライブラリ検索パスとして「runpath」を用いることをリンカーに対して指示します。 従来の「rpath」タグは用いません。 こうすると、動的リンクされた実行ファイルのデバッグが容易になり、いくつかのパッケージにおけるテストスイートにおいて発生する潜在的な問題を解決できるものとなります。

--enable-default-hash-style=gnu
リンカーにおいては、共有ライブラリや動的リンク実行ファイルのハッシュテーブルに関して、GNU スタイルのものと旧来の ELF 形式のものの双方を生成することがデフォルトとなっています。 ハッシュテーブルは、動的リンカーがシンボル検索を実現するためのものです。 LFS における動的リンカー（Glibc パッケージから提供されるもの）は、GNU スタイルのハッシュを常に用いることにしており、シンボル検索をより早くなるようにしています。 したがって旧来の ELF ハッシュテーブルは完全に無用です。 本指定はリンカーに対して、デフォルトでは GNU スタイルのハッシュテーブルしか生成しないように指示します。 こうすることで、パッケージビルドの際に、旧来の ELF ハッシュテーブルを生成する不要な時間、およびそれを収容するディスクスペースを軽減できます。

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-15.2.0 - 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: 3.8 SBU
必要ディスク容量: 5.4 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-15.2.0 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-4.2.2.tar.xz
mv -v mpfr-4.2.2 mpfr
tar -xf ../gmp-6.3.0.tar.xz
mv -v gmp-6.3.0 gmp
tar -xf ../mpc-1.3.1.tar.gz
mv -v mpc-1.3.1 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
```

注記

上の処理例においては *-i.orig* スイッチを利用しています。 これは sed が t-linux64 ファイルのコピーを t-linux64.orig として残すものです。 その上で元の t-linux64 ファイルの内容を編集します。 したがってこの処理の後に diff -u gcc/config/i386/t-linux64{.orig,} を実行すれば、その編集結果を確認することができます。 本書のこれ以降のパッケージの処理においては、単に *-i* を用いていくことにします。（元のファイルを直接編集し、そのファイルのコピーは取らないものとします。）もし元のファイルのコピーを取っておく必要があれば *-i.orig* に置き換えてください。

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=2.42 \
    --with-sysroot=$LFS        \
    --with-newlib              \
    --without-headers          \
    --enable-default-pie       \
    --enable-default-ssp       \
    --disable-nls              \
    --disable-shared           \
    --disable-multilib         \
    --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=2.42
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 回めの 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。

--enable-default-pie と *--enable-default-ssp*
このスイッチは GCC がプログラムをコンパイルする際にデフォルトとして、堅牢なセキュリティ機能（詳しくは PIE と SSP に関するメモ 参照）をある程度含める指示を行います。 厳密には、この段階で必要となるものではありません。 と言うのも、ここでのコンパイラーは一時的な実行ファイルを生み出すだけのものだからです。 ただし一時的なパッケージだとしても、最終形とするパッケージにできるだけ近づけておけば、理解しやすくなります。

--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-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx
これらのオプションは順に、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ていると、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するかもしれません。 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。

注記

以下に示すコマンドは、2 つの手法、つまりバッククォートと $() 構文を使って、ネスト化したコマンド置換を行う例を示しています。 これは、両方の置換において一つの手法のみを使って書き換えることもできます。 ただしここでは、両者を混在させても実現できることを示すものです。 一般的には $() 構文による手法がよく用いられます。

```
cd ..
cat gcc/limitx.h gcc/glimits.h gcc/limity.h > \
  `dirname $($LFS_TGT-gcc -print-libgcc-file-name)`/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-6.16.9 API ヘッダー

Linux API ヘッダー（linux-6.16.9.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 以下
必要ディスク容量: 1.7 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-2.42

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: 1.4 SBU
必要ディスク容量: 870 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。 間違いと思われる場合には 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-2.42-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) \
      --disable-nscd                     \
      libc_cv_slibdir=/usr/lib           \
      --enable-kernel=5.4
```

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=5.4
Linux カーネル 5.4 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。

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。

--disable-nscd
nscd (name service cache daemon) は使われることがないのでビルドしないようにします。

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」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、root ユーザーではない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」 に入ってからは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
```

ここにクロスツールチェーンが用意できましたので、コンパイルとリンクが思うとおりに作動する確認をとることが必要になります。 以下の健全性チェックを実行してこれを確認します。

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

エラーは出力しないはずであり、最後のコマンドからの出力は（ダイナミックリンカー名がプラットフォームごとに異なることを除けば）以下のようになるはずです。

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

このパスには /mnt/lfs（あるいは LFS に指定しているパス）は含まれていないはずです。 こうなるのも、そもそもパスが解釈されるのは、コンパイルしたプログラムが実行される際です。 そしてここでは chroot 環境に入っているため、カーネルが $LFS をルートディレクトリ（/）であると解釈しているからこそ、そのようになります。

ファイルが正しくできていることを確認します。

```
grep -E -o "$LFS/lib.*/S?crt[1in].*succeeded" 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。

```
/mnt/lfs/lib/../lib/Scrt1.o succeeded
/mnt/lfs/lib/../lib/crti.o succeeded
/mnt/lfs/lib/../lib/crtn.o succeeded
```

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 -B3 "^ $LFS/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:
 /mnt/lfs/tools/lib/gcc/x86_64-lfs-linux-gnu/15.2.0/include
 /mnt/lfs/tools/lib/gcc/x86_64-lfs-linux-gnu/15.2.0/include-fixed
 /mnt/lfs/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("=/mnt/lfs/tools/x86_64-lfs-linux-gnu/lib64")
SEARCH_DIR("=/usr/local/lib64")
SEARCH_DIR("=/lib64")
SEARCH_DIR("=/usr/lib64")
SEARCH_DIR("=/mnt/lfs/tools/x86_64-lfs-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。 ただしここで重要なことは、パス名の先頭がすべて等号記号 (=) で始まっている点です。 これはリンカーに対して設定した sysroot ディレクトリに置き換わります。

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 /mnt/lfs/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 /mnt/lfs/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 a.out dummy.log
```

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。

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-15.2.0 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 Libstdc++ 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: 0.2 SBU
必要ディスク容量: 1.3 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-15.2.0 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++/15.2.0
```

configure オプションの意味

--host=...
利用するクロスコンパイラー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/$LFS_TGT/include/c++/15.2.0
インクルードファイルをインストールするディレクトリを指定します。 Libstdc++ は LFS における標準 C++ ライブラリであるため、そのディレクトリは C++ コンパイラー ($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++ コンパイラーは 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++/15.2.0 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++{,exp,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. M4-1.4.20

M4 パッケージはマクロプロセッサーを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 38 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-6.5-20250809

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: 54 MB

6.3.1. Ncurses のインストール

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。 インストール先は $LFS/tools とします。 これは必要なときに PATH を通じて利用することができます。

```
mkdir build
pushd build
  ../configure --prefix=$LFS/tools AWK=gawk
  make -C include
  make -C progs tic
  install progs/tic $LFS/tools/bin
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          \
            AWK=gawk
```

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。

--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、ホスト上の strip を、ビルドシステムが利用しないようにします。 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。

AWK=gawk
本スイッチはホスト上の mawk を使ったビルドが行われないようにするものです。 mawk のバージョンによっては、本パッケージのビルドに失敗するものがあるためです。

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
ln -sv libncursesw.so $LFS/usr/lib/libncurses.so
sed -e 's/^#if.*XOPEN.*$/#if 1/' \
    -i $LFS/usr/include/curses.h
```

install オプションの意味

ln -sv libncursesw.so $LFS/usr/lib/libncurses.so
: これから作り出すパッケージの中で、わずかですが libncurses.so を必要としているものがあります。 このシンボリックリンクは libncursesw.so に代わるものとして生成します。

sed -e 's/^#if.*XOPEN.*$/#if 1/' ...
: ヘッダーファイル curses.h では Ncurses データ構造に関するさまざまな定義が行われています。 プリプロセッサーマクロ定義を変えることによって、データ構造定義を二つの異なるセットとして定義しているものがあります。つまり 8 ビット定義は libncurses.so と互換性があり、ワイドキャラクター定義は libncursesw.so と互換性があります。 これまで libncurses.so の代わりとして libncursesw.so を利用してきていることから、ヘッダーファイルを修正して、libncursesw.so と互換性を持つワイドキャラクターデータ構造を常に用いるものとします。

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-5.3

Bash は Bourne-Again Shell を提供します。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 72 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=$(sh support/config.guess) \
            --host=$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-9.8

Coreutils パッケージは、あらゆるオペレーティングシステムが必要とする基本的なユーティリティ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: 181 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"/'                    $LFS/usr/share/man/man8/chroot.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-3.12

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

概算ビルド時間: 0.1 SBU
必要ディスク容量: 35 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=$LFS_TGT \
            gl_cv_func_strcasecmp_works=y \
            --build=$(./build-aux/config.guess)
```

configure オプションの意味

gl_cv_func_strcasecmp_works=y
本オプションは strcasecmp のチェック結果を示します。 そのチェックでは C プログラムをコンパイルして実行することが必要になりますが、クロスコンパイル環境ではこれを行うことができません。 それは一般に、クロスコンパイルしたプログラムをホストディストロで実行することはできないからです。 普通であれば configure スクリプトにおけるそのようなチェックにあたっては、代替となる値を用いるものです。 ただしこのチェックに対する代替値はなく configure はエラーとなります。 アップストリームはすでにこの問題を修正していますが、その修正を適用するには autoconf の実行が必要です。 これはひょっとするとホストディストロにはないかもしれません。 そこでここではそのチェック結果 (Glibc-2.42 の strcasecmp は正しく動作が分かっているので、そのチェック結果 y) を用いることにします。 そうすれば configure がその値を利用しチェックをスキップするようになります。

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
```

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-5.46

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: 42 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、file コマンドの一時的なコピーを生成します。

```
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-4.10.0

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。）Findutils では xargs プログラムも提供します。 このプログラムは、検索された複数ファイルの個々に対して、指定されたコマンドを実行するために用いられます。

概算ビルド時間: 0.2 SBU
必要ディスク容量: 48 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-5.3.2

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: 0.1 SBU
必要ディスク容量: 49 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-3.12

Grep パッケージはファイル内の検索を行うプログラムを提供します。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 32 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=$LFS_TGT \
            --build=$(./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 「Grep の構成」を参照してください。

6.11. Gzip-1.14

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: 12 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-4.4.1

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 以下
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   \
            --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 「Make の構成」を参照してください。

6.13. Patch-2.8

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: 14 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-4.9

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: 21 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=$LFS_TGT \
            --build=$(./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 「Sed の構成」を参照してください。

6.15. Tar-1.35

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: 0.1 SBU
必要ディスク容量: 42 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-5.8.1

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: 23 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-5.8.1
```

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-2.45 - 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: 0.4 SBU
必要ディスク容量: 548 MB

6.17.1. Binutils のインストール

Binutils によるビルドシステムでは、内部的なスタティックライブラリにリンクさせる libtool は、内部にコピーしたバージョンを用いています。 しかしこのパッケージが提供する内部コピーバージョンである libiverty と zlib は、その libtool を利用していません。 このような不整合があるため、生成されるバイナリが誤ってホスト内のライブラリにリンクされてしまう場合があります。 これを回避するために以下を実行します。

```
sed '6031s/$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        \
    --enable-new-dtags         \
    --enable-default-hash-style=gnu
```

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 アーカイブ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,sframe}.{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-15.2.0 - 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: 4.5 SBU
必要ディスク容量: 6.0 GB

6.18.1. GCC のインストール

GCC の 1 回めのビルドと同様に、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に移動させます。

```
tar -xf ../mpfr-4.2.2.tar.xz
mv -v mpfr-4.2.2 mpfr
tar -xf ../gmp-6.3.0.tar.xz
mv -v gmp-6.3.0 gmp
tar -xf ../mpc-1.3.1.tar.gz
mv -v mpc-1.3.1 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=$LFS_TGT          \
    --prefix=/usr              \
    --with-build-sysroot=$LFS  \
    --enable-default-pie       \
    --enable-default-ssp       \
    --disable-nls              \
    --disable-multilib         \
    --disable-libatomic        \
    --disable-libgomp          \
    --disable-libquadmath      \
    --disable-libsanitizer     \
    --disable-libssp           \
    --disable-libvtv           \
    --enable-languages=c,c++   \
    LDFLAGS_FOR_TARGET=-L$PWD/$LFS_TGT/libgcc
```

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 $LFS にあるものが用いられるように指示されます。 しかし GCC 向けのビルドシステムは追加のツールを使っているので、上のような場所を認識できていません。 本スイッチは、そのツール類が必要とするファイルを、ホスト内からではなく、$LFS から探し出すようにします。

--target=$LFS_TGT
GCC はクロスコンパイルによって作り出してきているので、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ます。

LDFLAGS_FOR_TARGET=...
libstdc++ が今回ビルドされたライブラリ libgcc を用いるようにします。 それは gcc 1 回め においてビルドされた前回のバージョンではありません。 前回のバージョンは C++ 例外処理を適切に処理できません。 これは libc サポートを抜きにしてビルドされているためです。

--disable-libsanitizer
GCC のサニタイザーランタイムライブラリを無効にします。 これはここでの一時的インストールにおいては不要です。 gcc 1 回め においては、*--disable-libstdcxx* によって暗にそれを行っていましたが、ここではそれを明示的に行う必要があります。

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、パッケージビルドに必要となる多くの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 環境に入る前には、あらかじめマウントされているはずです。 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になります。 これは危険なことです。 後にユーザーアカウントが生成され同一のユーザーIDを持ったとすると $LFS 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 --from lfs -R root:root $LFS/{usr,var,etc,tools}
case $(uname -m) in
  x86_64) chown --from lfs -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

ユーザー名前空間内において稼働するアプリケーションは、カーネルが生成するさまざまなファイルシステムを使って、カーネルとのやり取りを行います。 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。 こういったファイルシステムは $LFS ディレクトリツリー内にマウントされていなければならず、それができて初めて、アプリケーションが chroot 環境内にてそれを認識できるようになります。

この仮想ファイルシステムがマウントされるディレクトリを、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

LFS システムの通常のブートの際に、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。 カーネルはブートプロセスを通じて、仮想ファイルシステム上にデバイスノードを生成します。 またデバイスが初めて検出されるかアクセスされるかした際に生成します。 udev デーモンは、カーネルが生成したデバイスノードの所有者やパーミッションを変更することがあります。 またディストリビューション管理者やシステム管理者の作業をやりやすくするために、新たなデバイスノードやシンボリックリンクを生成することもあります。（詳しくは 「デバイスノードの生成」 を参照してください。）ホストのカーネルが devtmpfs をサポートしている場合は、devtmpfs を $LFS/dev 上に簡単にマウントすることができ、デバイスの有効化をカーネルに委ねることができます。

しかしホストカーネルの中には、devtmpfs をサポートしていないものがあり、そういったディストリビューションでは /dev の内容を別の手法によって実現しています。 そこでホストに依存せずに $LFS/dev ディレクトリを有効にするには、ホストシステムの /dev ディレクトリをバインドマウントします。 バインドマウントは特殊なマウント方法の一つであり、ディレクトリのサブツリーやファイルを、別の場所から見えるようにするものです。 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 -vt devpts devpts -o gid=5,mode=0620 $LFS/dev/pts
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

devpts に対するマウントオプションの意味

gid=5
このオプションは、devpts により生成されるデバイスノードを、グループID が 5 となるようにするものです。 この ID は、この後に tty グループにおいて利用します。 ここではグループ名ではなくグループ ID を用いるものとしています。 この理由は、ホストシステムが tty グループに対して異なる ID を利用していることがあるためです。

mode=0620
このオプションは、devpts により生成されるデバイスノードのモードを 0620 にします。（所有ユーザーが読み書き可、グループが書き込み可）前のオプションとともにこのオプションを指定することによって、devpts が生成するデバイスノードが grantpt() の要求を満たすようにします。 これはつまり、Glibc の ヘルパーコマンド pt_chown（デフォルトではインストールされない）が必要ないことを意味します。

ホストシステムによっては /dev/shm が通常 /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、これを行うのはこのディレクトリを適切なパーミッションにより生成する必要がある場合のみです。

別のホストシステムでは /dev/shm が tmpfs へのマウントポイントの場合があります。 その場合 /dev のマウントは /dev/shm をchroot 環境内のディレクトリとして生成します。 この状況においては tmpfs を明示的にマウントしなければなりません。

```
if [ -h $LFS/dev/shm ]; then
  install -v -d -m 1777 $LFS$(realpath /dev/shm)
else
  mount -vt tmpfs -o nosuid,nodev tmpfs $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、ここまでにすべてビルドしました。 そこで chroot 環境に入って、一時的ツールのインストールを済ませます。 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用います。 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     \
    MAKEFLAGS="-j$(nproc)"      \
    TESTSUITEFLAGS="-j$(nproc)" \
    /bin/bash --login
```

本章と次章のパッケージビルドにおいて、論理コアをすべて利用したくない場合、*$(nproc)* の部分は、利用したい論理コア数に書き換えてください。 第 8 章 において（特に Autoconf、Libtool、Tar など）は、テストスイートにおいて MAKEFLAGS を参照しないものがあり、そこでは環境変数 TESTSUITEFLAGS が代わりに用いられています。 そこでここでは同様にして、テストスイートを複数コアにより実行するための設定も行います。

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 などが必要であれば、ここで定義しておくと良いでしょう。

ここから先は LFS 変数は不要となります。 すべての作業は LFS ファイルシステム内で行っていくことになるからです。 chroot コマンドは、$LFS ディレクトリがルート（/ ディレクトリ）となるようにして Bash シェルを起動します。

/tools/bin が PATH 内には存在しません。 つまりクロスチェーンは、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します。

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/lib/locale
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、どのディレクトリであっても、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、追加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 LFS では、本当に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。

警告

FHS ではディレクトリ /usr/lib64 の利用を必須とはしていません。 そこで LFS 編集者はこれを利用しないことに取り決めました。 LFS や BLFS での手順を有効なものにするためには、このディレクトリをないものとして扱うことが必要です。 このディレクトリがないことを繰り返し確認してください。 うっかり生成してしまうようなことがあると、システムが壊れてしまうことがあるからです。

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きました。 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
systemd-journal-gateway:x:73:73:systemd Journal Gateway:/:/usr/bin/false
systemd-journal-remote:x:74:74:systemd Journal Remote:/:/usr/bin/false
systemd-journal-upload:x:75:75:systemd Journal Upload:/:/usr/bin/false
systemd-network:x:76:76:systemd Network Management:/:/usr/bin/false
systemd-resolve:x:77:77:systemd Resolver:/:/usr/bin/false
systemd-timesync:x:78:78:systemd Time Synchronization:/:/usr/bin/false
systemd-coredump:x:79:79:systemd Core Dumper:/:/usr/bin/false
uuidd:x:80:80:UUID Generation Daemon User:/dev/null:/usr/bin/false
systemd-oom:x:81:81:systemd Out Of Memory Daemon:/:/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:
clock:x:14:
cdrom:x:15:
adm:x:16:
messagebus:x:18:
systemd-journal:x:23:
input:x:24:
mail:x:34:
kvm:x:61:
systemd-journal-gateway:x:73:
systemd-journal-remote:x:74:
systemd-journal-upload:x:75:
systemd-network:x:76:
systemd-resolve:x:77:
systemd-timesync:x:78:
systemd-coredump:x:79:
uuidd:x:80:
systemd-oom:x:81:
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 (https://refspecs.linuxfoundation.org/lsb.shtml 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 systemd 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

wtmp, btmp, lastlog の各ファイルでは、32 ビットの整数値を使ってタイムスタンプを表現していますが、これは 2038 年以降には基本的に壊れるでしょう。 多くのパッケージにおいてはこういった利用を停止しており、その他についても順次停止予定です。 したがってこういった表記は非推奨であると捉えるべきです。

7.7. Gettext-0.26

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: 1.5 SBU
必要ディスク容量: 463 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: 0.2 SBU
必要ディスク容量: 58 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-5.42.0

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: 0.6 SBU
必要ディスク容量: 295 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                                         \
             -D prefix=/usr                               \
             -D vendorprefix=/usr                         \
             -D useshrplib                                \
             -D privlib=/usr/lib/perl5/5.42/core_perl     \
             -D archlib=/usr/lib/perl5/5.42/core_perl     \
             -D sitelib=/usr/lib/perl5/5.42/site_perl     \
             -D sitearch=/usr/lib/perl5/5.42/site_perl    \
             -D vendorlib=/usr/lib/perl5/5.42/vendor_perl \
             -D vendorarch=/usr/lib/perl5/5.42/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。

-D vendorprefix=/usr
これは perl に対して、Perl モジュールをどこにインストールするのかを指示するものです。

-D useshrplib
Perl モジュールの中には libperl をスタティックライブラリではなく共有ライブラリとして必要とするものがあるため、これをビルドします。

-D privlib,-D archlib,-D sitelib,...
この設定は、Perl がインストール済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 (5.42) 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と、新たなパッチレベル (5.42.0 のようなフルバージョンにおいて最後のドット以降のバージョン部分) 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
```

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-3.13.7

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。 Python はインタープリター言語です。

概算ビルド時間: 0.5 SBU
必要ディスク容量: 546 MB

7.10.1. Python のインストール

注記

「python」 というプレフィックス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-3.13.7.tar.xz` です。 (l 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 \
            --without-static-libpython
```

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。

`--without-static-libpython`
本スイッチは、膨大でしかも不要なスタティックライブラリ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
```

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。 `ssl` モジュールに対しては `Python requires a OpenSSL 1.1.1 or newer` というメッセージが出力されます。 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-7.2

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: 152 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-2.41.2

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: 0.2 SBU
必要ディスク容量: 192 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 --libdir=/usr/lib     \
            --runstatedir=/run    \
            --disable-chfn-chsh   \
            --disable-login       \
            --disable-nologin     \
            --disable-su          \
            --disable-setpriv     \
            --disable-runuser     \
            --disable-pylibmount  \
            --disable-static      \
            --disable-liblastlog2 \
            --without-python      \
            ADJTIME_PATH=/var/lib/hwclock/adjtime \
            --docdir=/usr/share/doc/util-linux-2.41.2
```

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 uuidd 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ファイルは削除します。 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}/*
```

最近の Linux システムにおいて libtool の .la ファイルは、libltdl に対してのみ用いられます。 LFS 内のライブラリは、libltdl によってロードされるものは一つもありません。 これらのライブラリによって BLFS パッケージのビルドに失敗することが分かっています。 そこでそのようなファイルをここで削除します。

```
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、$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 と umask の設定」 において説明済です。

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。

```
mountpoint -q $LFS/dev/shm && umount $LFS/dev/shm
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-r12.4-29-systemd.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しているので、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-r12.4-29-systemd.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。

コンパイラーにはカスタマイズ可能な最適化がありますが、これを利用することはお勧めしません。 コンパイラーのカスタマイズ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カスタマイズ最適化によってパッケージビルドが出来なかったら、最適化をなしにしてもう一度コンパイルすることで解決するかどうかを確認してください。 最適化を行ってパッケージがコンパイル出来たとしても、コードとビルドツールの複雑な関連に起因してコンパイルが適切に行われないリスクをはらんでいます。 また `-march` オプションや `-mtune` オプションにて指定する値は、本書には明示しておらずテストも行っていませんので注意してください。 これらはツールチェーンパッケージ（Binutils、GCC、Glibc）に影響を及ぼすことがあります。 最適化オプションを用いることによって得られるものがあったとしても、それ以上にリスクを伴うことがしばしばです。 初めて LFS 構築を手がける方は、最適化オプションをなしにすることをお勧めします。

一方で、各パッケージにおける最適化のデフォルト設定は、そのまま用いることにします。 さらにデフォルトでは有効になっていないものであっても、パッケージが提供する最適化設定を有効にする場合もあります。 パッケージ管理者はそういった設定についてのテストは行っていて、安全だと考えているからです。 したがってその設定を利用しても、ビルドに失敗することはないはずです。 一般的にデフォルトの設定では `-O2` または `-O3` を有効にしています。 つまりビルドされる結果のシステムは、他のカスタマイズ最適化オプションがなくても、充分に早く動作し、同時に安定しているはずです。

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

注記

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 値は特に断りのない限り、4 CPU コア（-j4）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のほとんどは、現在の Linux システムにとってはもはや古いものになっています。 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 などにおいては、パッケージビルドの手順にとって重要な機能となるため、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優れていれば、パッケージを再インストールしたりアップグレードしたりするときでも、ユーザーによる設定を保持しつつ、設定ファイルを適切に取り扱ってくれます。 パッケージ管理ツールでは、バイナリファイルやライブラリファイルだけ

でなく、設定ファイル類のインストールも取り扱います。 パッケージ管理ツールをどうしたら・・・ いえいえ本節は特定のパッケージ管理ツールを説明するわけでなく、その利用を勧めるものでもありません。 もっと広い意味で、管理手法にはどういったものがあり、どのように動作するかを説明します。 あなたにとって最適なパッケージ管理がこの中にあるかもしれません。 あるいはそれらをいくつか組み合わせて実施することになるかもしれません。 本節ではパッケージのアップグレードを行う際に発生する問題についても触れます。

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

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

ヒントプロジェクト（Hints Project）ページにパッケージ管理についての情報が示されています。 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はありません。 カーネルとユーザー空間のインターフェース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す。 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。 アップグレードしたカーネルは、システムを再起動しさえすれば利用できるようになります。
- Glibc を新しいバージョン（たとえば Glibc-2.31 から Glibc-2.42）にアップグレードする必要が発生した場合は、システムが壊れないようにすることが必要です。 詳しくは 「Glibc-2.42」 を参照してください。
- 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[1] 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。 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。
- あるパッケージが（直接的か間接的に）一つの共有ライブラリにリンクしていて、しかも古いライブラリ名と新しいライブラリ名にリンクしているとします。 （たとえばそのパッケージが **libfoo.so.2** と **libbar.so.1** にリンクしていて、さらに後者のライブラリは **libfoo.so.3** にリンクしているとします。） その場合にはパッケージが誤動作する可能性があります。 なぜなら共有ライブラリのリビジョンが異なると、一部のシンボル名に対する定義の互換性が失われる可能性があるからです。 こういった状況が起こりうるのは、共有ライブラリを提供するパッケージがアップグレードされた際に、古い共有ライブラリ名にリンクしているパッケージを（すべてではなく）一部だけ再ビルドしたような場合です。 この問題を回避するため、共有ライブラリにリンクするパッケージをすべて、（たとえば libfoo.so.2 から libfoo.so.3 のように）アップグレードされたリビジョンを使ってできるだけ、早くに再ビルドすることです。
- 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 **libfoo.so.1** はそのままの名前であったとして、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、パッケージの作者がバージョン番号づけの取り決めを変更してしまった場合に起こり得るものです。

[1] 共有ライブラリ名は、その ELF 動的セクションの **DT_SONAME** エントリーに名称がコーディングされます。 readelf -d **<ライブラリファイル名>** | grep SONAME コマンドを使えば、それを取得することができます。 その後ろには **.so.<バージョン番号>** がつくのが普通です。 ただし時には（**libbz2.so.1.0** などのように）複数のバージョン番号がつく場合、**.so** の前に（**libbfd-2.45** などのように）つく場合、または（**libmemusage.so** のように）まったくバージョン番号がつかない場合もあります。 一般的に言って、パッケージバージョンとライブラリ名の中のバージョン番号は関連性がありません。

- 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、どういったプロセス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 '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、前に示した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。

  systemd デーモンが（PID 1 として実行されていて）、アップデートしたライブラリにリンクされていた場合は、リブートするのではなく、root ユーザーになって systemctl daemon-reexec を実行すれば再起動できます。

- 実行プログラムや共有ライブラリが上書きされると、その実行プログラム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、プログラム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、たいていのパッケージにおいて、バイナリファイルやライブラリをインストールするコマンドとして利用しています。 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す。 しかしパッケージの中には（特に BLFS にある SpiderMonkey 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は、いつでもシステム全体を再構築しようと考える人なら、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テクニックであり、パッケージ管理のための特別なプログラムを必要としません。 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 /opt/foo-1.1 ディレクトリにインストールし、この /opt/foo-1.1 に対するシンボリックリンク /opt/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 がリリースされた際には /opt/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、MANPATH、INFOPATH、PKG_CONFIG_PATH、CPPFLAGS、LDFLAGS といった環境変数、あるいは設定ファイル /etc/ld.so.conf に対しては、/opt/foo ディレクトリを加えることで、対応する /opt/foo-x.y ディレクトリを含める必要があるかもしれません。

このやり方は BLFS ブックが採用するものであり、大規模のパッケージを用意にアップグレードできるようにします。 ただし多数のパッケージをインストールするとなると、このやり方では管理がしにくくなってきます。 また（たとえば Linux API ヘッダーや Glibc などのように）パッケージの中には、このやり方ではうまく動作しないものも出てきてしまいます。 このやり方は、システム全体に渡るものについては用いないでください。

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、上で説明した方法と同じようにインストールします。 ただし先ほどのように、汎用的なパッケージ名による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はユーザーが生成することもできますが、パッケージ管理者の多くは、この手法を使っています。 よく知られているものとして Stow、Epkg、Graft、Depot があります。

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

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

インストール自体は動作しますが、このパッケージに依存している他のパッケージは期待どおりには libfoo を正しくリンクしません。 例えば libfoo をリンクするパッケージをコンパイルする際には /usr/lib/libfoo.so.1 がリンクされると思うかもしれませんが、実際には /usr/pkg/libfoo/1.1/lib/libfoo.so.1 がリンクされることになります。 結局、正しい方法は DESTDIR 変数を使って、パッケージを直接インストールすることです。 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、インストールされるパッケージが、その時には一つだけであ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と、インストール中に予期しない副作用が発生するかもしれません。 したがって、ある程度のテスト確認を行って、パッケージ管理ツールが不具合を引き起こさないこと、しかるべきファイルの記録が取られていることが良いとされます。

別の方法として strace を用いるものがあります。 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たり、他の PC へのインストールにも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 https://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。 他のコンピューター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 900MB 程度）をまとめ、これをネットワーク転送か、あるいは CD-ROM や USB スティック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。 その後は、設定ファイルにいくらかの変更を行うことが必要です。 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`

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、カーネルを新しいシステム向けに再ビルドする必要があるでしょう。

重要

LFS システムを CPU の異なるシステム上にデプロイしたい場合、「GMP-6.3.0」 と 「Libffi-3.5.2」 のビルドにあたっては、以下のメモ内容に従ってください。 つまりアーキテクチャー固有の最適化を通じて、ホストシステムと LFS デプロイ先のシステム双方に適したライブラリを生成するようにしてください。 これを行っていないと LFS 実行時に `Illegal Instruction` エラーが発生することになります。

GMP ビルドシステムは GMP をビルドするために必要となる各アーキテクチャー特有の最適化オプションを `gmp.h` に保存します。 そして GMP を利用するパッケージでは、そのヘッダーファイルから最適化オプションを呼び込んで、パッケージビルドに利用するものがあります。 少なくとも MPFR のビルドシステムは、この仕組みを理解し利用しています。 完全な LFS システムにとって、GMP を単に再ビルドしようとする作業は不十分な事態を引き起こします。 もし他 CPU 向けに完全 LFS システムを「移行」しようとするなら、MPFR はもちろん、おそらく GMP を利用する他のパッケージもすべて再ビルドする必要があります。

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-6.15

Man-pages パッケージは 2,400 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
必要ディスク容量: 52 MB

8.3.1. Man-pages のインストール

パスワードのハッシュ処理に関する man ページを 2 つ削除します。 その man ページは Libxcrypt が、より良いものを提供してくれます。

```
rm -v man3/crypt*
```

Man-pages をインストールするために以下を実行します。

```
make -R GIT=false prefix=/usr install
```

オプションの意味

-R
これは make がビルトイン変数を設定しないようにします。 man-pages のビルドシステムにおいては、ビルトイン変数が適切に制御できません。 現状においてコマンドラインから *-R* を指定する以外に、それを制御する方法は存在していません。

GIT=false
これはビルドシステムが `git: command not found` という警告メッセージ行を出力しないようにします。

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-20250926

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-2.42

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: 12 SBU
必要ディスク容量: 3.3 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-2.42-fhs-1.patch
```

BLFS における Valgrind に不備を起こす問題を修正します。

```
sed -e '/unistd.h/i #include <string.h>' \
    -e '/libc_rwlock_init/c\
  __libc_rwlock_define_initialized (, reset_lock);\
  memcpy (&lock, &reset_lock, sizeof (lock));' \
    -i stdlib/abort.c
```

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                   \
             --disable-nscd                     \
             libc_cv_slibdir=/usr/lib           \
             --enable-stack-protector=strong \
             --enable-kernel=5.4
```

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=5.4
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 5.4 のように古くても、 Glibc が利用されるように指示します。 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リアンブル内に、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）のようなバッファーオーバーフローをチェックします。 なお Glibc に対して明示的に指定されたオプションは、常に GCC のデフォルトを上書きします。 したがってこのオプションは、GCC に対して --enable-default-ssp の設定を行っているからこそ、ここでも指定が必要になります。

--disable-nscd
nscd（name service cache daemon）は使われることがないのでビルドしないようにします。

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。 6000 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-preadvwritev2 と misc/tst-preadvwritev64v2 は、カーネルが Linux-6.14 またはそれ以降のときに失敗します。
- nss/tst-nss-files-hosts-multi、nptl/tst-thread-affinity* のようなテストでは、タイムアウトを原因として（特に比較的処理性能の低いシステムの利用や平行ビルドを使ったテストスイート実行において）そのテストが失敗します。 このようなテストは以下を実行すれば一覧として得られます。

```
grep "Timed out" $(find -name \*.out)
```

TIMEOUTFACTOR=***<factor>*** make test t=***<test name>*** というコマンド実行により、テスト 1 つずつに対してタイムアウト時間を拡張して再実行することができます。 たとえば TIMEOUTFACTOR=10 make test t=nss/tst-nss-files-hosts-multi とすれば nss/tst-nss-files-hosts-multi のテストが、元々のタイムアウトの 10 倍としながら再実行できます。

- さらに CPU モデルが古い場合に（たとえば elf/tst-cpu-features-cpuinfo が）、またホストのカーネルバージョンが古い場合に（たとえば stdlib/tst-arc4random-thread が）、それぞれ失敗することがあります。

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、古くなった健全性チェックをスキップするようにします。 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
```

重要

稼働中の LFS システムにおいて Glibc のマイナーバージョンを最新のものにアップグレードする場合（たとえば Glibc-2.36 を Glibc-2.42 にあげる場合)、いくつか追加の措置を講じることで、システムが壊れないようにすることが必要です。

- 11.0 未満の Glibc をアップグレードすることは LFS システムにおいてはサポートしていません。 そのような古い LFS システムにおいて最新の Glibc が必要になる場合は、LFS を再構築してください。
- 12.0 未満の Glibc をアップグレードする場合は 「Libxcrypt-4.4.38.」 に従って Libxcrypt をインストールしてください。 さらにその Libxcrypt の通常インストール手順に加えて、Libxcrypt の注記の節に示されている手順に従って **libcrypt.so.1*** のインストールを必ず行ってください（それ以前に行っている Glibc のビルドにおいてインストールした **libcrypt.so.1** を置き換えてください)。.
- 12.1 未満の Glibc をアップグレードする場合は nscd プログラムを削除してください。

```
rm -f /usr/sbin/nscd
```

（12.1 未満の）システムが Systemd に基づいている場合は、nscd サービスは停止させ無効化します。

```
systemctl disable --now nscd
```

- （現時点のカーネルバージョンは uname -r により確認することができますが）5.4 よりも古いカーネルをアップグレードして再起動する場合には 「Linux-6.16.9」 の説明に従ってください。
- （現時点の API ヘッダーバージョンは cat /usr/include/linux/version.h により確認することができますが）5.4 よりも古い API ヘッダーをアップグレードする場合は、「Linux-6.16.9 API ヘッダー」 の説明に従ってください（ただし cp コマンドからは **$LFS** を取り除いてください)。
- DESTDIR を利用したインストール方法により Glibc の共有ライブラリをアップグレードする場合には、以下のように一つの install コマンドにより行ってください。

```
make DESTDIR=$PWD/dest install
install -vm755 dest/usr/lib/*.so.* /usr/lib
```

自分が何をしているのかを完全に理解できていないのであれば、この手順に忠実に従ってください。 不用意にこの手順を見逃して進めてしまうと、システムが完全に利用不能になりかねません。ここに警告しておきます。

make install コマンド、**/usr/bin/ldd** に対する sed コマンド、ロケールをインストールするコマンドを順に実行します。 ここまで行ったら即座にシステムを再起動してください。

システムの再起動が成功したら次を行います。 今起動させている LFS システムが 12.0 よりも最新のものであって、GCC のビルド時に *--disable-fixincludes* オプションをつけていなかった場合は、2 つの GCC ヘッダーファイルを適切な位置に移動させた上で Glibc ヘッダーファイルにおける「fixed」を削除します。

```
DIR=$(dirname $(gcc -print-libgcc-file-name))
[ -e $DIR/include/limits.h ]    || mv $DIR/include{-fixed,}/limits.h
[ -e $DIR/include/syslimits.h ] || mv $DIR/include{-fixed,}/syslimits.h
rm -rfv $DIR/include-fixed/*
unset DIR
```

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
```

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、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パッケージがあるかもしれません。

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。

```
localedef -i C -f UTF-8 C.UTF-8
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 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-2.42/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
```

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 systemd
group: files systemd
shadow: files systemd

hosts: mymachines resolve [!UNAVAIL=return] files myhostname 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 -xf ../../tzdata2025b.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 tz
```

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、さほど気に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, 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, 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/i18n, /usr/share/zoneinfo, /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。
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-2.42/locale/broken_cur_max.c に書かれたコメントを参照してください。
libanl	関数を何も含まないダミーライブラリ。 かつては非同期の名前解決（asynchronous name lookup）ライブラリでしたが、今その関数は libc に含まれます。
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。
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.1c Threads Extensions によって規定されているインターフェースと、POSIX.1b Real-time Extensions によって規定されるセマフォー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。
libresolv	インターネットドメインネームサーバーに対しての、パケットの生成、送信、解析を行う関数を提供します。
librt	POSIX.1b リアルタイム拡張（Realtime Extension）にて既定されているインターフェースをほぼ網羅した関数を提供します。
libthread_db	マルチスレッドプログラム用のデバッガーを構築するための有用な関数を提供します。

libutil　　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 libc に含まれるようになりました。

8.6. Zlib-1.3.1

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: 6.4 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
```

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
```

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: 0.1 SBU 以下
必要ディスク容量: 7.3 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 (bzdiff へのリンク), bzdiff, bzegrep (bzgrep へのリンク), bzfgrep (bzgrep へのリンク), bzgrep, bzip2, bzip2recover, bzless (bzmore へのリンク), bzmore
インストールライブラリ: libbz2.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-5.8.1

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: 24 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-5.8.1
```

パッケージをコンパイルします。

```
make
```

ビルド結果をテストする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 (xzdiff へのリンク), lzdiff (xzdiff へのリンク), lzegrep (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 (xzdiff へのリンク), xzdec, xzdiff, xzegrep (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-5.8.1

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。
lzdiff LZMA 圧縮されたファイルに対して diff を実行します。
lzegrep 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. Lz4-1.10.0

Lz4 は可逆圧縮アルゴリズムであり、1 コアあたり 500 MB/秒を超える圧縮速度を誇ります。 非常に高速なデコーダーも備えており、コアあたりの GB/秒 速度がさまざまにあります。 Lz4 は Zstandard (zstd) と連携して、双方のアルゴリズムを用いて、より早く圧縮することができます。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 4.2 MB

8.9.1. Lz4 のインストール

パッケージをコンパイルします。

```
make BUILD_STATIC=no PREFIX=/usr
```

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 BUILD_STATIC=no PREFIX=/usr install
```

8.9.2. Lz4 の構成

インストールプログラム: lz4, lz4c (lz4 へのリンク), lz4cat (lz4 へのリンク), unlz4 (lz4 へのリンク)
インストールライブラリ: liblz4.so

概略説明

lz4 LZ4 フォーマットを使ってファイルの圧縮、伸長を行います。

lz4c LZ4 フォーマットを使ってファイルの圧縮を行います。

lz4cat LZ4 フォーマットにより圧縮されたファイルの内容一覧を表示します。

unlz4 LZ4 フォーマットを使ってファイルの伸長を行います。

liblz4 LZ4 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を提供します。

8.10. Zstd-1.5.7

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: 0.4 SBU
必要ディスク容量: 86 MB

8.10.1. Zstd のインストール

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 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
```

8.10.2. 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。

8.11. File-5.46

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 以下
必要ディスク容量: 19 MB

8.11.1. 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
```

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
```

8.11.2. 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。

8.12. Readline-8.3

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 以下
必要ディスク容量: 17 MB

8.12.1. 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
```

共有ライブラリに対して、ライブラリ検索パス（rpath）がハードコーディングされないようにします。 本パッケージは標準的なディレクトリにインストールするため rpath を必要ありません。 rpath は時に思わぬ弊害やセキュリティ問題を引き起こす場合があります。

```
sed -i 's/-Wl,-rpath,[^ ]*//' support/shobj-conf
```

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-8.3
```

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、個別の 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。 詳しくは本パッケージの README ファイルにある「Shared Libraries」という節を参照してください。

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
```

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-8.3
```

8.12.2. 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-8.3

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。

8.13. Pcre2-10.46

pcre2 パッケージは、次世代の Perl 互換正規表現 (Perl Compatible Regular Expression) ライブラリを提供します。

概算ビルド時間: 0.5 SBU
必要ディスク容量: 20 MB

8.13.1. Pcre2 のインストール

pcre2 をコンパイルするための準備をします。

```
./configure --prefix=/usr                        \
            --docdir=/usr/share/doc/pcre2-10.46 \
            --enable-unicode                     \
            --enable-jit                         \
            --enable-pcre2-16                    \
            --enable-pcre2-32                    \
            --enable-pcre2grep-libz              \
            --enable-pcre2grep-libbz2            \
            --enable-pcre2test-libreadline       \
            --disable-static
```

configure オプションの意味

--enable-unicode
このオプションは Unicode サポートを有効にして、このライブラリ内に UTF-8/16/32 文字列を扱う関数を含めるものです。

--enable-jit
このオプションはジャストインタイムコンパイルを有効にします。 これによりパターンマッチングが高速になります。

--enable-pcre2-16
このオプションは 16 ビット文字サポートを有効にします。

--enable-pcre2-32
このオプションは 32 ビット文字サポートを有効にします。

--enable-pcre2grep-libz
このオプションは pcregrep に対して .gz により圧縮されたファイルの読込機能を追加します。

--enable-pcre2grep-libbz2
このオプションは pcre2grep に対して .bz2 により圧縮されたファイルの読込機能を追加します。

--enable-pcre2test-libreadline
このオプションは pcre2test プログラムに対して行編集および履歴機能を追加します。

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
```

8.13.2. Pcre2 の構成

インストールプログラム: pcre2grep, pcre2test
インストールライブラリ: libpcre2-8.so, libpcre2-16.so, libpcre2-32.so, libpcre2-posix.so

概略説明

pcre2grep　grep と同等であり Perl 互換の正規表現を扱います。

pcre2test　　Perl 互換の正規表現をテストします。

8.14. M4-1.4.20

M4 パッケージはマクロプロセッサーを提供します。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 60 MB

8.14.1. 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
```

8.14.2. 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ーとして用いることもできます。

8.15. Bc-7.0.3

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: 7.8 MB

8.15.1. 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 -std=c99' ./configure --prefix=/usr -G -O3 -r
```

configure オプションの意味

CC='gcc -std=c99'
このパラメーターはコンパイラーと C 標準を指定します。

-G
bc がまだインストールされていない状態では動作しないテストスイートがあるため、それを省略します。

-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
```

8.15.2. 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。

8.16. 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: 0.1 SBU
必要ディスク容量: 33 MB

8.16.1. 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
```

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
```

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。 なお lex に対する man ページもシンボリックリンクとして生成します。

```
ln -sv flex   /usr/bin/lex
ln -sv flex.1 /usr/share/man/man1/lex.1
```

8.16.2. 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。

8.17. Tcl-8.6.17

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（発音は ”tickle”）によって書かれています。

概算ビルド時間: 3.0 SBU
必要ディスク容量: 91 MB

8.17.1. 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。

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 \
            --disable-rpath
```

configure パラメーターの意味

--disable-rpath
このパラメーターはバイナリ実行ファイルや共有ライブラリにおいて、ライブラリ検索パス（rpath）がハードコーディングされないようにします。 本パッケージは標準的なディレクトリにインストールするため rpath を必要ありません。 rpath は時に思わぬ弊害やセキュリティ問題を引き起こす場合があります。

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|$SRCDIR/unix/pkgs/tdbc1.1.12|/usr/lib/tdbc1.1.12|" \
    -e "s|$SRCDIR/pkgs/tdbc1.1.12/generic|/usr/include|"     \
    -e "s|$SRCDIR/pkgs/tdbc1.1.12/library|/usr/lib/tcl8.6|"  \
    -e "s|$SRCDIR/pkgs/tdbc1.1.12|/usr/include|"             \
    -i pkgs/tdbc1.1.12/tdbcConfig.sh

sed -e "s|$SRCDIR/unix/pkgs/itcl4.3.4|/usr/lib/itcl4.3.4|" \
    -e "s|$SRCDIR/pkgs/itcl4.3.4/generic|/usr/include|"    \
    -e "s|$SRCDIR/pkgs/itcl4.3.4|/usr/include|"            \
    -i pkgs/itcl4.3.4/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。

```
LC_ALL=C.UTF-8 make test
```

パッケージをインストールします。

```
make install
chmod 644 /usr/lib/libtclstub8.6.a
```

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

```
chmod -v u+w /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
```

任意の作業として、 以下のコマンドを実行してインストールします。

```
cd ..
tar -xf ../tcl8.6.17-html.tar.gz --strip-components=1
mkdir -v -p /usr/share/doc/tcl-8.6.17
cp -v -r  ./html/* /usr/share/doc/tcl-8.6.17
```

8.17.2. 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

8.18. 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

8.18.1. Expect のインストール

Expect は PTY が動作していることを必要とします。 chroot 環境内において PTY が適切に動作しているかどうかを、以下の単純なテストにより確認します。

```
python3 -c 'from pty import spawn; spawn(["echo", "ok"])'
```

上のコマンドの出力は ok となるべきものです。 そうならずに OSError: out of pty devices となったら、その環境は PTY 操作を適切に行うような設定が行われていないということです。 その場合は chroot から抜け出て、再度「仮想カーネルファイルシステムの準備」 を読み返して、devpts ファイルシステム（および他の仮想カーネルファイルシステム）を適切にマウントしてください。 「Chroot 環境への移行」 に従って chroot 環境に再度入ってください。 このメッセージは、先に進む前に解消しておくことが必要です。 そうでないと Expect を必要とするテストスイート（たとえば Bash, Binutils, GCC, GDBM そして Expect 自身のテストスイート）が大失敗し、些末な不備ならいくらでも発生してしまいます。

パッケージが gcc-15.1 以降に対応するための変更を加えます。

```
patch -Np1 -i ../expect-5.45.4-gcc15-1.patch
```

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         \
            --disable-rpath         \
            --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
```

8.18.2. 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。

8.19. 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

8.19.1. 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/dejagnu.html ../doc/dejagnu.texi
makeinfo --plaintext       -o doc/dejagnu.txt  ../doc/dejagnu.texi
```

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

8.19.2. DejaGNU の構成

インストールプログラム: dejagnu, runtest

概略説明

dejagnu 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。

8.20. Pkgconf-2.5.1

pkgconf パッケージは pkg-config の後継となるものです。 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: 0.1 SBU 以下
必要ディスク容量: 5.0 MB

8.20.1. Pkgconf のインストール

Pkgconf 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/pkgconf-2.5.1
```

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
```

元の Pkg-config との互換性を維持するため、以下の 2 つのシンボリックリンクを生成します。

```
ln -sv pkgconf   /usr/bin/pkg-config
ln -sv pkgconf.1 /usr/share/man/man1/pkg-config.1
```

8.20.2. Pkgconf の構成

インストールプログラム: pkgconf, pkg-config (pkgconf へのリンク), bomtool
インストールライブラリ: libpkgconf.so
インストールディレクトリ: /usr/share/doc/pkgconf-2.5.1

概略説明

pkgconf 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。

bomtool pkg-config の .pc ファイルから Software Bill Of Materials を生成します。

libpkgconf pkgconf の機能をほぼ提供します。 たとえば IDE のような他ツールやコンパイラーがそのフレームワークを利用できるようにします。

8.21. Binutils-2.45

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: 1.6 SBU
必要ディスク容量: 832 MB

8.21.1. Binutils のインストール

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-ld=default \
             --enable-plugins    \
             --enable-shared     \
             --disable-werror    \
             --enable-64-bit-bfd \
             --enable-new-dtags  \
             --with-system-zlib  \
             --enable-default-hash-style=gnu
```

configure パラメーターの意味

--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。

--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。

ビルド結果をテストします。

```
make -k check
```

失敗したテストの一覧は、以下を実行すれば得られます。

```
grep '^FAIL:' $(find -name '*.log')
```

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などのファイルを削除します。

```
rm -rfv /usr/lib/lib{bfd,ctf,ctf-nobfd,gprofng,opcodes,sframe}.a \
        /usr/share/doc/gprofng/
```

8.21.2. 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, 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, libgprofng.so, libopcodes.so, libsframe.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ー。
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.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ギン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。
libgprofng	gprofng によって利用される処理ルーチンをほぼ含むライブラリ。
libopcodes	opcodes（オペレーションコード; プロセッサー命令を「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。
libsframe	simple unwinder を使って、オンラインバックトレースをサポートするライブラリ。

8.22. GMP-6.3.0

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: 0.3 SBU
必要ディスク容量: 54 MB

8.22.1. 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ー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ー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、configure コマンドに --host=none-linux-gnu オプションを加えます。

はじめに gcc-15 以上への互換性を調整します。

```
sed -i '/long long t1;/,+1s/()/(...)/' configure
```

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-6.3.0
```

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-6.3.0
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ー向けに高度に最適化されます。 このためプロセッサー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 --host=none-linux-gnu をつける必要があります。

最低でも 199 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
```

8.22.2. 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-6.3.0

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。

8.23. MPFR-4.2.2

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: 0.2 SBU
必要ディスク容量: 43 MB

8.23.1. 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-4.2.2
```

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。

198 個のテストすべて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
```

8.23.2. 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-4.2.2

概略説明

`libmpfr` 倍精度演算の関数を提供します。

8.24. MPC-1.3.1

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: 0.1 SBU
必要ディスク容量: 22 MB

8.24.1. 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-1.3.1
```

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
```

ビルド結果をテストする場合は以下を実行します。

```
make check
```

パッケージとドキュメントをインストールします。

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

8.24.2. 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-1.3.1

概略説明

libmpc 複素数による演算関数を提供します。

8.25. Attr-2.5.2

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

8.25.1. 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-2.5.2
```

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
```

8.25.2. 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-2.5.2

概略説明

attr 拡張属性を制御するライブラリ関数を提供します。
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。

8.26. Acl-2.3.2

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 以下
必要ディスク容量: 6.5 MB

8.26.1. 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-2.3.2
```

パッケージをコンパイルします。

```
make
```

Acl のテストは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。 ビルド結果をテストする場合は以下を実行します。

```
make check
```

test/cp.test というテストが 1 つだけ失敗します。 これは Coreutils が Acl サポートつきでまだビルドできていないためです。

パッケージをインストールします。

```
make install
```

8.26.2. 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-2.3.2

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。

8.27. Libcap-2.76

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: 3.1 MB

8.27.1. 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
```

8.27.2. 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 ライブラリに関しての 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。

8.28. Libxcrypt-4.4.38

Libxcrypt パッケージは、パスワードに対する一方向のハッシュ処理を行う最新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

8.28.1. Libxcrypt のインストール

Libxcrypt をコンパイルするための準備をします。

```
./configure --prefix=/usr                \
            --enable-hashes=strong,glibc \
            --enable-obsolete-api=no     \
            --disable-static             \
            --disable-failure-tokens
```

configure オプションの意味

--enable-hashes=strong,glibc
安全なユースケースに対して推奨される強力なハッシュアルゴリズムを用いてビルドを行います。 このハッシュアルゴリズムは Glibc による従来の `libcrypt` と互換性があります。

--enable-obsolete-api=no
古い API 関数を無効にします。 最新の Linux システムをソースからビルドする際には不要なものです。

--disable-failure-tokens
failure token 機能を無効にします。 これは、特定プラットフォームにおいて、古くからあるハッシュライブラリとの互換性を保つために必要となります。 ただし Glibc ベースの Linux システムでは不要なものです。 it.

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

上に示した手順では、古い API 関数を無効にしました。 このようにしても、ソースからコンパイルしてインストールしたパッケージ類は、実行時にそのライブラリにリンクされるものは一つもありません。 ただし、バイナリでのみ提供されている特定のアプリケーションが、その関数ライブラリへのリンクを行い、そこでは ABI バージョン 1 を必要としています。 そういったバイナリのみで提供されているアプリケーションの利用においてその関数を必要とするか、あるいは LSB への準拠を必要とする場合には、以下のコマンドを使って本パッケージをもう一度ビルドしてください。

```
make distclean
./configure --prefix=/usr                \
            --enable-hashes=strong,glibc \
            --enable-obsolete-api=glibc  \
            --disable-static             \
            --disable-failure-tokens
make
cp -av --remove-destination .libs/libcrypt.so.1* /usr/lib
```

8.28.2. Libxcrypt の構成

インストールライブラリ: libcrypt.so

概略説明

`libcrypt` パスワードをハッシュする関数を提供します。

8.29. Shadow-4.18.0

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: 0.1 SBU
必要ディスク容量: 115 MB

8.29.1. Shadow のインストール

重要

Linux-PAM をすでにインストールしている場合は、本ページではなく BLFS の 手順 に従って shadow のビルド（または再ビルドやアップグレード）を行う必要があります。

注記

もっと強力なパスワードを利用したい場合は、まずは Linux-PAM のインストールと設定 を行ってください。 そして PAM サポートつきの shadow のインストールと設定 を行ってください。 最後に libpwquality のインストールと、これを利用する PAM の設定 を行います。

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-6.15」 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 YESCRYPT 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。 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。

警告

`PATH` 変数に `/bin` や `/sbin` を含めると、BLFS パッケージのビルドに失敗することがあります。 したがって `.bashrc` ファイルをはじめ、どの設定ファイルでもその設定は行わないでください。

```
sed -e 's:#ENCRYPT_METHOD DES:ENCRYPT_METHOD YESCRYPT:' \
    -e 's:/var/spool/mail:/var/mail:'                   \
    -e '/PATH=/{s@/sbin:@@;s@/bin:@@}'                  \
    -i 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-{b,yes}crypt \
            --without-libbsd    \
            --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ます。 それがまだ存在していない場合には、インストールスクリプトが間違った場所に作り出してしまいます。

`--with-{b,yes}crypt`
これはシェルによって 2 つのスイッチ、つまり *`--with-bcrypt`* と *`--with-yescrypt`* に展開されます。 パスワードのハッシュ処理を行うための Bcrypt および Yescrypt アルゴリズムが Libxcrypt において実装されており、

本スイッチは shadow がそれを用いることを指示します。 このアルゴリズムは従来の SHA アルゴリズムに比べて、（特に GPU ベースの攻撃への耐性が高く）より安全性を有しています。

--with-group-name-max-length=32
ユーザー名の最大文字数は 32 です。 そこでグループ名の最大文字数も同様とします。

--without-libbsd
libbsd の readpassphrase 関数は LFS 内には無いため用いないようにします。 その代わりに内部にコピーされている分を用います。

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
```

8.29.2. 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、説明が必要です。 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追加でパラメーターを与えなければ、追加するユーザーのグループは新たな固有グループが生成されることになります。 この動作が不適当であれば useradd コマンドの実行時に *-g* パラメーターか *-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、各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
```

8.29.3. 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
```

8.29.4. 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, 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, /usr/include/shadow
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 指定した名前でグループを生成します。
groupdel 指定された名前のグループを削除します。
groupmems スーパーユーザー権限を持たなくても、自分自身のグループのメンバーリストを管理可能とします。
groupmod 指定されたグループの名前や GID を修正します。
grpck グループファイル /etc/group と /etc/gshadow の整合性を確認します。
grpconv 通常のグループファイルから Shadow グループファイルを生成、更新します。
grpunconv /etc/gshadow ファイルを元に /etc/group ファイルを更新し /etc/gshadow ファイルを削除します。
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 指定した名前で新たなユーザーを生成します。 あるいは新規ユーザーのデフォルトの情報を更新します。
userdel 指定されたユーザーアカウントを削除します。

usermod　指定されたユーザーのログイン名、UID (User Identification)、利用シェル、初期グループ、ホームディレクトリなどを変更します。

vigr　`/etc/group` ファイルあるいは `/etc/gshadow` ファイルを編集します。

vipw　`/etc/passwd` ファイルあるいは `/etc/shadow` ファイルを編集します。

`libsubid`　ユーザーに対するサブ ID 範囲を取り扱うライブラリ。

8.30. GCC-15.2.0

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: 46 SBU（テスト込み）
必要ディスク容量: 6.6 GB

8.30.1. 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++ \
             --enable-default-pie     \
             --enable-default-ssp     \
             --enable-host-pie        \
             --disable-multilib       \
             --disable-bootstrap      \
             --disable-fixincludes    \
             --with-system-zlib
```

GCC では 7 つのコンピューター言語をサポートしていますが、それらのほとんどが必要としている依存パッケージは、まだこの時点でインストールしていません。 GCC がサポートする他のコンピューター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。

ここで有効にするのは C と C++ のみです。 ビルド時間を節約する目的があり、また LFS や BLFS において GCC が他の言語を必要とするパッケージが存在しないからです。 GCC においてさらに言語を増やしてプログラムコンパイルを行いたい場合は *--enable-languages* オプションにそれぞれ以下を追加してください。 Cobol に対して **cobol**（32 ビット LFS システム上においてビルドすると GCC がエラーとなります）、 Fortran に対して **fortran**、 Go に対して **go**、 Objective C に対して **objc**、 Objective C++ に対して **obj-c++**、 Modula 2 に対して **m2**。 GCC ではさらに Ada と D をサポートします。 ただしそれをサポートするコードは Ada あるいは D そのものによって書かれています。 したがってこのサポートを含めてビルドするには、あらかじめ Ada あるいは D コンパイラーがインストールされていないければなりません。 ここではそのサポートを有効にすることはできません。

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 プログラムを使うことを 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。

--disable-bootstrap
GCC のビルドシステムでは、クロスコンパイラーをビルドする場合、あるいはクロスコンパイルを行っている場合を除くと、デフォルトでは 3 ステージにおいてブートストラップを行います。 ブートストラップ処理は堅牢性のためであり、特に GCC をより新しいバージョンにアップグレードする際に必要となります。 LFS ではブートストラップとはことなる方法をとっています（ツールチェーンの技術的情報 において説明しています）。 したがってビルドシステムが提供するブートストラップ処理を必要としません。 この処理を用いないことからビルド時間を大幅に軽減しています。 完璧な（LFS のビルド中ではない）LFS システム上において GCC をアップグレードする場合は、本オプションを取り除いてください。

--disable-fixincludes
デフォルトにおいて、GCC のインストール中に GCC が利用するシステムヘッダーが「固定される」場合があります。これは最近の Linux システムにおいては不要なことです。 また GCC のインストール後に何かのパッケージをインストールすることを考えると、潜在的な危険を生み出すことになります。 本スイッチは GCC がヘッダーファイルを「固定（fix)」 しないようにします。

--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。

注記

PIE（position independent executable; 位置独立実行形式）とは、メモリ上のどこであっても、実行プログラムをロードできるようにします。 PIE がない場合には ASLR（Address Space Layout Randomization; アドレス空間配置のランダム化）という技術が適用されますが、適用先は共有ライブラリのみであって実行ファイルには適用されません。 共有ライブラリに加えて実行ファイルに対しても、PIE と ASLR を有効にすれば、実行ファイル内にある機密コードやデータが、固定的なアドレスに存在することを前提とした攻撃を軽減できます。

SSP（Stack Smashing Protection）とは、パラメータースタックが破壊されないようにする技術です。 スタック破壊が起きると、たとえばサブルーチンから返されるアドレスが変化してしまいます。 そうなった場合には、危険なコード（プログラムや共有ライブラリに元からあるものや、攻撃者が何らかの方法によって挿入したもの）に制御が移ってしまうことにもなります。

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 -k check 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 CPU コア数を指定するものです。

コンパイルするコードパターンが極端に複雑な場合に GCC はより多くのスタック領域を必要とする場合があります。 ホストディストロのスタック制限が厳しいかもしれないため、それを予防する意味でスタックサイズのハード上限を無制限に設定します。 ホストシステムのほとんど（そして最終的な LFS システム）はデフォルトでハード上限は無制限としていますが、それを明示的に設定したところで何も問題はありません。 スタックサイズのソフト上限を変更する必要はありません。 これは GCC が自動的に設定するものであり、その値がハード上限を超えない限りは適切に設定してくれます。

```
ulimit -s -H unlimited
```

失敗するテストをいくつか削除します。

```
sed -e '/cpython/d' -i ../gcc/testsuite/gcc.dg/plugin/plugin.exp
```

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 -R 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/development/ 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。

pr90579.c に関係するテストは失敗します。

テスト失敗は回避することができません。 その中には特定のハードウェアに起因するものもあります。 上記の URL に示されている結果と大きく異なっていなかったら、問題はありませんので先に進んでください。

パッケージをインストールします。

```
make install
```

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です。 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)/15.2.0/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
```

各種パッケージは C コンパイラーとして cc を呼び出しているものが数多くあります。 GCC 2回め においては、シンボリックリンクとして cc をすでに生成しています。 同様にしてその man ページについてもシンボリックリンクとして生成することにします。

```
ln -sv gcc.1 /usr/share/man/man1/cc.1
```

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)/15.2.0/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(){}' | cc -x 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。）

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

上のコマンドの出力は以下のようになるはずです。

```
/usr/lib/gcc/x86_64-pc-linux-gnu/15.2.0/../../../../lib/Scrt1.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/15.2.0/../../../../lib/crti.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/15.2.0/../../../../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/15.2.0/include
 /usr/local/include
 /usr/lib/gcc/x86_64-pc-linux-gnu/15.2.0/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 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
```

8.30.2. 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}, libcc1.so, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libhwasan.{a,so}, libitm.{a,so}, liblsan.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so}, libstdc++exp.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-15.2.0

概略説明

c++	C++ コンパイラー

cc	C コンパイラー
cpp	C プリプロセッサー。 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 や同じようなディレクティブを展開します。
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。
libcc1	GDB が GCC を利用可能とするためのライブラリ。
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。
libhwasan	ハードウェアをアシストする Address Sanitizer ランタイムライブラリ。
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 Binnutils 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。 Glibc から同じルーチンが提供されているため、通常は用いられません。
libstdc++	標準 C++ ライブラリ
libstdc++exp	試験的な C++ Contract 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。

8.31. Ncurses-6.5-20250809

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: 0.2 SBU
必要ディスク容量: 46 MB

8.31.1. 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       \
            --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。

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.6.5 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、cp コマンドの *--remove-destination* オプション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。（ヘッダーファイル curses.h も 「Ncurses-6.5-20250809」 で行ったものと同様に、ワイドキャラクター ABI が確実に利用されるように修正されます。）

```
make DESTDIR=$PWD/dest install
sed -e 's/^#if.*XOPEN.*$/#if 1/' \
    -i dest/usr/include/curses.h
cp --remove-destination -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。（.so のリンクは、編集された curses.h がワイドキャラクターに対して常に用いられるようにするためだけのものです。）

```
for lib in ncurses form panel menu ; do
    ln -sfv lib${lib}w.so /usr/lib/lib${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 -lcurses を指定するものがあるため、これもビルド可能なものにします。

```
ln -sfv libncursesw.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。

```
cp -v -R doc -T /usr/share/doc/ncurses-6.5-20250809
```

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
```

8.31.2. Ncurses の構成

インストールプログラム: captoinfo (tic へのリンク), clear, infocmp, infotocap (tic へのリンク), 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: libcurses.so (シンボリックリンク), libform.so (シンボリックリンク), libformw.so, libmenu.so (シンボリックリンク), libmenuw.so, libncurses.so (シンボリックリンク), libncursesw.so, libncurses++w.so, libpanel.so (シンボリックリンク), and libpanelw.so,

インストールディレクトリ: /usr/share/tabset, /usr/share/terminfo, /usr/share/doc/ncurses-6.5-20250809

概略説明

captoinfo	termcap の記述を terminfo の記述に変換します。
clear	画面消去が可能ならこれを行います。
infocmp	terminfo の記述どうしを比較したり出力したりします。
infotocap	terminfo の記述を termcap の記述に変換します。
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ーです。 これはソース形式の terminfo ファイルをバイナリ形式に変換し、ncurses ライブラリ内の処理ルーチンが利用できるようにします。 terminfo ファイルは特定端末の特性に関する情報が記述されるものです。
toe	利用可能なすべての端末タイプを一覧表示します。 そこでは端末名と簡単な説明を示します。
tput	端末に依存する機能設定をシェルが利用できるようにします。 また端末のリセットや初期化、あるいは長い端末名称の表示も行います。
tset	端末の初期化に利用します。
libncursesw	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 make menuconfig の実行によって表示されるメニューがあります。
libncurses++w	本パッケージ内でのその他のライブラリに対応する C++ バインディングを提供します。
libformw	フォームを実装するための関数を提供します。

libmenuw	メニューを実装するための関数を提供します。
libpanelw	パネルを実装するための関数を提供します。

8.32. Sed-4.9

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: 0.3 SBU
必要ディスク容量: 30 MB

8.32.1. 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
```

ビルド結果をテストする場合は以下を実行します。

```
chown -R 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-4.9
install -m644 doc/sed.html /usr/share/doc/sed-4.9
```

8.32.2. 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-4.9

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。

8.33. Psmisc-23.7

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: 6.7 MB

8.33.1. 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
```

テストスイートを実施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
```

8.33.2. Psmisc の構成

インストールプログラム: fuser, killall, peekfd, prtstat, pslog, pstree, pstree.xll (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.xll 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。

8.34. Gettext-0.26

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: 2.1 SBU
必要ディスク容量: 395 MB

8.34.1. 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-0.26
```

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

8.34.2. Gettext の構成

インストールプログラム: autopoint, 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-0.26, /usr/share/gettext, /usr/share/gettext-0.26

概略説明

autopoint Gettext 標準のインフラストラクチャーファイル (infrastructure file) をソースパッケージ内にコピーします。
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す。
recode-sr-latin	セルビア語のテキストに対し、キリル文字からラテン文字にコード変換します。
xgettext	指定されたソースファイルから、翻訳対象となるメッセージ行を抽出して、翻訳テンプレートとして生成します。
libasprintf	autosprintf クラスを定義します。 これは C++ プログラムにて利用できる C 言語書式の出力ルーチンを生成するものです。 <string> 文字列と <iostream> ストリームを利用します。
libgettextlib	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します。 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します。 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。

8.35. 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: 2.1 SBU
必要ディスク容量: 63 MB

8.35.1. 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
```

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
```

8.35.2. 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。

8.36. Grep-3.12

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: 0.6 SBU
必要ディスク容量: 48 MB

8.36.1. Grep のインストール

各種パッケージのテストにおいて、egrep と fgrep を用いた際の警告が原因でテストが失敗するため、その警告を削除します。

```
sed -i "s/echo/#echo/" src/egrep.sh
```

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
```

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. 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。

8.37. Bash-5.3

Bash は Bourne-Again Shell を提供します。

概算ビルド時間: 1.5 SBU
必要ディスク容量: 56 MB

8.37.1. 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             \
            --without-bash-malloc     \
            --with-installed-readline \
            --docdir=/usr/share/doc/bash-5.3
```

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
```

テストスィートを実行しない場合は「パッケージをインストールします。」と書かれた箇所まで読み飛ばしてください。

テストを実施するにあたっては 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 -R 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。

```
LC_ALL=C.UTF-8 su -s /usr/bin/expect tester << "EOF"
set timeout -1
spawn make tests
expect eof
lassign [wait] _ _ _ value
exit $value
EOF
```

テストスイートでは diff を使って、テストスクリプトの出力結果と期待される出力結果との差異を調べています。 diff からの出力（先頭行に < と >）があれば、テストが失敗したことを表します。 ただしその差異は無視できる旨を示すメッセージがあれば問題ありません。 run-builtins というテストは特定のホストディストロにおいては失敗します。 その際には出力結果の 479 行めと480 行めが異なると示されます。 テストの中には zh_TW.BIG5 と ja_JP.SJIS というロケールを必要とするものがあるため、これらがインストールされていない場合には失敗します。

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
```

8.37.2. 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-5.3

概略説明

bash 広く活用されているコマンドインタープリター。 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ー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。

8.38. Libtool-2.5.4

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、一貫した移植性の高いインターフェースとして実現します。

概算ビルド時間: 0.6 SBU
必要ディスク容量: 44 MB

8.38.1. 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
```

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
```

テストスイートでしか使わない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
```

8.38.2. 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 動的ロードライブラリのオープンに伴うさまざまな複雑さを隠蔽します。

8.39. GDBM-1.26

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、キーデータペアの収容、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: 0.2 SBU 以下
必要ディスク容量: 13 MB

8.39.1. 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
```

8.39.2. 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

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

8.40. Gperf-3.3

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: 0.2 SBU
必要ディスク容量: 12 MB

8.40.1. 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-3.3
```

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
```

8.40.2. 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-3.3

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。

8.41. Expat-2.7.3

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: 14 MB

8.41.1. 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-2.7.3
```

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-2.7.3
```

8.41.2. 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-2.7.3

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。

8.42. Inetutils-2.6

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: 36 MB

8.42.1. Inetutils のインストール

gcc-14.1 以降を用いて本パッケージがビルドできるようにします。

```
sed -i 's/def HAVE_TERMCAP_TGETENT/ 1/' telnet/telnet.c
```

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しか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
```

ビルド結果をテストする場合は以下を実行します。

```
make check
```

libls.sh というテストが場合によっては失敗することがあります。

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
```

8.42.2. 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。

8.43. Less-679

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: 16 MB

8.43.1. 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
```

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
```

8.43.2. 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。

8.44. Perl-5.42.0

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: 1.3 SBU
必要ディスク容量: 257 MB

8.44.1. 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                                         \
             -D prefix=/usr                               \
             -D vendorprefix=/usr                         \
             -D privlib=/usr/lib/perl5/5.42/core_perl     \
             -D archlib=/usr/lib/perl5/5.42/core_perl     \
             -D sitelib=/usr/lib/perl5/5.42/site_perl     \
             -D sitearch=/usr/lib/perl5/5.42/site_perl    \
             -D vendorlib=/usr/lib/perl5/5.42/vendor_perl  \
             -D vendorarch=/usr/lib/perl5/5.42/vendor_perl \
             -D man1dir=/usr/share/man/man1               \
             -D man3dir=/usr/share/man/man3               \
             -D pager="/usr/bin/less -isR"                \
             -D useshrplib                                \
             -D usethreads
```

configure オプションの意味

-D pager="/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。

-D man1dir=/usr/share/man/man1 -D man3dir=/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 ページを生成しません。 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。

-D usethreads
スレッドサポートを含めて Perl をビルドします。

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。

```
TEST_JOBS=$(nproc) make test_harness
```

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
```

8.44.2. 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, perl5.42.0 (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 を作ることができます。
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。
perl5.42.0	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。

8.45. XML::Parser-2.47

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: 2.4 MB

8.45.1. 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
```

8.45.2. 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。

8.46. 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

8.46.1. 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。 ここで行っているのは '¥${' という記述の並びに対して、右ブレースの前にバックスラッシュを追加して '¥$¥{' 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
```

8.46.2. 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。

8.47. Autoconf-2.72

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 0.4 SBU）
必要ディスク容量: 25 MB

8.47.1. 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
```

8.47.2. 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 マクロプロセッサーに対するラッパー。

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 プリプロセッサーの条件ディレクティブの識別子を出力します。 可搬性を考慮した構築ができている場合は、本プログラムが configure スクリプトにおいて何をチェックするべきかを決定してくれます。 また autoscan によって生成された configure.in ファイルでの過不足を調整する働きもします。

8.48. Automake-1.18.1

Automake パッケージは Autoconf が利用する Makefile などを生成するプログラムを提供します。

概算ビルド時間: 0.1 SBU 以下（テスト込みで約 1.1 SBU）
必要ディスク容量: 123 MB

8.48.1. 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-1.18.1
```

パッケージをコンパイルします。

```
make
```

4 つの並行ビルドとすることにより、テストを速く進めることができます。 たとえ論理コア数がそれより少ない場合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場合は以下を実行します。

```
make -j$(($(nproc)>4?$(nproc):4)) check
```

$((...)) の部分は、利用したい論理コア数に書き換えてください。

パッケージをインストールします。

```
make install
```

8.48.2. Automake の構成

インストールプログラム: aclocal, aclocal-1.18 (aclocal へのハードリンク), automake, automake-1.18 (automake へのハードリンク)
インストールディレクトリ: /usr/share/aclocal-1.18, /usr/share/automake-1.18, /usr/share/doc/automake-1.18.1

概略説明

aclocal — configure.in ファイルの内容に基づいて aclocal.m4 ファイルを生成します。

aclocal-1.18 — aclocal へのハードリンク。

automake — Makefile.am ファイルから Makefile.in ファイルを自動生成するツール。 パッケージ内のすべての Makefile.in ファイルを作るには、このプログラムをトップディレクトリから実行します。 configure.in ファイルを調べて、適切な Makefile.am ファイルを検索します。 そして対応する Makefile.in ファイルを生成します。

automake-1.18 — automake へのハードリンク。

8.49. OpenSSL-3.5.4

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）ウェブブラウザーなどです。

概算ビルド時間: 1.9 SBU
必要ディスク容量: 1.1 GB

8.49.1. 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。

```
HARNESS_JOBS=$(nproc) make test
```

30-test_afalg.t というテストが 1 つだけ失敗します。 それはカーネルオプションの CONFIG_CRYPTO_USER_API_SKCIPHER が有効でない場合、あるいは CBC が実装された AES 機能を提供するオプション（たとえば CONFIG_CRYPTO_AES と CONFIG_CRYPTO_CBC との組み合わせや、CPU が AES-NI をサポートする際の CONFIG_CRYPTO_AES_NI_INTEL など）が一つもない場合です。 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-3.5.4
```

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-3.5.4
```

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。

8.49.2. 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-3.5.4

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。

openssl 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。 openssl(1) に示される数多くの関数を利用することができます。

libcrypto.so 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。

libssl.so トランスポート層セキュリティ (Transport Layer Security; TLFS 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 ssl(7) にあります。

8.50. Elfutils-0.193 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: 0.3 SBU
必要ディスク容量: 156 MB

8.50.1. Libelf のインストール

Libelf は elfutils-0.193 パッケージに含まれます。 ソース tarball として elfutils-0.193.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 -k check
```

dwarf_srclang_check と run-backtrace-native-core.sh という 2 つのテストは失敗します。

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
```

8.50.2. Libelf の構成

インストールライブラリ: libelf.so
インストールディレクトリ: /usr/include/elfutils

概略説明

libelf.so 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。

8.51. Libffi-3.5.2

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。

FFI は Foreign Function Interface を表します。 FFI は、1 つの言語で書かれたプログラムから、別の言語で書かれたプログラムを呼び出せるようにするものです。 特に Libffi は、Perl や Python のようなインタープリターや、C, C++ で書かれた共有ライブラリサブルーチン間のブリッジ機能を提供します。

概算ビルド時間: 1.7 SBU
必要ディスク容量: 10 MB

8.51.1. 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ー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、ホスト CPU とそのシステム上の CPU の 双方の 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。 双方の CPU に対しての適切な値が分からない場合は **--without-gcc-arch** を指定して、汎用ライブラリを生成してください。

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
```

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。

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
```

8.51.2. 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。

8.52. Sqlite-3500400

SQLite パッケージは トランザクション SQL データベースエンジンです。 特徴として、自己完結 (self-contained) していて、サーバーモジュールが不要、かつ設定が不要なものです。

概算ビルド時間: 0.4 SBU
必要ディスク容量: 71 MB

8.52.1. Sqlite のインストール

ドキュメントを伸長（解凍）します。

```
tar -xf ../sqlite-doc-3500400.tar.xz
```

Sqlite をコンパイルするための準備をします。

```
./configure --prefix=/usr    \
            --disable-static  \
            --enable-fts{4,5} \
            CPPFLAGS="-D SQLITE_ENABLE_COLUMN_METADATA=1 \
                      -D SQLITE_ENABLE_UNLOCK_NOTIFY=1   \
                      -D SQLITE_ENABLE_DBSTAT_VTAB=1     \
                      -D SQLITE_SECURE_DELETE=1"
```

make オプションの意味

--enable-fts{4,5}
このスイッチは完全文字列検索（full text search; FTS）拡張機能のバージョン 4 と 5 を有効にします。

CPPFLAGS="-D SQLITE_ENABLE_COLUMN_METADATA=1 ...
アプリケーションの中には本オプションを有効にしておく必要のあるものがあります。 これを行うため、CFLAGS または CPPFLAGS の中に含めることにします。 後者まで利用するのは、CFLAGS のデフォルト値（あるいはユーザーが明示した値）では適用されないためです。 詳細は https://www.sqlite.org/compile.html を参照してください。

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
```

必要であればドキュメントをインストールします。

```
install -v -m755 -d /usr/share/doc/sqlite-3.50.4
cp -v -R sqlite-doc-3500400/* /usr/share/doc/sqlite-3.50.4
```

8.52.2. Sqlite の構成

インストールプログラム: sqlite3
インストールライブラリ: libsqlite3.so
インストールディレクトリ: /usr/share/doc/sqlite-3.50.4

概略説明

sqlite3 端末操作により SQLite ライブラリへアクセスするフロントエンドです。 クエリーを対話的に実行しその結果を得ることができます。

libsqlite3.so SQLite API 関数を提供します。

8.53. Python-3.13.7

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。 Python はインタープリター言語です。

概算ビルド時間: 2.0 SBU
必要ディスク容量: 453 MB

8.53.1. 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    \
            --enable-optimizations \
            --without-static-libpython
```

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。

--enable-optimizations
本スイッチは、拡張的ではあるものの高くつく最適化を有効にします。 インタープリターは二度ビルドされます。 そこでは 1 回めのビルドにて実施されるテストを用いて、最適化された最終バージョンが適正化されます。

パッケージをコンパイルします。

```
make
```

テストの中には不安定でありハングするものがあります。 そういったものをテストする場合は、各テストケースにおいて 2 分以内の制限をかけてテストスイートを実行してください。

```
make test TESTOPTS="--timeout 120"
```

比較的遅いシステムの場合は、その時間制限を増やせば 1 SBU（1 コアを使った Binutils 1 回目のビルド時間）で処理できるはずです。 テストの中には一風変わったものがあって、自動的に再実行された上で失敗するものがあります。 一度失敗して再実行の際に成功したものは、テストが成功したものとみなすことができます。 test_ssl というテストが chroot 環境内では失敗します。

パッケージをインストールします。

```
make install
```

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と、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てい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、（最新版の入手可能時にはいつでも）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、pip3 の最新版を求める警告は無視してください。 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。 ここでは設定ファイルを生成します。

```
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に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、すでにインストールされているモジュールを自動的に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-3.13.7/html

tar --strip-components=1  \
    --no-same-owner       \
    --no-same-permissions \
    -C /usr/share/doc/python-3.13.7/html \
    -xvf ../python-3.13.7-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います。

8.53.2. 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: libpython3.13.so, libpython3.so
インストールディレクトリ: /usr/include/python3.13, /usr/lib/python3, /usr/share/doc/python-3.13.7

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 Python インタープリターであり、対話的なオブジェクト指向プログラミング言語。

8.54. Flit-Core-3.12.0

Flit-core は Flit（簡単な Python モジュール向けパッケージングツール）の配布物ビルド部分です。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 1.3 MB

8.54.1. Flit-Core のインストール

パッケージをビルドします。

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

パッケージをインストールします。

```
pip3 install --no-index --find-links dist flit_core
```

pip3 の configure オプションとコマンドの意味

wheel
このコマンドは、本パッケージ向けの wheel アーカイブを生成します。

-w dist
生成した wheel を dist ディレクトリに置くことを指示します。

--no-cache-dir
生成された wheel を /root/.cache/pip ディレクトリにコピーしないようにします。

install
このコマンドはパッケージをインストールします。

--no-build-isolation, --no-deps, --no-index
これらのオプションは、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。

--find-links dist
dist ディレクトリから wheel アーカイブを検索することを指示します。

8.54.2. Flit-Core の構成

インストールディレクトリ: /usr/lib/python3.13/site-packages/flit_core, /usr/lib/python3.13/site-packages/flit_core-3.12.0.dist-info

8.55. Packaging-25.0

packaging モジュールは、相互運用性を実装したユーティリティーを提供する Python ライブラリです。 これを通じて明確で唯一の動作を規定（PEP440）したり、共有的な実装（PEP425）を採用することでそのメリットを最大限活用したりします。 ここには、バージョンの取り扱い、識別子、マーカー、タグなどを取り扱うユーティリティーが含まれます。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 3.3 MB

8.55.1. Packaging のインストール

以下のコマンドを実行して packaging をコンパイルします。

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

以下のコマンドを実行して packaging をインストールします。

```
pip3 install --no-index --find-links dist packaging
```

8.55.2. Packaging の構成

インストールディレクトリ: /usr/lib/python3.13/site-packages/packaging, /usr/lib/python3.13/site-packages/packaging-25.0.dist-info

8.56. Wheel-0.46.1

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: 608 KB

8.56.1. Wheel のインストール

以下のコマンドを実行して Wheel をコンパイルします。

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

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 --find-links dist wheel
```

8.56.2. 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/python3.13/site-packages/wheel, /usr/lib/python3.13/site-packages/wheel-0.46.1.dist-info

概略説明

wheel　wheel アーカイブの解凍、圧縮、変換を行うユーティリティーです。

8.57. Setuptools-80.9.0

Setuptools は Python パッケージに対してのダウンロード、ビルド、インストール、アンインストール、アップグレードを行うツールです。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 25 MB

8.57.1. Setuptools のインストール

パッケージをコンパイルします。

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

パッケージをインストールします。

```
pip3 install --no-index --find-links dist setuptools
```

8.57.2. Setuptools の構成

インストールディレクトリ: /usr/lib/python3.13/site-packages/_distutils_hack, /usr/lib/python3.13/site-packages/pkg_resources, /usr/lib/python3.13/site-packages/setuptools, /usr/lib/python3.13/site-packages/setuptools-80.9.0.dist-info

8.58. Ninja-1.13.1

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: 0.2 SBU
必要ディスク容量: 43 MB

8.58.1. Ninja のインストール

ninja は、可能な限り最大数のプロセスを使って並行処理により実行します。 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。 ninja を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、以下のようにストリームエディターを実行して、ninja が環境変数 NINJAJOBS を認識するようにします。

```
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 --verbose
```

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。

--verbose
本パラメーターは、configure.py の実行にあたって Ninja のビルドプロセスを表示させます。

本パッケージのテストは chroot 環境のもとでは実行することができません。 実行するには cmake が必要です。 ただし（*--bootstrap* オプションを使って）もう一度ビルドを行えば、本パッケージの基本的な関数についてはテストが実施されます。

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
```

8.58.2. 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。

8.59. Meson-1.9.1

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: 45 MB

8.59.1. 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-cache-dir --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。

8.59.2. 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/python3.13/site-packages/meson-1.9.1.dist-info, /usr/lib/python3.13/site-packages/mesonbuild

概略説明

meson 生産性の高いビルドシステム。

8.60. Kmod-34.2

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: 6.7 MB

8.60.1. 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。

```
mkdir -p build
cd       build

meson setup --prefix=/usr ..    \
            --buildtype=release \
            -D manpages=false
```

configure オプションの意味

-D manpages=false
このオプションは、man ページ生成において外部プログラムを必要とするため、これ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。

```
ninja
```

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。

パッケージをインストールします。

```
ninja install
```

8.60.2. 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, lsmod (kmod へのリンク), modinfo (kmod へのリンク), modprobe (kmod へのリンク), rmmod (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 稼動中のカーネルに対してロード可能なモジュールをインストールします。
kmod カーネルモジュールのロード、アンロードを行います。
lsmod その時点でロードされているモジュールを一覧表示します。
modinfo カーネルモジュールに関連付いたオブジェクトファイルを調べて、出来る限りの情報を表示します。
modprobe depmod によってモジュールの依存関係を記述したファイル (dependency file) が生成されます。 これを使って関連するモジュールを自動的にロードします。
rmmod 稼動中のカーネルからモジュールをアンロードします。
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。

8.61. Coreutils-9.8

Coreutils パッケージは、あらゆるオペレーティングシステムが必要とする基本的なユーティリティプログラムを提供します。

概算ビルド時間: 1.2 SBU
必要ディスク容量: 180 MB

8.61.1. 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-9.8-i18n-2.patch
```

注記

このパッチに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 -fv
automake -af
FORCE_UNSAFE_CONFIGURE=1 ./configure \
            --prefix=/usr            \
            --enable-no-install-program=kill,uptime
```

コマンドと configure オプションの意味

autoreconf -fv
国際化対応を行うパッチによって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。 通常なら *-i* オプションを使って標準的な補助（auxilary）ファイルのアップデートを行うところですが、本パッケージに関してはそれが通用しません。 それは `configure.ac` が古い gettext バージョンを指定しているためです。

automake -af
automake の補助（auxilary）ファイルは、autoreconf において *-i* オプションを指定しなかったため更新されていません。 以下のコマンドによってこれを更新し、ビルドが失敗しないようにします。

`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、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。

```
groupadd -g 102 dummy -U tester
```

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 -R tester .
```

テストを実行します。（テストの実行は標準入力を **/dev/null** とします。 そうしておかないと、LFS をグラフィック端末上でビルドしている場合、あるいは SSH 上や GNU Screen 上でのセッションで実行している場合に、2 つのテストが失敗します。 この理由は標準入力がホストディストロにおいて PTY に接続されているからであり、その PTY のようなデバイスノードは、LFS の chroot 環境からはアクセスできないからです。）

```
su tester -c "PATH=$PATH make -k RUN_EXPENSIVE_TESTS=yes check" \
   < /dev/null
```

一時的に作成したグループを削除します。

```
groupdel dummy
```

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
```

8.61.2. 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, mktemp, mv, nice, nl, nohup, nproc, numfmt, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, realpath, rm, rmdir, runcon, seq, shalsum, 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日付、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	指定されたファイル名からディレクトリ名部分を取り出します。
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す。
pathchk	ファイル名が有効で移植可能であるかをチェックします。

pinky	軽量な finger クライアント。 指定されたユーザーに関する情報を表示します。
pr	ファイルを印刷するために、ページ番号を振りカラム整形を行います。
printenv	環境変数の内容を表示します。
printf	指定された引数を指定された書式で表示します。 C 言語の printf 関数に似ています。
ptx	指定されたファイル内のキーワードに対して整列済インデックス (permuted index) を生成します。
pwd	現在の作業ディレクトリ名を表示します。
readlink	指定されたシンボリックリンクの対象を表示します。
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。
shalsum	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。

8.62. Diffutils-3.12

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: 0.5 SBU
必要ディスク容量: 51 MB

8.62.1. 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
```

8.62.2. 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、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。

8.63. Gawk-5.3.2

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

8.63.1. 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
```

ビルド結果をテストする場合は以下を実行します。

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

パッケージをインストールします。

```
rm -f /usr/bin/gawk-5.3.2
make install
```

コマンドの意味

rm -f /usr/bin/gawk-5.3.2
: gawk-5.3.2 が存在している場合、ビルドシステムはハードリンクを再生成しません。 「Gawk-5.3.2」 においてインストールしたハードリンクをここで削除することにより、確実に再生成されるようにします。

インストール処理においては、awk が gawk のシンボリックリンクとして、すでに生成されています。 同様にしてその man ページについてもシンボリックリンクとして生成することにします。

```
ln -sv gawk.1 /usr/share/man/man1/awk.1
```

必要ならドキュメントをインストールします。

```
install -vDm644 doc/{awkforai.txt,*.{eps,pdf,jpg}} -t /usr/share/doc/gawk-5.3.2
```

8.63.2. Gawk の構成

インストールプログラム: awk (gawk へのリンク), gawk, gawk-5.3.2
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, rwarray.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-5.3.2

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-5.3.2 gawk へのハードリンク。

8.64. Findutils-4.10.0

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。）Findutils では xargs プログラムも提供します。 このプログラムは、検索された複数ファイルの個々に対して、指定されたコマンドを実行するために用いられます。

概算ビルド時間: 0.7 SBU
必要ディスク容量: 61 MB

8.64.1. 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
```

configure オプションの意味

--localstatedir
このオプションは 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
```

ビルド結果をテストする場合は以下を実行します。

```
chown -R 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
```

8.64.2. 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。

8.65. Groff-1.23.0

Groff パッケージはテキストやイメージ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: 0.2 SBU
必要ディスク容量: 108 MB

8.65.1. Groff のインストール

Groff はデフォルトの用紙サイズを設定する環境変数 PAGE を参照します。 米国のユーザーであれば *PAGE=letter* と設定するのが適当です。 その他のユーザーなら *PAGE=A4* とするのが良いかもしれません。 このデフォルト用紙サイズはコンパイルにあたって設定されます。 「A4」なり「letter」なりの値は **/etc/papersize** ファイルにて設定することも可能です。

Groff をコンパイルするための準備をします。

```
PAGE=<paper_size> ./configure --prefix=/usr
```

パッケージをビルドします。

```
make
```

ビルド結果をテストする場合は以下を実行します。

```
make check
```

パッケージをインストールします。

```
make install
```

8.65.2. 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-1.23.0, /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 プロセッサー。

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 プリプロセッサーであり groff ファイルへの perl コード追加を行います。

gpinyin groff プリプロセッサーであり groff ファイルへの Pinyin（北京語のローマ字つづり）追加を行います。

grap2graph grap プログラムファイルを、刈り込んだ（crop した）ビットマップイメージに変換します。 （grap は、ダイアグラムを生成する、かつての Unix プログラミング言語です。）

grn gremlin 図を表すファイルを処理するための groff プリプロセッサー。

grodvi TeX の出力ファイルである 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ーを呼び出します。

groffer	groff ファイルや man ページを X 上や TTY 端末上に表示します。
grog	入力ファイルを読み込んで、印刷時には groff コマンドオプションのどれが必要かを推定します。 コマンドオプションは -e、-man、-me、-mm、-ms、-p、-s のいずれかです。 そしてそのオプションを含んだ groff コマンドを表示します。
grolbp	Canon CAPSL プリンター（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 ドライバープログラム。
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 ドライバープログラム。
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	（標準入力が端末であれば）標準エラー出力にプロンプトを表示して、標準入力から複数のキーワードを含んだ一行を読み込みます。 そして指定されたファイルにて示される参考文献データベース内に、そのキーワードが含まれるかどうかを検索します。 キーワードが含まれるものを標準出力に出力します。入力がなくなるまでこれを繰り返します。
mmroff	groff 用の単純なプリプロセッサー。
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。
tfmtodit	コマンド groff -Tdvi を使ってフォントファイルを生成します。
troff	Unix の troff コマンドと高い互換性を持ちます。 通常は groff コマンドを用いて本コマンドが起動されます。 groff コマンドは、プリプロセッサー、ポストプロセッサーを、適切な順で適切なオプションをつけて起動します。

8.66. GRUB-2.12

GRUB パッケージは GRand Unified Bootloader を提供します。

概算ビルド時間: 0.3 SBU
必要ディスク容量: 166 MB

8.66.1. 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、UEFI サポートを含む GRUB（およびその依存パッケージ）をインストールする必要があります。 その場合は BLFS ページ の手順に従ってください。 このパッケージのインストールは省略できます。 あるいは BLFS ブックに示す UEFI パッケージのサポートを含む GRUB を競合することなくインストールすることもできます（BLFS ではどちらの状況に対しても、その手順を説明しています）。

警告

ビルドに影響を与える可能性のある環境変数をリセットします。

```
unset {C,CPP,CXX,LD}FLAGS
```

このパッケージをビルドする際に、独自のコンパイルフラグを使って「チューニング」することは止めてください。 このパッケージはブートローダーです。 ソースコード内には低レベル操作が用いられており、過激な最適化フラグによってはその機能を壊してしまうかもしれないためです。

リリース tarball に含まれていないファイルを追加します。

```
echo depends bli part_gpt > grub-core/extra_deps.lst
```

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を無効にし、一部のテストプログラムを実行しないようにした上で、ビルドを行います。

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。 また Bash completion サポートファイルを、その開発者が推奨するディレクトリに移動させます。

```
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。

8.66.2. 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 が初めに起動される時)

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	指定されたファイル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ー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単一ファイル (Apple マシン向け) 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。
grub-kbdcomp	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。
grub-macbless	HFS または HFS+ ファイルシステムに対する Mac 風の bless。 (bless は Apple マシン専用です。デバイスをブータブルにします。)
grub-menulst2cfg	GRUB Legacy の `menu.lst`を GRUB 2 にて利用される `grub.cfg` に変換します。
grub-mkconfig	`grub.cfg` ファイルを生成します。
grub-mkimage	GRUB のブートイメージ (bootable image) を生成します。
grub-mklayout	GRUB のキーボードレイアウトファイルを生成します。
grub-mknetdir	GRUB のネットブートディレクトリを生成します。
grub-mkpasswd-pbkdf2	ブートメニューにて利用する、PBKDF2 により暗号化されたパスワードを生成します。
grub-mkrelpath	システムのパスをルートからの相対パスとします。
grub-mkrescue	フロッピーディスク、CDROM/DVD、USB ドライブ向けの GRUB のブートイメージを生成します。
grub-mkstandalone	スタンドアロンイメージを生成します。
grub-ofpathname	GRUB デバイスのパスを出力するヘルパープログラム。
grub-probe	指定されたパスやデバイスに対するデバイス情報を検証 (probe) します。
grub-reboot	デフォルトのブートメニューを設定します。 これは次にブートした時だけ有効なものです。
grub-render-label	Apple Mac に対して Apple .disk_label を提供します。
grub-script-check	GRUB の設定スクリプトにおける文法をチェックします。
grub-set-default	デフォルトのブートメニューを設定します。
grub-sparc64-setup	grub-setup に対するヘルパープログラム。
grub-syslinux2cfg	syslinux の設定ファイルを grub.cfg フォーマットに変換します。

8.67. Gzip-1.14

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: 21 MB

8.67.1. 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
```

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
```

8.67.2. 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。

8.68. IPRoute2-6.16.0

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: 0.1 SBU
必要ディスク容量: 17 MB

8.68.1. 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。

```
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。

```
install -vDm644 COPYING README* -t /usr/share/doc/iproute2-6.16.0
```

8.68.2. IPRoute2 の構成

インストールプログラム: bridge, ctstat (lnstat へのリンク), genl, ifstat, ip, lnstat, nstat, routel, rtacct, rtmon, rtpr, rtstat (lnstat へのリンク), ss, tc
インストールディレクトリ: /etc/iproute2, /usr/lib/tc, /usr/share/doc/iproute2-6.16.0

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。

ifstat インターフェースの統計情報を表示します。 インターフェースによって送受信されたパケット量が示されます。

ip 主となる実行モジュールで、複数の機能性を持ちます。 以下のようなものです。
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。

routel ip route のコンポーネント。 これはルーティングテーブルの一覧を表示します。

rtacct /proc/net/rt_acct の内容を表示します。

rtmon ルート監視ユーティリティー。

rtpr　　ip -o コマンドにより出力される内容を読みやすい形に戻します。

rtstat　ルートステータスの表示ユーティリティー。

ss　　netstat コマンドと同じ。 アクティブな接続を表示します。

tc　　QoS (Quality Of Service) と CoS (Class Of Service) を実装するトラフィック制御です。
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 filter は、QOS/COS パケットのフィルタリング設定を行います。
tc monitor は、カーネル内のトラフィック制御に対して行われた変更を参照するために用いられます。

8.69. Kbd-2.9.0

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: 43 MB

8.69.1. 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-2.9.0-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 ユーティリティーをビルドしないようにします。 そのユーティリティーは 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
```

本パッケージのテストは chroot 環境では失敗します。 これは valgrind を必要としているためです。 さらに valgrind を完全に含んだシステムであっても、グラフィック環境下で失敗するテストが複数あります。 そういったテストはグラフィックではない環境において成功します。

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。

```
cp -R -v docs/doc -T /usr/share/doc/kbd-2.9.0
```

8.69.2. Kbd の構成

インストールプログラム: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbdinfo, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (psfxtable へのリンク), psfgettable (psfxtable へのリンク), psfstriptable (psfxtable へのリンク), psfxtable, 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-2.9.0, /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しての情報を取得します。
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ーにロードするために利用します。 現在では setfont を利用します。
openvt	新しい仮想端末（virtual terminal; VT）上でプログラムを起動します。
psfaddtable	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。
psfstriptable	コンソールフォントから埋め込められた 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。

8.70. Libpipeline-1.5.8

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

8.70.1. 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
```

テストには Check ライブラリが必要ですが、これは LFS から削除しました。

パッケージをインストールします。

```
make install
```

8.70.2. 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。

8.71. Make-4.4.1

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: 0.7 SBU
必要ディスク容量: 13 MB

8.71.1. 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
```

ビルド結果をテストする場合は以下を実行します。

```
chown -R 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
```

8.71.2. 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。

8.72. Patch-2.8

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: 13 MB

8.72.1. 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
```

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
```

8.72.2. 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。

8.73. Tar-1.35

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: 0.6 SBU
必要ディスク容量: 43 MB

8.73.1. 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こととされますが、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
```

ビルド結果をテストする場合は以下を実行します。

```
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またはセキュリティラベル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-1.35
```

8.73.2. 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-1.35

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。

8.74. Texinfo-7.2

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: 0.4 SBU
必要ディスク容量: 160 MB

8.74.1. Texinfo のインストール

Perl-5.42 およびそれ以降において警告となるコードパターンを修正します。

```
sed 's/! $output_file eq/$output_file ne/' -i tp/Texinfo/Convert/*.pm
```

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
```

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 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
```

8.74.2. 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。

8.75. Vim-9.1.1806

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: 3.7 SBU
必要ディスク容量: 259 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/systemd/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。

8.75.1. 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
```

ビルド結果をテストする場合は、tester ユーザーがソースツリーに書き込みできるようにし、また curl と wget を必要とする 1 つのテストを除外するようにします。

```
chown -R tester .
sed '/test_plugin_glvs/d' -i src/testdir/Make_all.mak
```

tester ユーザーによりテストを実行します。

```
su tester -c "TERM=xterm-256color 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上に出力します。 これは端末画面の設定によっては問題を引き起こします。（特にテストスイートの要請を満たすため TERM 変数を上書きしている場合などです。）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。

test_matchfuzzy.vim というテストに失敗するシステムがあります。

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、いわば反射的に 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-9.1.1806 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/vim91/doc /usr/share/doc/vim-9.1.1806
```

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/systemd/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。

8.75.2. 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て行っておくことが必要です。 このモード設定は他の設定を置き換えるものとなることから、まず初めに行っておかなければならないものだからです。 以下のコマンドを実行して vim の設定ファイルを生成します。

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

" 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 と設定しておくと vi 互換モードでの動作に比べて有用な動作となります。（これがデフォルトになっています。）その設定の記述から「no」の文字を取り除けば、旧来の vi コマンドの動作となります。 *set backspace=2* を設定しておくと、行を超えてもバックスペースキーによる編集が可能となります。 またインデントが自動的に行われ、コマンド起動時には自動的に挿入モードとなります。 *syntax on* パラメーターを指定すれば 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チェックファイル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チェックファイルをインストールするなら 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 /usr/share/vim/vim91/spell/ にコピーしてください。

スペルチェック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 runtime/spell/README.txt を参照してください。

8.75.3. Vim の構成

インストールプログラム: ex (vim へのリンク), rview (vim へのリンク), rvim (vim へのリンク), vi (vim へのリンク), view (vim へのリンク), vim, vimdiff (vim へのリンク), vimtutor, xxd
インストールディレクトリ: /usr/share/vim

概略説明

ex vim を ex モードで起動します。

rview	view の機能限定版。 シェルは起動できず、サスペンドも行うことはできません。
rvim	vim の機能限定版。 シェルは起動できず、サスペンドも行うことはできません。
vi	vim へのリンク。
view	vim を読み込み専用モード（read-only mode）で起動します。
vim	エディター。
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。

8.76. MarkupSafe-3.0.3

MarkupSafe は、XML/HTML/XHTML マークアップセーフな文字列を実装する Python モジュールです。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 500 KB

8.76.1. MarkupSafe のインストール

以下のコマンドを実行して MarkupSafe をコンパイルします。

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

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。

```
pip3 install --no-index --find-links dist Markupsafe
```

8.76.2. MarkupSafe の構成

インストールディレクトリ: /usr/lib/python3.13/site-packages/MarkupSafe-3.0.3.dist-info

8.77. Jinja2-3.1.6

Jinja2 は、Python の簡単なテンプレート言語を実装する Python モジュールです。

概算ビルド時間: 0.1 SBU 以下
必要ディスク容量: 2.6 MB

8.77.1. Jinja2 のインストール

パッケージをビルドするために以下を実行します。

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

パッケージをインストールします。

```
pip3 install --no-index --find-links dist Jinja2
```

8.77.2. Jinja2 の構成

インストールディレクトリ: /usr/lib/python3.13/site-packages/Jinja2-3.1.6.dist-info

8.78. Systemd-258

systemd パッケージは、システムの起動、稼動、終了の制御を行うプログラムを提供します。

概算ビルド時間: 1.4 SBU
必要ディスク容量: 337 MB

8.78.1. systemd のインストール

デフォルトの udev ルールから、不要な 2 つのグループ render と sgx を削除します。

```
sed -e 's/GROUP="render"/GROUP="video"/' \
    -e 's/GROUP="sgx", //'               \
    -i rules.d/50-udev-default.rules.in
```

systemd をコンパイルするための準備をします。

```
mkdir -p build
cd       build

meson setup ..                \
      --prefix=/usr           \
      --buildtype=release     \
      -D default-dnssec=no    \
      -D firstboot=false      \
      -D install-tests=false  \
      -D ldconfig=false       \
      -D sysusers=false       \
      -D rpmmacrosdir=no      \
      -D homed=disabled       \
      -D man=disabled         \
      -D mode=release         \
      -D pamconfdir=no        \
      -D dev-kvm-mode=0660    \
      -D nobody-group=nogroup \
      -D sysupdate=disabled   \
      -D ukify=disabled       \
      -D docdir=/usr/share/doc/systemd-258
```

meson オプションの意味

--buildtype=release
本スイッチは、デフォルトのビルドタイプ（「debug」）をオーバーライドします。 そのままにしておくと、最適化されていない実行モジュールが生成されるためです。

-D default-dnssec=no
本スイッチは、実験的な DNSSEC サポートを無効にします。

-D firstboot=false
本スイッチは、systemd サービスを、システムの初回構築用としてインストールしないようにします。 LFS ではすべて手作業で行うため、この機能が必要ないからです。

-D install-tests=false
本スイッチはコンパイルされたテストをインストールしないようにします。

-D ldconfig=false
本スイッチは、システム起動時に ldconfig を実行するような systemd ユニットはインストールしないようにします。 LFS のようにソースから作り出すディストリビューションにとっては無用なものであり、起動時間も長くなります。 起動時の ldconfig 実行を有効にするには、本オプションを除いてください。

-D sysusers=false
本スイッチは、システム起動初期に /etc/group ファイルと /etc/passwd ファイルを設定する systemd サービスをインストールしないようにします。 この二つのファイルは前章にて生成済です。 LFS システム上におけるこのデーモンは、ユーザーアカウントを手動で生成するまでは、利用することはできません。

-D rpmmacrosdir=no
本スイッチは systemd において利用される RPM マクロをインストールしないようにします。 LFS では RPM をサポートしていないためです。

-D homed=disabled
LFS が取り扱う範囲にそぐわない依存関係を持ったデーモンを削除します。

-D man=disabled
man ページを生成することで発生する追加パッケージの導入を行わないようにします。 systemd の man ページは、生成済みの tarball を使ってインストールすることにします。

-D mode=release
アップストリームにおいて試験的機能とみなされている機能を無効にします。

-D pamconfdir=no
PAM 設定は LFS 上では機能しないため、これをインストールしないようにします。

-D dev-kvm-mode=0660
デフォルトの udev ルールは、あらゆるユーザーが /dev/kvm にアクセスできるようにします。 当編集者としてこれは危険なことと考えています。 本オプションはその設定を上書きします。

-D nobody-group=nogroup
nogroup がグループ GID 65534 であるグループ名として指定します。

-D sysupdate=disabled
systemd-sysupdate ツールをインストールしないようにします。 これはバイナリディストロを自動的に更新する目的のものです。 したがってソースからビルドするという Linux システムにおいては、基本的に無用なものです。 またこれが利用可能でありながら適切に設定されていない場合には、起動時にエラーが表示されることになります。

-D ukify=disabled
systemd-ukify スクリプトをインストールしないようにします。 このスクリプトは実行時に Python モジュール pefile を必要としますが、これは LFS と BLFS のいずれにおいても提供していません。

パッケージをコンパイルします。

```
ninja
```

テストの中には、あのシンプルな /etc/os-release ファイルを必要とするものがあります。 ビルド結果をテストする場合は以下を実行します。

```
echo 'NAME="Linux From Scratch"' > /etc/os-release
ninja test
```

systemd:core / test-namespace というテストが、LFS の chroot 環境内では失敗します。 また別のテストでも失敗するものがありますが、これはさまざまなカーネルオプションに依存しているためです。 systemd:test / test-copy という名前のテストは、多大な並行ジョブ数による I/O の複雑化によって失敗することがあります。 ただし単独で meson test test-copy を実行すれば成功するはずです。

パッケージをインストールします。

```
ninja install
```

man ページをインストールします。

```
tar -xf ../../systemd-man-pages-258.tar.xz \
    --no-same-owner --strip-components=1     \
    -C /usr/share/man
```

systemd-journald に対して必要となる /etc/machine-id ファイルを生成します。

```
systemd-machine-id-setup
```

基本的なターゲット構造を設定します。

```
systemctl preset-all
```

8.78.2. systemd の構成

インストールプログラム: bootctl, busctl, coredumpctl, halt (systemctl へのシンボリックリンク), hostnamectl, init, journalctl, kernel-install, localectl, loginctl, machinectl, mount.ddi (systemd-dissect へのシンボリックリンク), networkctl, oomctl, portablectl, poweroff (systemctl へのシンボリックリンク), reboot (systemctl へのシンボリックリンク), resolvconf (resolvectl へのシンボリックリンク), resolvectl, run0, runlevel (systemctl へのシンボリックリンク), shutdown (systemctl へのシンボリックリンク), systemctl, systemd-ac-power, systemd-analyze, systemd-ask-password, systemd-cat, systemd-cgls, systemd-cgtop, systemd-confext (systemd-sysext へのシンボリックリンク), systemd-creds, systemd-delta, systemd-detect-virt, systemd-dissect, systemd-escape, systemd-hwdb, systemd-id128, systemd-inhibit, systemd-machine-id-setup, systemd-mount, systemd-notify, systemd-nspawn, systemd-path, systemd-pty-forward, systemd-repart, systemd-resolve (resolvectl へのシンボリックリンク), systemd-run, systemd-socket-activate, systemd-stdio-bridge, systemd-sysext, systemd-tmpfiles, systemd-tty-ask-password-agent, systemd-vpick, systemd-umount (systemd-mount へのシンボリックリンク), timedatectl, udevadm, userdbctl, varlinkctl

インストールライブラリ: libnss_myhostname.so.2, libnss_mymachines.so.2, libnss_resolve.so.2, libnss_systemd.so.2, libsystemd.so, libsystemd-shared-258.so (/usr/lib/systemd ディレクトリ内), libudev.so

インストールディレクトリ: /etc/binfmt.d, /etc/init.d, /etc/kernel, /etc/modules-load.d, /etc/sysctl.d, /etc/systemd, /etc/tmpfiles.d, /etc/udev, /etc/xdg/systemd, /usr/include/systemd, /usr/lib/binfmt.d, /usr/lib/credstore, /usr/lib/environment.d, /usr/lib/kernel, /usr/lib/modprobe.d, /usr/lib/modules-load.d, /usr/lib/systemd, /usr/lib/udev, /usr/lib/sysctl.d, /usr/lib/systemd, /usr/lib/tmpfiles.d, /usr/share/doc/systemd-258, /usr/share/factory, /usr/share/systemd, /var/lib/systemd, /var/log/journal

概略説明

bootctl — EFI ファームウェアのブート設定の制御を行います。

busctl — D-Bus のバスを監視するために用います。

coredumpctl — systemd journal よりコアダンプを抽出します。

halt — 普通は shutdown にオプション *-h* をつけて実行します。 ただし既にランレベルが 0 である場合を除きます。 カーネルに対してシステムの停止を指示します。 システムが停止したことは /var/log/wtmp ファイルに記録されます。

hostnamectl — システムのホスト名および関連設定を確認し変更します。

init — カーネルがハードウェアを初期化した後に起動される最初のプロセスです。 init は、この後の起動処理を担い、設定ファイルに応じたブートプロセスと他の全てのプロセスを起動します。 つまり systemd を起動するということです。

journalctl — Systemd のジャーナルの内容を確認します。

kernel-install — カーネルや initramfs イメージを /boot ディレクトリに対して追加、削除します。

localectl — システムロケールやキーボードレイアウト設定を確認し変更します。

loginctl — Systemd のログインマネージャーの状態を確認し制御します。

machinectl — Systemd の仮想マシンとコンテナー登録マネージャー (Container Registration Manager) の状態を確認し制御します。

networkctl — systemd-netword から見えるネットワークリンクの状態を確認 (introspect) し設定します。

oomctl — systemd の Out Of Memory デーモンを制御します。

portablectl — ローカルシステムにおいてポータブルサービスのアタッチ、デタッチを行います。

poweroff — カーネルに対してシステム停止を指示し、コンピューターの電源を落とします。(halt参照)

reboot	カーネルに対してシステム再起動を指示します。(halt参照)
resolvconf	systemd-resolved に対する DNS サーバーやドメイン設定を登録します。
resolvectl	ネットワーク名前解決マネージャーに対して制御コマンドを送信します。 あるいはドメイン名、IPv4、IPv6 アドレス、DNS レコードやサービスなどを解決します。
run0	sudo と同様に一時的な権限昇格や別権限の取得を行います。
runlevel	現時点とその直前のランレベルを表示します。 最新のランレベルは /run/utmp ファイルに記録されます。
shutdown	すべてのプロセスとすべてのログインユーザーへの通知を行なった上で、システムを安全に停止します。
systemctl	Systemd システムとサービスマネージャーの状態について確認し制御します。
systemd-ac-power	システムが外部電源につながっているかどうかを報告します。
systemd-analyze	起動処理パフォーマンスを解析します。 また問題のある systemd ユニットを特定します。
systemd-ask-password	Linux コマンドラインから指定されたメッセージを用いて、システムパスワードやユーザーのパスフレーズを確認します。
systemd-cat	systemd journal に対してプロセスの STDOUT と STDERR に接続します。
systemd-cgls	指定された Linux コントロールグループ（control group）の階層を再帰的に表示します。
systemd-cgtop	最上位のローカル Linux コントロールグループ（control group）を表示し、CPU、メモリ、ディスクI/Oロードの並びにより示します。
systemd-creds	資格情報を表示し処理します。
systemd-delta	/etc ディレクトリにある設定ファイルを同定したり比較したりします。 この設定ファイルは /usr ディレクトリにあるデフォルト設定をオーバーライドします。
systemd-detect-virt	システムが仮想化環境で動作しているかどうかを検出し、それに応じて udev を調整します。
systemd-dissect	OS ディスクイメージの調査に用いられます。
systemd-escape	systemd ユニット名での文字エスケープを行います。
systemd-hwdb	ハードウェアデータベース（hwdb）を管理します。
systemd-id128	id128（UUID）文字列を生成し表示します。
systemd-inhibit	システム停止、休止、アイドル禁止ロックを行うプログラムを実行します。 プロセスが正常起動するまでは、システムシャットダウンのような処理は行いません。
systemd-machine-id-setup	システムインストールツールがマシンIDを初期化するために利用します。 このマシンIDは /etc/machine-id ファイル内にあるものから、インストール時にランダムに生成されます。
systemd-mount	ディスクの一時的あるいは自動マウントを行ないます。
systemd-notify	init システムに対してステータス変更が発生したことを通知するデーモンスクリプトが利用します。
systemd-nspawn	軽量な名前空間コンテナー（light-weight namepspace container）においてコマンドや OS 全体の実行に用いられます。
systemd-path	システムパスやユーザーパスを検索します。
systemd-pty-forward	カスタムの端末背景色あるいはタイトルを使ってコマンド実行を行います。
systemd-repart	systemd が OS イメージ内（たとえばコンテナーなど）で用いられている場合に、パーティションテーブルに対してパーティションの拡張や追加を行うために用いられます。
systemd-resolve	ドメイン名、IPV4 と IPv6 アドレス、DNSリソースレコード、サービスの名前解決を行います。
systemd-run	一時的な .service ユニットや .scope ユニットを生成および起動し、その指定コマンドを実行します。 これは systemd ユニットの検証を行うことができます。

systemd-socket-activate	ソケットデバイスの情報を読み取って、ソケットに対するコネクション上にてプロセスを起動します。
systemd-sysext	システム拡張イメージを有効にします。
systemd-tmpfiles	tmpfiles.d ディレクトリにて指定された設定ファイルの内容に基づいて、テンポラリファイルなどの生成削除等を行います。
systemd-umount	マウントポイントをアンマウントします。
systemd-tty-ask-password-agent	未定となっている Systemd のパスワード変更指示の一覧を表示し処理します。
systemd-vpick	".v/ バージョンディレクトリへのパスの解決に用いられます。
timedatectl	システムクロックとその設定を確認し変更します。
udevadm	汎用的な udev 管理ツール。 udevd デーモンの制御、udev データベースデータの提供、uevent の監視、uevent の完了までの待機、udev 設定のテスト、指定デバイスに対する uevent の起動、といったことを行います。
userdbctl	ユーザー、グループ、グループメンバーの確認に用いられます。
varlinkctl	Varlink サービスとの対話および起動に用いられます。
libsystemd	主となる systemd ユーティリティライブラリ。
libudev	Udev デバイス情報にアクセスするためのライブラリ。

8.79. D-Bus-1.16.2

D-Bus はメッセージバスシステムであり、アプリケーションから他のアプリケーションへの通信を容易に行う方法を提供します。 D-Bus にはシステムデーモン（例えば ”新たなハードウェアデバイスが追加されました” や ”プリンターキューが変更されました” といったイベント）やログインユーザーごとのセッションデーモン（ユーザーアプリケーション間で必要な一般的なIPC）があります。 またメッセージバスは、一般的な1対1によるメッセージ送受信のフレームワーク上にビルドされます。 これは二つのアプリケーション間にて（メッセージバスデーモンを介さずに）直接通信するために利用されます。

概算ビルド時間: 0.1 SBU
必要ディスク容量: 17 MB

8.79.1. D-Bus のインストール

D-Bus をコンパイルするための準備をします。

```
mkdir build
cd    build

meson setup --prefix=/usr --buildtype=release --wrap-mode=nofallback ..
```

meson オプションの意味

--wrap-mode=nofallback
このスイッチを指定することで、テスト実施時において meson が Glib パッケージのコピーをダウンロードしないようにします。

パッケージをコンパイルします。

```
ninja
```

ビルド結果をテストする場合は以下を実行します。

```
ninja test
```

テストの多くは、LFS に含まれない別のパッケージを必要とするため、無効化されます。 テストスイートの実行を分かりやすく説明する手順が BLFS ブック に示されています。

パッケージをインストールします。

```
ninja install
```

シンボリックリンクを生成します。 D-Bus と systemd が同一の `machine-id` ファイルを利用できるようにするためです。

```
ln -sfv /etc/machine-id /var/lib/dbus
```

8.79.2. D-Bus の構成

インストールプログラム: dbus-cleanup-sockets, dbus-daemon, dbus-launch, dbus-monitor, dbus-run-session, dbus-send, dbus-test-tool, dbus-update-activation-environment, dbus-uuidgen
インストールライブラリ: libdbus-1.so
インストールディレクトリ: /etc/dbus-1, /usr/include/dbus-1.0, /usr/lib/dbus-1.0, /usr/share/dbus-1, /usr/share/doc/dbus-1.16.2, /var/lib/dbus

概略説明

dbus-cleanup-sockets	ディレクトリ内に取り残されたソケットを削除します。
dbus-daemon	D-Bus メッセージバスデーモン。
dbus-launch	シェルスクリプトから dbus-daemon を起動します。
dbus-monitor	D-Bus メッセージバスを通じたメッセージ送信を監視します。
dbus-run-session	シェルスクリプトから dbus-daemon のセッションバスインスタンスを起動します。 そしてそのセッションにて指定されたプログラムを起動します。

dbus-send	D-Bus メッセージバスにメッセージを送ります。
dbus-test-tool	D-Bus のテストを補助するツールです。
dbus-update-activation-environment	D-Bus のセッションサービスに対して設定される環境変数を更新します。
dbus-uuidgen	ユニーク ID を生成します。
libdbus-1	D-Bus メッセージバスとの通信を行う API 関数を提供します。

8.80. Man-DB-2.13.1

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: 0.3 SBU
必要ディスク容量: 44 MB

8.80.1. 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-2.13.1 \
            --sysconfdir=/etc                     \
            --disable-setuid                      \
            --enable-cache-owner=bin              \
            --with-browser=/usr/bin/lynx          \
            --with-vgrind=/usr/bin/vgrind         \
            --with-grap=/usr/bin/grap
```

configure オプションの意味

--disable-setuid
これは man プログラムが man ユーザーに対して setuid を実行しないようにします。

--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。

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
```

8.80.2. 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
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

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
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。

8.80.3. 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-2.13.1

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。

8.81. Procps-ng-4.0.5

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: 28 MB

8.81.1. 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-4.0.5 \
            --disable-static                         \
            --disable-kill                           \
            --enable-watch8bit                       \
            --with-systemd
```

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。

--enable-watch8bit
本スイッチは watch コマンドに対して ncursesw サポートを有効にします。 これによって 8 ビット文字を取り扱うことができるようになります。

パッケージをコンパイルします。

```
make
```

テストスイートを実施する場合は以下を実行します。

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

ホストカーネルが CONFIG_BSD_PROCESS_ACCT を有効にしてビルドされていない場合に、ps with output flag bsdtime,cputime,etime,etimes という名前のテスト 1 つが失敗します。 また pgrep のテスト 1 つが chroot 環境内では失敗します。

パッケージをインストールします。

```
make install
```

8.81.2. 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-4.0.5

概略説明

free 物理メモリ、スワップメモリの双方において、メモリの使用量、未使用量を表示します。
pgrep プロセスの名前などの属性によりプロセスを調べます。
pidof 指定されたプログラムの PID を表示します。
pkill プロセスの名前などの属性によりプロセスに対してシグナルを送信します。
pmap 指定されたプロセスのメモリマップを表示します。
ps 現在実行中のプロセスを一覧表示します。
pwdx プロセスが実行されているカレントディレクトリを表示します。
slabtop リアルタイムにカーネルのスラブキャッシュ（slab cache）情報を詳細に示します。
sysctl システム稼動中にカーネル設定を修正します。

tload	システムの負荷平均（load average）をグラフ化して表示します。
top	CPU をより多く利用しているプロセスの一覧を表示します。 これはリアルタイムにプロセッサーの動作状況を逐次表示します。
uptime	システムの稼動時間、ログインユーザー数、システム負荷平均（load average）を表示します。
vmstat	仮想メモリの統計情報を表示します。 そこではプロセス、メモリ、ページング、ブロック入出力（Input/Output; IO)、トラップ、CPU 使用状況を表示します。
w	どのユーザーがログインしていて、どこから、そしていつからログインしているかを表示します。
watch	指定されたコマンドを繰り返し実行します。 そしてその出力結果の先頭の一画面分を表示します。 出力結果が時間の経過とともにどのように変わるかを確認することができます。
libproc-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。

8.82. Util-linux-2.41.2

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: 0.5 SBU
必要ディスク容量: 346 MB

8.82.1. 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 --bindir=/usr/bin     \
            --libdir=/usr/lib     \
            --runstatedir=/run    \
            --sbindir=/usr/sbin   \
            --disable-chfn-chsh   \
            --disable-login       \
            --disable-nologin     \
            --disable-su          \
            --disable-setpriv     \
            --disable-runuser     \
            --disable-pylibmount  \
            --disable-liblastlog2 \
            --disable-static      \
            --without-python      \
            ADJTIME_PATH=/var/lib/hwclock/adjtime \
            --docdir=/usr/share/doc/util-linux-2.41.2
```

--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
```

ダミーの /etc/fstab ファイルを作れば 2 つのテストがうまく実行できます。 テストスイートの実行は root ユーザー以外にて以下を実行します。

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、構築済 LFS システムを起動して以下を実行します。

```
bash tests/run.sh --srcdir=$PWD --builddir=$PWD
```

```
touch /etc/fstab
chown -R tester .
su tester -c "make -k check"
```

hardlinkテストは失敗する場合があります。 それはカーネルオプションの CONFIG_CRYPTO_USER_API_HASH が有効でない場合、あるいは SHA256 機能を提供するオプション（たとえば CONFIG_CRYPTO_SHA256 や CPU が Supplemental SSE3 をサポートする際の CONFIG_CRYPTO_SHA256_SSSE3 など）が一つもない場合です。 さらにカーネルオプション CONFIG_NETLINK_DIAG が無効である場合、lsfd 内のテスト inotify が失敗します。

パッケージをインストールします。

```
make install
```

8.82.2. 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, fincore, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hardlink, hexdump, hwclock, i386 (setarch へのリンク), ionice, ipcmk, ipcrm, ipcs, irqtop, isosize, kill, last, lastb (last へのリンク), ldattach, linux32 (setarch へのリンク), linux64 (setarch へのリンク), logger, look, losetup, lsblk, lscpu, lsipc, lsirq, lsfd, 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, rfkill, rtcwake, script, scriptlive, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swaplabel, swapoff, swapon, switch_root, taskset, uclampset, ul, umount, uname26 (setarch へのリンク), unshare, utmpdump, uuidd, uuidgen, uuidparse, wall, wdctl, whereis, wipefs, x86_64 (setarch へのリンク), 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-2.41.2, /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 ゾーン処理されたブロックデバイスの管理に用いられます。

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。 Linux が Out Of Memory となった場合に、どのプロセスを最初に kill するべきかを判断するために用いられます。

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。

fincore 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）を行います。
fsck.minix	指定されたデバイス上の Minix ファイルシステムに対して一貫性検査（consistency check）を行います。
fsfreeze	カーネルドライバー制御における 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。
hardlink	ハードリンクを生成することで重複ファイルを統合します。
hexdump	指定されたファイルを 16 進数、10 進数、8 進数、アスキーの各書式でダンプします。
hwclock	システムのハードウェアクロックを読み取ったり設定したりします。 このハードウェアクロックはリアルタイムクリック（Real-Time Clock; RTC）または BIOS（Basic Input-Output System）クロックとも呼ばれます。
i386	setarch へのシンボリックリンク。
ionice	プログラムに対する I/O スケジュールクラスとスケジュール優先度を取得または設定します。
ipcmk	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。
isosize	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。
lsfd	オープンしているファイルについての情報を表示します。 lsof に代わるものです。
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。
rfkill	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。
swaplabel	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。
uuidd	UUID ライブラリから利用されるデーモン。 時刻情報に基づく UUID を、安全にそして一意性を確保して生成します。
uuidgen	新しい UUID を生成します。 生成される UUID は乱数であり、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。 その可能性は極めて高いものです（2^{128}個の UUID が可能です）。
uuidparse	ユニークな識別子を解析するためのユーティリティー。
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。
libblkid	デバイスの識別やトークンの抽出を行う処理ルーチンを提供します。
libfdisk	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。
libmount	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。
libsmartcols	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。
libuuid	ローカルシステム内だけに限らずアクセスされるオブジェクトに対して、一意性が保証された識別子を生成する処理ルーチンを提供します。

8.83. E2fsprogs-1.47.3

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 2.4 SBU、SSD で 0.5 SBU
必要ディスク容量: 100 MB

8.83.1. 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-uuidd     \
             --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、uuidd デーモン、fsck ラッパーをいずれもビルドせずインストールしないようにします。 これらは util-linux パッケージによって、より最新のものが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
```

ビルド結果をテストする場合は以下を実行します。

```
make check
```

m_assume_storage_prezeroed というテストが 1 つだけ失敗します。 別のテスト m_rootdir_acl というものは、LFS システム向けに ext4 以外を利用している場合に失敗します。

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/{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
```

8.83.2. E2fsprogs の設定

/etc/mke2fs.conf では mke2fs のさまざまなコマンドラインオプションに対するデフォルト値が設定されてます。このファイルにおいて、必要となるデフォルト値を設定することができます。 たとえば（LFS や BLFS には含まれていない）ユーティリティーの中には、metadata_csum_seed 機能が有効になった ext4 ファイルシステムを認識できないものがあります。 もし そのようなユーティリティーを必要とする場合は、以下のコマンドを通じて ext4 のデフォルト機能を取り除くことができます。

```
sed 's/metadata_csum_seed,//' -i /etc/mke2fs.conf
```

詳しくは man ページ mke2fs.conf(5) を参照してください。

8.83.3. 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	ext{234} ファイルシステム上のファイル属性を変更します。
compile_et	エラーテーブルコンパイラー。 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ー。 これは ext{234} 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	ext{234} 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	ext{234} 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 ext{234} 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 (Multiple Mount Protection) 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すべての 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 ext{234} 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ー。
e4defrag	ext4 ファイルシステムに対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 ext{234} 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	ext{234} 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	ext{234} 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	ext{234} 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。
libe2p	dumpe2fs、chattr、lsattr の各コマンドが利用します。
libext2fs	ユーザーレベルのプログラムが ext{234} ファイルシステムを操作可能とするためのルーチンを提供します。
libss	debugfs コマンドが利用します。

8.84. 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 2 つ示します。

- 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（60% 減）
- 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（82% 減）

利用するコンパイラーや 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。 デバッグシンボルがストリップされたプログラムは、ストリップされていないものに比べて 50% から 80% のサイズ減となります。 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。

8.85. 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。 普通の Linux ユーザーにとっては、実質的な問題はありません。

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とシステムを利用不能にしてしまいます。 したがって strip コマンドを実行する前に、現時点の LFS システムのバックアップを取っておくことをお勧めします。

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通常なら不要なシンボルテーブル項目もすべて削除します。 *--strip-debug* を使うと、特定のアプリケーションに必要となるシンボルテーブルエントリは削除しなくなります。 unneeded と debug の違いはわずかなものです。 たとえば unstripped な libc.a は 22.4 MB です。 *--strip-debug* によってストリップを行うと 5.9 MB になります。 一方 *--strip-unneeded* では減じられる容量はほんのわずかで 5.8 MB となります。

選択したライブラリから得られたデバッグシンボルは、Zstd によって圧縮され、個別のファイルに保存されます。 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、元の場所に再インストールすることにします。 （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」 において説明しています。）

注記

ELF ローダーの名前は、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しています。 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。

重要

各パッケージのバージョンが、本書に示すバージョンとは異なる場合（セキュリティアドバイザリに従った場合や、必要に応じて変更した場合）、**save_usrlib** や **online_usrlib** に含まれるライブラリ名を変更することが必要かもしれません。 これを行わなかった場合には、システムが全く動作しないことも起こりえます。

```
save_usrlib="$(cd /usr/lib; ls ld-linux*[^g])
             libc.so.6
             libthread_db.so.1
             libquadmath.so.0.0.0
```

```
             libstdc++.so.6.0.34
             libitm.so.1.0.0
             libatomic.so.1.2.0"

cd /usr/lib

for LIB in $save_usrlib; do
    objcopy --only-keep-debug --compress-debug-sections=zstd $LIB $LIB.dbg
    cp $LIB /tmp/$LIB
    strip --strip-debug /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-2.45.so
               libsframe.so.2.0.0
               libhistory.so.8.3
               libncursesw.so.6.5
               libm.so.6
               libreadline.so.8.3
               libz.so.1.3.1
               libzstd.so.1.5.7
               $(cd /usr/lib; find libnss*.so* -type f)"

for BIN in $online_usrbin; do
    cp /usr/bin/$BIN /tmp/$BIN
    strip --strip-debug /tmp/$BIN
    install -vm755 /tmp/$BIN /usr/bin
    rm /tmp/$BIN
done

for LIB in $online_usrlib; do
    cp /usr/lib/$LIB /tmp/$LIB
    strip --strip-debug /tmp/$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-debug $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エラーとなりますが、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バイナリ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。

8.86. 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あります。 最近の Linux システムにおいて libtool の .la ファイルは、libltdl に対してのみ用いられます。 LFS 内のライブラリは、libltdl によってロードされるものは一つもありません。 これらのライブラリによって BLFS パッケージのビルドに失敗することが分かっています。 そこでそのようなファイルをここで削除します。

```
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

本章ではシステム設定ファイルと systemd サービスについて説明します。 まずはネットワークの設定に必要となる一般的な設定ファイルです。

- 「全般的なネットワークの設定」
- 「ホスト名の設定」
- 「/etc/hosts ファイルの設定」

次にデバイスを適切に設定するための方法について説明します。

- 「デバイスとモジュールの扱いについて」
- 「デバイスの管理」

そしてシステムクロックとキーボードレイアウトです。

- 「システムクロックの設定」
- 「Linux コンソールの設定」

またユーザーログの出力に利用されるスクリプトや設定ファイルについて触れます。

- 「システムロケールの設定」
- 「/etc/inputrc ファイルの生成」

最後に systemd の処理設定です。

- 「Systemd の利用と設定」

9.2. 全般的なネットワークの設定

本節はネットワークカードを設定する場合にのみ作業を行っていきます。

9.2.1. ネットワークインターフェースの設定ファイル

systemd はバージョン 209 から、ネットワーク設定を行うデーモン systemd-networkd を提供するようになりました。このデーモンが基本的なネットワーク設定を行います。 さらにバージョン 213 からは、DNS 名前解決を固定的に /etc/resolv.conf ファイルによって行っていたものが systemd-resolved により行うよう変更されています。 いずれのデーモンもデフォルトで有効となっています。

注記

ネットワーク設定に systemd-networkd を利用しない場合（たとえばネットワークに接続しないシステムを利用する場合や、NetworkManager のようなネットワーク設定を行う別ユーティリティーを利用する場合）は、起動時にエラーメッセージが表示されないように、サービスを無効にしてください。

```
systemctl disable systemd-networkd-wait-online
```

systemd-networkd（および systemd-resolved）に対する設定ファイルは /usr/lib/systemd/network ディレクトリまたは /etc/systemd/network ディレクトリに置きます。 /usr/lib/systemd/network ディレクトリにある設定ファイルよりも /etc/systemd/network ディレクトリにある設定ファイルの方が優先されます。 設定ファイルには .link, .netdev, .network の三種類があります。 これらの説明や設定例については man ページ systemd.link(5), systemd.netdev(5), systemd.network(5) を参照してください。

9.2.1.1. ネットワークデバイスの命名

通常 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。 インタフェース名が分からない場合は、システム起動直後に ip link を実行して確認してください。

注記

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（「Systemd-258」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。

システムにおいて、接続タイプに応じたネットワークインターフェースは、それぞれに 1 つであるのが通常です。 例えば有線接続のインターフェース名は、従来より eth0 とされます。 また無線接続の場合は wifi0 や wlan0 といった名前が用いられます。

ネットワークインターフェース名を従来どおりとしたり、カスタマイズしたりするには、以下に示す 3 通りの方法があります。

- udev のデフォルトポリシーに対する .link ファイルをマスクして無効にします。

```
ln -s /dev/null /etc/systemd/network/99-default.link
```

- インターフェースに対する名前として internet0, dmz0, lan0 といった命名スキームを自分で定めます。 これを行うには /etc/systemd/network/ ディレクトリに .link ファイルを生成し、必要なインターフェースに対して具体的な名前、つまりより良い命名スキームを定めます。 例えば以下のようにします。

```
cat > /etc/systemd/network/10-ether0.link << "EOF"
[Match]
# Change the MAC address as appropriate for your network device
MACAddress=12:34:45:78:90:AB

[Link]
Name=ether0
EOF
```

詳細は systemd.link(5) を確認してください。

- /boot/grub/grub.cfg ファイル内において、カーネルの設定行に net.ifnames=0 を追加します。

9.2.1.2. 固定 IP アドレスの設定

以下のコマンドは固定IPアドレスの設定を行う設定ファイルを生成するものです。（systemd-networkd と systemd-resolved を利用します。）

```
cat > /etc/systemd/network/10-eth-static.network << "EOF"
[Match]
Name=<network-device-name>

[Network]
Address=192.168.0.2/24
Gateway=192.168.0.1
DNS=192.168.0.1
Domains=<Your Domain Name>
EOF
```

複数のDNSサーバーを有している場合は、DNS設定行を複数指定することができます。 固定的に /etc/resolv.conf ファイルを利用する場合は DNS および Domains の設定行は記載しません。

9.2.1.3. DHCP 設定

以下のコマンドは IPv4 DHCP 設定を行う設定ファイルを生成します。

```
cat > /etc/systemd/network/10-eth-dhcp.network << "EOF"
[Match]
Name=<network-device-name>

[Network]
DHCP=ipv4

[DHCPv4]
UseDomains=true
EOF
```

9.2.2. /etc/resolv.conf ファイルの生成

インターネットへの接続を行う場合には、ドメイン名サービス（domain name service; DNS）による名前解決を必要とします。 これによりインターネットドメイン名を IP アドレスに、あるいはその逆の変換を行います。 これを行うには ISP やネットワーク管理者が指定する DNS サーバーの割り振り IP アドレスを /etc/resolv.conf ファイルに設定します。

9.2.2.1. systemd 解決による設定

注記

ネットワークインターフェース設定を systemd-resolved とは別の方法（例えば ppp など）で行う場合、 または別のタイプのローカルリゾルバー（local resolver; たとえば bind や dnsmasq や unbound など）や /etc/resolv.conf を生成するソフトウェア（つまり systemd が提供するものでない resolvconf プログラム）などを用いる場合、systemd-resolved サービスは用いてはなりません。

systemd-resolved を無効にするには、以下のコマンドを実行します。

```
systemctl disable systemd-resolved
```

DNS 設定に systemd-resolved を用いると /run/systemd/resolve/resolv.conf ファイルが生成されます。 また /etc/resolv.conf が存在していない場合は、systemd-resolved が /run/systemd/resolve/stub-resolv.conf へのシンボリックリンクとして生成します。 その場合は /etc/resolv.conf を手動で生成する必要はありません。

注記

LFS システムにおいて chroot 環境下においても systemd-resolved を利用できるようにしたい場合（たとえば BLFS パッケージのビルドにあたってビルド処理内にてインターネット接続を必要とする場合）、以下に示すような /etc/resolv.conf のスタティックな設定を行い、chroot 環境でも名前解決が動作するようにします。 chroot 環境から抜け出たときに、その設定を削除すれば systemd-resolved が起動時にシンボリックリンクを生成します。

9.2.2.2. スタティックな resolv.conf 設定

スタティックな /etc/resolv.conf ファイルを必要とする場合は、以下のコマンドにより生成します。

```
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。 これ以外の方法として、IP アドレスに Google Public DNS サービスをネームサーバーとして利用する方法もあります。

注記

Google Public IPv4 DNS アドレスは *8.8.8.8* と *8.8.4.4* です。 また IPv6 では *2001:4860:4860::8888* と *2001:4860:4860::8844* です。

9.2.3. 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。

9.2.4. /etc/hosts ファイルの設定

完全修飾ドメイン名（Fully Qualified Domain Name; FQDN）、エイリアスの各設定は /etc/hosts ファイルにて行います。 固定アドレスを用いる場合は IPアドレスを定める必要があります。 ホストファイルの文法は以下のとおりです。

```
IP_address myhost.example.org aliases
```

インターネットに公開されていないコンピューターである場合（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。

インターネット上にコンピューターを公開しようとする場合、正しい FQDN はドメイン名そのものか、あるいはプレフィックス（たいていはホスト名）とドメイン名を「.」でつなげて記述します。 そしてドメインプロバイダーに問い合わせて、FQDN を公開 IP アドレスとして解決する必要があります。

インターネット上にコンピューターが公開されていない場合であっても、特定のプログラム、たとえば MTA などにおいては、正常な処理が行われるように FQDN が必要になります。 特別な FQDN localhost.localdomain は、その目的で利用されます。

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

<192.168.0.2> <FQDN> [alias1] [alias2] ...
::1       ip6-localhost ip6-loopback
ff02::1   ip6-allnodes
ff02::2   ip6-allrouters

# End /etc/hosts
EOF
```

<192.168.0.2>, <FQDN> 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。 また <192.168.0.2> の行も、DHCP や IPv6 による自動設定による接続を行う場合、あるいは FQDN として localhost.localdomain を指定する場合には省略可能です。

/etc/hostname には localhost, localhost.localdomain, （ドメイン名を含まない）ホスト名は記述しません。 これらは NSS モジュール myhostname が取り扱います。 詳しくは man ページ nss-myhostname(8) を参照してください。

::1 という項目は IPv6 における 127.0.0.1 に相当し、IPv6 のループバックインターフェースを表します。

9.3. デバイスとモジュールの扱いについて

第 8 章の systemd のビルドを通じて udev デーモンをインストールしました。 この udev 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ます。

udev による方法では、カーネルが検知したデバイスに対してのみ、デバイスノードが生成されます。 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、システム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 devfs は非推奨（deprecated）とされ、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に対して提供したことです。 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、そのモジュールがロードされたときに登録されます。 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 udevd 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 ファイルシステム内に作り出されます。 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、所有者の情報とともに名前空間が公開されます。

この後にカーネルは udevd に対して uevent を送信します。 udevd 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 udevd 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 udevd 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ーの場合、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 ドライバーは、ベンダーID 0x1319 とデバイスID 0x0801 の PCI ドライバーをサポートします。 そして pci:v00001319d00000801sv*sd*bc04sc01i* というエイリアスがあります。 たいていのデバイスでは、sysfs を通じてドライバーがデバイスを扱うものであり、ドライバーのエイリアスをバスドライバーが提供します。 /sys/bus/pci/devices/0000:00:0d.0/modalias ファイルならば pci:v00001319d00000801sv00001319sd00001319bc04sc01i00 という文字列を含んでいるはずです。 udev が提供するデフォルトの生成規則によって udevd から /sbin/modprobe が呼び出されることになり、その際には uevent に関する環境変数 MODALIAS の設定内容が利用されます。（この環境変数の内容は sysfs 内の modalias ファイルの内容と同じはずです。）そしてワイルドカードが指定されているならそれが展開された上で、エイリアス文字列に合致するモジュールがすべてロードされることになります。

上の例で forte ドライバーがあったとすると、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。 不要なドライバー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 udevd が取り扱うことになります。

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、バスドライバーが 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-6.16.9 においての udev は、INPUT、IDE、PCI、USB、SCSI、SERIO、FireWire の各デバイスに対するドライバーをロードします。 それらのデバイスドライバー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ーが 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、そのドライバー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、それはドライバーのバグです。 その場合は udev に頼ることなくドライバー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-6.16.9 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)」ドライバーや 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ーは、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 が間違った名前のデバイスを生成していないことです。

カーネルドライバーがそのデータを sysfs にエクスポートしていない場合、udev はデバイスノード生成に必要な情報を得ていないことになります。 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変わってしまう問題

これは udev の設計仕様に従って発生するもので、uevent の扱いとモジュールのロードが平行して行われるためです。 このために命名順が予期できないものになります。 これを「固定的に」することはできません。 ですからカーネルがデバイス名を固定的に定めるようなことを求めるのではなく、シンボリックリンクを用いた独自の生成規則を作り出して、そのデバイスの固定的な属性を用いた固定的な名前を用いる方法を取ります。 固定的な属性とは例えば、udev によってインストールされるさまざまな *_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 https://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. 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変わりありません。（したがって直接このデバイス名を使ってはなりません。）しかしシンボリックリンク /dev/tvtuner と /dev/webcam は常に正しいデバイスを指し示すようになります。

9.5. システムクロックの設定

本節ではシステムサービス systemd-timedated の設定方法について示します。 このサービスはシステムクロックとタイムゾーンの設定を行うものです。

ハードウェアクロックが UTC に設定されているかどうか忘れた場合は **hwclock --localtime --show** を実行すれば確認できます。 このコマンドにより、ハードウェアクロックに基づいた現在時刻が表示されます。 その時刻が手元の時計と同じ時刻であれば、ローカル時刻として設定されているわけです。 一方それがローカル時刻でなかった場合は、おそらくは UTC に設定されているからでしょう。 hwclock によって示された時刻からタイムゾーンに応じた一定時間を加減してみてください。 例えばタイムゾーンが MST であった場合、これは GMT -0700 なので、7時間を加えればローカル時刻となります。

systemd-timedated コマンドは /etc/adjtime ファイルを読み込みます。 そしてこのファイルの設定内容に応じて、システムクロックを UTC かあるいはローカル時刻に設定します。

ハードウェアクロックをローカル時刻に設定する場合は、以下の内容により /etc/adjtime ファイルを生成します。

```
cat > /etc/adjtime << "EOF"
0.0 0 0.0
0
LOCAL
EOF
```

起動時に /etc/adjtime ファイルが存在しなかった場合、ハードウェアクロックは UTC に設定されているものとして systemd-timedated が判断し、このファイルを調整します。

timedatectl ユーティリティーを用いる方法もあります。 これを使って systemd-timedated に対し、ハードウェアクロックが UTC かローカル時刻かを設定することができます。

```
timedatectl set-local-rtc 1
```

timedatectl コマンドを用いれば、システム時刻やタイムゾーンを変更することもできます。

システム時刻を変更するには以下を実行します。

```
timedatectl set-time YYYY-MM-DD HH:MM:SS
```

ハードウェアクロックも同様に設定することができます。

タイムゾーンを変更するには以下を実行します。

```
timedatectl set-timezone TIMEZONE
```

利用可能なタイムゾーンの一覧は以下を実行して確認できます。

```
timedatectl list-timezones
```

注記

timedatectl コマンドは chroot 環境内では動作しない点に注意してください。 systemd を使って LFS システムを起動したときになって、初めて利用できるものです。

9.5.1. ネットワークによる時刻同期

systemd のバージョン 213 からは systemd-timesyncd というデーモンが提供されています。 これはシステム時刻とリモートの NTP サーバーの時刻同期を行うものです。

このデーモンは、NTP デーモンとして充実したものではありません。 NTP デーモンに代わるものと位置づけられるものではなく、SNTP プロトコルのクライアントのみの実装であり、簡単なタスクの処理やリソースが限られているシステム上にて用いられます。

systemd のバージョン 216 からはデフォルトで systemd-timesyncd デーモンが用いられます。 これを無効にしたい場合は以下を実行します。

```
systemctl disable systemd-timesyncd
```

systemd-timesyncd が利用する NTP サーバーを変更するには /etc/systemd/timesyncd.conf ファイルを用います。

システムクロックがローカル時刻に設定されている場合、systemd-timesyncd はハードウェアクロックを更新しない点に注意してください。

9.6. Linux コンソールの設定

この節ではシステムサービス systemd-vconsole-setup の設定方法について説明します。 このサービスは仮想コンソールフォントとコンソールキーマップを設定します。

systemd-vconsole-setup サービスは、/etc/vconsole.conf ファイルにて示される設定情報を読み込みます。 キーマップやスクリーンフォントには何を用いるのかを定めてください。 各言語に対する HOWTO も確認してください。 https://tldp.org/HOWTO/HOWTO-INDEX/other-lang.html が参考になるでしょう。 localectl list-keymaps を実行すると、設定可能なコンソールキーマップを確認できます。 また /usr/share/consolefonts ディレクトリを見れば、設定可能なスクリーンフォントを確認できます。

/etc/vconsole.conf ファイルの各行は VARIABLE=value といった書式により構成されます。 VARIABLE には以下の変数を利用します。

KEYMAP
: この変数はキーボードに対するキーマッピングテーブルを指定します。 これが定められていない場合はデフォルトで us が設定されます。

KEYMAP_TOGGLE
: この変数は二番目のトグルキーマップを設定します。 デフォルトでは本変数は設定されません。

FONT
: この変数は仮想コンソールにて用いられるフォントを指定します。

FONT_MAP
: この変数はコンソールマップを指定します。

FONT_UNIMAP
: この変数は Unicode フォントマップを指定します。

「システムロケールの設定」 においては、Linux コンソールでの対話セッションのロケールは C.UTF-8 を用いることにします。 C.UTF-8 のもとでプログラムメッセージの全文字に対するグリフを持っているコンソールフォントと言えば、Kbd の中では LatArCyrHeb*.psfu.gz, LatGrkCyr*.psfu.gz, Lat2-Terminus16.psfu.gz, pancyrillic.f16.psfu.gz です。 これらは /usr/share/consolefonts にあります。（その他のコンソールフォントは、ユニコードの左右のクォーテーションマークや英語ダッシュなどのグリフが欠落しています。）したがってデフォルトのコンソールフォントとして、たとえば Lat2-Terminus16 を設定します。

```
echo FONT=Lat2-Terminus16 > /etc/vconsole.conf
```

ドイツのキーボードおよびコンソールの設定例は以下です。

```
cat > /etc/vconsole.conf << "EOF"
KEYMAP=de-latin1
FONT=Lat2-Terminus16
EOF
```

localectl ユーティリティーを用いれば、システム稼動中に KEYMAP 変数を変更することができます。

```
localectl set-keymap MAP
```

注記

localectl コマンドは chroot 環境内では動作しない点に注意してください。 systemd を使って LFS システムを起動したときになって、初めて利用できるものです。

localectl ユーティリティーはまた、X11 キーボードレイアウト、モデル、ヴァリアント、オプションをそれぞれ対応する変数により設定することができます。

```
localectl set-x11-keymap LAYOUT [MODEL] [VARIANT] [OPTIONS]
```

localectl set-x11-keymap に対して設定可能な値の一覧は、以下の変数を使って localectl を実行して得ることができます。

list-x11-keymap-models
X11 キーボードマッピングモデルを表示します。

list-x11-keymap-layouts
X11 キーボードマッピングレイアウトを表示します。

list-x11-keymap-variants
X11 キーボードマッピングヴァリアントを表示します。

list-x11-keymap-options
X11 キーボードマッピングオプションを表示します。

注記

上に示す変数を利用するにあたっては BLFS ブックに説明する XKeyboard-Config パッケージが必要です。

9.7. システムロケールの設定

環境変数の中には、ネイティブな言語サポートのために必要になるものがあります。 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、第 8 章にてロケールをインストールしていないか、あるいはそのロケールが Glibc のデフォルトのインストールではサポートされていないかのいずれかです。

```
locale: Cannot set LC_* to default locale: No such file or directory
```

このエラーが発生したら localedef コマンドを使って、目的とするロケールをインストールするか、別のロケールを選ぶ必要があります。 これ以降の説明では Glibc がこのようなエラーを生成していないことを前提に話を進めます。

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/locale.conf ファイルを生成します。

```
cat > /etc/locale.conf << "EOF"
LANG=<ll>_<CC>.<charmap><@modifiers>
EOF
```

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。

ログインシェルは /etc/locale.conf における設定に影響を受けないこともあります。 /etc/locale.conf のロケール設定を読み込んでエクスポートするために /etc/profile を生成します。 , ただし Linux コンソールの起動中は、上ではなく C.UTF-8 を設定します。（Linux コンソールが表示できない文字を出力しないようにするためです。）

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

for i in $(locale); do
  unset ${i%=*}
done

if [[ "$TERM" = linux ]]; then
  export LANG=C.UTF-8
else
  source /etc/locale.conf

  for i in $(locale); do
    key=${i%=*}
    if [[ -v $key ]]; then
      export $key
    fi
  done
fi

# End /etc/profile
EOF
```

/etc/locale.conf ファイルは systemd のユーティリティープログラム localectl を使って定めることもできます。 例えば上と同じ設定を行うには以下を実行します。

```
localectl set-locale LANG="<ll>_<CC>.<charmap><@modifiers>"
```

言語に関連する環境変数、例えば LANG, LC_CTYPE, LC_NUMERIC などや、locale が出力する環境変数を指定することもできます。 その場合は各設定をスペースにより区切ります。 例として LANG を en_US.UTF-8 とし LC_CTYPE を単に en_US とする場合は以下のようにします。

```
localectl set-locale LANG="en_US.UTF-8" LC_CTYPE="en_US"
```

注記

localectl コマンドは chroot 環境内では動作しない点に注意してください。 systemd を使って LFS システムを起動したときになって、初めて利用できるものです。

ロケール設定の C（デフォルト）と en_US（米国の英語利用ユーザーに推奨）は異なります。 C は US-ASCII 7 ビットキャラクタ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ョン記号が表示されることがあるのはこのためです。 また Mutt や Pine などにより電子メールが送信される際に、そういった文字は RFC には適合しないメールとして送信されます。 送信された文字は unknown 8-bit(不明な 8ビット）として示されます。 そこで 8ビット文字を必要としないことが明らかな場合には C ロケールを指定してください。

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ければ、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
```

9.10. Systemd の利用と設定

9.10.1. 基本的な設定

/etc/systemd/system.conf ファイルには、基本的な systemd 動作を制御するための設定オプション項目があります。 デフォルトのファイルは、各項目のデフォルト値が示された上でそれがコメントアウトされています。 このファイルでは基本的なジャーナル設定やログレベルを設定する必要があります。 各オプションの詳細については man ページ systemd-system.conf(5) を参照してください。

9.10.2. ブート時の画面クリアの防止

通常 systemd はブート処理の最後に画面をクリアします。 必要ならばこの動きを以下のようにして変更することができます。

```
mkdir -pv /etc/systemd/system/getty@tty1.service.d

cat > /etc/systemd/system/getty@tty1.service.d/noclear.conf << EOF
[Service]
TTYVTDisallocate=no
EOF
```

ブートメッセージは、root ユーザーになってコマンド **journalctl -b** を実行することで、常に表示しておくこともできます。

9.10.3. /tmp の tmpfs としての生成抑止

デフォルトでは /tmp は tmpfs として生成されます。 これが適当ではないならば、以下のコマンドによりオーバーライドすることができます。

```
ln -sfv /dev/null /etc/systemd/system/tmp.mount
```

それとは別に /tmp を別パーティションとする場合は、/etc/fstab にそのパーティションを指定します。

警告

/tmp を別パーティションとした場合、このパーティションに対してシンボリックリンクを作成することは避けてください。 これを行ってしまうと、ルートファイルシステム (/) を r/w として再マウントすることができなくなり、システムを再起動すると利用できなくなります。

9.10.4. 自動的なファイル生成、削除の設定

ファイルやディレクトリを生成、削除するサービスがいくつかあります。

- systemd-tmpfiles-clean.service
- systemd-tmpfiles-setup-dev.service
- systemd-tmpfiles-setup.service

システム用設定ファイルは /usr/lib/tmpfiles.d/*.conf です。 ローカル用設定ファイルは /etc/tmpfiles.d/*.conf に置きます。 /etc/tmpfiles.d にあるファイルは /usr/lib/tmpfiles.d にある同名ファイルをオーバーライドします。 ファイル書式の詳細については man ページ tmpfiles.d(5) を参照してください。

/usr/lib/tmpfiles.d/*.conf ファイルの文法はやっかいなものです。 例えば /tmp ディレクトリ内のファイルを消去するためのデフォルト設定は /usr/lib/tmpfiles.d/tmp.conf ファイルに以下のように記述されます。

```
q /tmp 1777 root root 10d
```

型を表わす q はクォータを用いたサブボリュームを生成することを意味します。 ただこれが適用できるのは btrfs ファイルシステムのみです。 この型は v を参照し、次に d（ディレクトリ）を参照します。 指定されたディレクトリが存在しない場合はそれが生成されて、パーミッションと所有者が指定されたものに設定されます。 時間指定が行われた場合、そのディレクトリ内のファイルは、それに応じて削除されます。

デフォルトパラーメーターを必要としない場合は、設定ファイルを /etc/tmpfiles.d にコピーして必要な設定を行っておきます。 例えば以下です。

```
mkdir -p /etc/tmpfiles.d
cp /usr/lib/tmpfiles.d/tmp.conf /etc/tmpfiles.d
```

9.10.5. デフォルトのサービス動作のオーバーライド

ユニットパラメーターをオーバーライドするには /etc/systemd/system ディレクトリを生成して設定ファイルを作成します。 例えば以下のとおりです。

```
mkdir -pv /etc/systemd/system/foobar.service.d

cat > /etc/systemd/system/foobar.service.d/foobar.conf << EOF
[Service]
Restart=always
RestartSec=30
EOF
```

詳しくは man ページ systemd.unit(5) を参照してください。 設定ファイルを作成したら **systemctl daemon-reload** と **systemctl restart foobar** を実行します。 これによりサービスの設定内容が反映されます。

9.10.6. ブートシーケンスのデバッグ

SysVinit や BSD スタイルの起動システムにおいては単純なシェルスクリプトが用いられていますが、 systemd ではさまざまな形式の起動ファイル（あるいはユニット）を統一化するフォーマットが用いられています。 systemctl コマンドがユニットファイルの有効/無効、状態制御/参照を行います。 以下に示すものがよく用いられます。

- systemctl list-units -t ***<service>*** [--all]: サービスタイプのユニットファイルをロードします。
- systemctl list-units -t ***<target>*** [--all]: ターゲットタイプのユニットファイルをロードします。
- systemctl show -p Wants ***<multi-user.target>***: マルチユーザーターゲットに依存するユニットをすべて表示します。 ターゲットは特別なユニットファイルであり、SysVinit におけるランレベルに相当します。
- systemctl status ***<servicename.service>***: servicename で示されるサービスの状態を表示します。 拡張子 .service は、他に同名のサービスがない限り、例えば .socket ファイルであるような場合は省略することができます。（.socket ファイルは inetd/xinetd と同様の機能を提供するソケットを生成します。）

9.10.7. Systemd ジャーナル関連の操作

systemd により起動したシステムのシステムログは、従来の unix syslog デーモンとは異なり、デフォルトで systemd-journald により扱われます。 必要に応じて標準的な syslog デーモンを追加することも可能で、両者を併用することもできます。 systemd-journald プログラムはジャーナル項目を保存しますが、それはテキストログファイルではなく、バイナリフォーマットファイルです。 そのファイル内容を確認するために journalctl コマンドが提供されています。 以下に示すものがよく用いられます。

- journalctl -r: ジャーナル項目すべてを日付の昇順により表示します。
- journalctl -u ***UNIT***: 指定された UNIT ファイルに関連したジャーナル項目を表示します。
- journalctl -b[=ID] -r: 直近の起動成功から（あるいはブートIDから）のジャーナル項目を、日付の昇順により表示します。
- journalctl -f: tail -f と同様の機能を提供します。

9.10.8. コアダンプ関連の操作

クラッシュしたプログラムをデバッグするのに、コアダンプというものが重宝します。 特にデーモンプロセスがクラッシュした場合です。 systemd によるブートシステムにおいて、コアダンプは systemd-coredump が取り扱います。 このプログラムはジャーナル内にコアダンプのログを出力し、コアダンプそのものは /var/lib/systemd/coredump に保存します。 コアダンプを取り出して処理するために coredumpctl というツールが提供されています。 よく利用されるコマンド例を以下に示します。

- coredumpctl -r: すべてのコアダンプを新しい順に一覧表示します。
- coredumpctl -1 info: 最新のコアダンプの情報を表示します。
- coredumpctl -1 debug: 最新のコアダンプを GDB にロードします。

コアダンプはディスク容量を大量に消費することがあります。 /etc/systemd/coredump.conf.d に設定ファイルを生成して、 コアダンプに利用するディスク容量の最大を制御することができます。 たとえば以下のとおりです。

```
mkdir -pv /etc/systemd/coredump.conf.d

cat > /etc/systemd/coredump.conf.d/maxuse.conf << EOF
[Coredump]
MaxUse=5G
EOF
```

詳細は systemd-coredump(8), coredumpctl(1), coredump.conf.d(5) の各 man ページを参照してください。

9.10.9. 稼動し続けるプロセス

systemd-230 より取り入れられた機能として、ユーザープロセスは、たとえ nohup が用いられたり、あるいは daemon() や setsid() が利用されたプロセスであっても、ユーザーセッションが終了するとともに終了します。 この機能変更は、従来からの柔軟な実装を厳格なものとする意図で行われたものです。 したがって稼動し続けるプロセスが利用されていると（例えば screen や tmux など)、この機能変更が問題を引き起こすことになるかもしれません。 つまりユーザーセッションが終了した後にもプロセスをアクティブにしておくことが必要になります。 ユーザーセッション終了後にプロセスを継続させる方法として、以下の三つの方法があります。

- 指定ユーザーのプロセスを継続させる方法: 標準的なユーザーは自身のユーザー権限においてコマンド loginctl enable-linger を実行して、プロセスを継続させることができます。 システム管理者は *user* 引数を利用して、そのユーザーに対して同一のコマンドを実行可能です。 そしてそのユーザーは systemd-run コマンドを実行することでプロセスを継続的に稼動させます。 例えば systemd-run --scope --user /usr/bin/screen などとします。 特定ユーザーに対してのプロセス継続を行った場合、ログインセッションがすべて終了しても user@.service が残ります。 そしてこれはシステム起動時にも自動実行されます。 つまりユーザーセッションが終了した後にもプロセスの有効無効の制御が明示的に行えるものであり、nohup や deamon() を利用するユーティリティーなどの下位互換性をなくすものです。
- システムワイドなプロセスを継続させる方法: /etc/systemd/logind.conf ファイル内に *KillUserProcesses=no* を指定すれば、全ユーザーに対してグローバルにプロセスを継続起動させることができます。 これは明示的に制御する方法を無用とし、従来どおり全ユーザーに対しての方式を残すメリットがあります。
- 機能変更をビルド時に無効化する方法: プロセス継続をデフォルトとするために systemd のビルド時に meson コマンドにおいて *-D default-kill-user-processes=false* スイッチを指定する方法があります。 この方法をとれば、systemd がセッション終了時にユーザープロセスを終了させてしまう機能を完全に無効化することができます。

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

# 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 fstab(5) 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のようになります。

```
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。

10.3. Linux-6.16.9

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: 0.4 - 32 SBU（一般的には 2.5 SBU 程度）
必要ディスク容量: 1.7 - 14 GB（一般的には 2.3 GB 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。

重要

Linux カーネルの構築を初めて行うなら、LFS の中でも、かなりハードルの高い作業になります。 これをうまく成功させることができるかどうかは、対象システム向けの特定ハードウェアの存在や、どのように作り上げたいかの要求に依存します。 カーネルに設定できる項目は、ほぼ 12,000 項目もあります。 ただしたいていのコンピューターにおいて、必要となる項目はその 3 分の 1 程度です。 LFS 編集者としては、この作業手順に不慣れなユーザーであれば、以降に示す手順をほぼそっくり従って頂くことをお勧めしています。 ここでの目的は、後に 「システムの再起動」 を経てシステムを再起動した際に、この新システムに向けて、コマンドラインからログインできるようにすることです。 この段階では、最適化やカスタマイズを目指すものではありません。

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。 さらに詳しくカーネルの構築や設定を説明している https://anduin.linuxfromscratch.org/LFS/kernel-nutshell/ もあります。 この情報を少々古いものですが、理にかなった作業過程をおおまかに示しています。

それでもうまくいかなかった場合は、lfs-support メーリングリストに問い合わせる方法があります。 スパムメールを避ける目的から、このメーリングリストは登録が必要です。

コンパイルするための準備として以下のコマンドを実行します。

```
make mrproper
```

これによりカーネルソースが完全にクリーンなものになります。 カーネル開発チームは、カーネルコンパイルするなら、そのたびにこれを実行することを推奨しています。 tar コマンドにより伸張しただけのソースではクリーンなものにはなりません。

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。

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。

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 --->
  [*] x2APIC interrupt controller architecture support              [X86_X2APIC]

Device Drivers --->
  [*] PCI support --->                                                      [PCI]
    [*] Message Signaled Interrupts (MSI and MSI-X)                     [PCI_MSI]
  [*] IOMMU Hardware Support --->                                   [IOMMU_SUPPORT]
    [*] Support for Interrupt Remapping                                [IRQ_REMAP]
```

LFS システムを配置するパーティションが NVME SSD（つまりデバイスノードが /dev/sd* でなく /dev/nvme*）である場合は、NVME サポートを有効にしてください。 これを行っていないと、LFS システムが起動しません。

```
Device Drivers --->
  NVME Support --->
    <*> NVM Express block device                                    [BLK_DEV_NVME]
```

注記

”The IPv6 Protocol”については厳密には不要としても良いものですが、システム開発者は強く推奨しているものです。

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 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。 これは、ホストディストリビューションにて UEFI ブートローダーを利用している場合であっても同様です。

上の設定項目の説明

Randomize the address of the kernel image (KASLR)
カーネルイメージにおいて ASLR を有効にします。 これによって、カーネル内にある機密コードやデータが、固定的なアドレスに存在することを前提とした攻撃を軽減できます。

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。

Configure standard kernel features (expert users)
これは設定項目上にいくつかのオプションを表示するものですが、そのオプションを変更することは非常に危険なことです。 何を行っているのかがわかっていない場合には、触れないようにしてください。

Strong Stack Protector
カーネルにおいて SSP を有効にします。 ユーザー空間全体に対してこれを有効にするには、GCC のコンパイルにあたって *--enable-default-ssp* を指定します。 ただしカーネルは、GCC のデフォルト設定として SSP を利用しません。 したがってここで明示的な指定を行います。

Support for uevent helper
本項目を有効にすることで、デバイス管理を Udev 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 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。

Display a user-friendly message when a kernel panic occurs
カーネルパニック発生にあたって、起動中の DRM ドライバーの出力機能が適切にサポートされている場合に、メッセージを正しく表示します。 これがなかった場合には、パニック内容を調べることがより困難になります。 たとえば DRM ドライバーが起動していなかった場合は、VGA コンソールを利用することになり、その場合には 24 行の表示しか行われず、相当数のカーネルメッセージは消えてなくなってしまいます。 また DRM ドライバーが起動していても、パニック時のメッセージは非常に複雑です。 Linux-6.12 の場合、主要な GPU モデルの専用ドライバーはどれもこれに対応していませんが、「Simple framebuffer driver」であれば対応しています。 これであれば VESA（あるいは EFI）フレームバッファー上で作動し、GPU 専用ドライバーがロードされる前であってかまいません。 GPU 専用ドライバーが（カーネルイメージの一部としてではなく）モジュールとしてビルドされていて、かつ initramfs が利用されていない場合は、ルートファイルシステムのマウント前であっても正しく機能します。 そして LFS の設定誤りがパニックを引き起こしている（たとえば 「GRUB を用いたブートプロセスの設定」における *root=* の設定が適切でない）場合に、情報表示が充分に行われることになります。

Panic screen formatter
これを kmsg に設定すると、カーネルパニックが発生した際に、カーネルメッセージの最終行付近を確実に表示するようになります。 デフォルト設定は user であり、その場合カーネルは「ユーザーフレンドリーな」 パニックメッセージしか表示せず、これでは解析になんら役立ちません。 もう一つの設定として qr_code がありますが、これはカーネルメッセージの最終行付近を圧縮して QR コードとして表示します。 QR コードであればプレーンテキストに比べて、それ以上に多くのメッセージを保持することができ、別のデバイス（たとえばスマートホン）上で圧縮の展開を行うことができます。 ただしこれを実現するためには LFS では提供していない Rust コンパイラーが必要となります。

Mark VGA/VBE/EFI FB as generic system framebuffer , Simple framebuffer driver
これは DRM デバイスとして VESA フレームバッファーを利用するようにします（UEFI 経由により LFS システムを起動する場合には EFI フレームバッファーを利用するようにします）。 VESA フレームバッファーは GRUB によって（あるいは EFI フレームバッファーにおいては UEFI ファームウェアによって）設定されます。 したがって DRM におけるパニック処理は、GPU 固有の DRM ドライバーがロードされる前であっても正しく機能します。

Enable legacy fbdev support for your modesetting driver , Framebuffer Console support
これは DRI（Direct Rendering Infrastructure）ドライバーにより起動される GPU 上に Linux コンソールを表示するために必要となります。 CONFIG_DRM（Direct Rendering Manager）を有効にしている場合は、この 2 つのオプションも同じく有効にしておく必要があります。 そうしておかないと、DRI ドライバーのロードの際に画面がブランクになってしまいます。

Support x2apic
64 ビット x86 プロセッサーの 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。

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-6.16.9 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-6.16.9/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なければ、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

LFS システムにおいて、/boot パーティションを切り分けて用意することにした場合（おそらくホストディストロの /boot パーティションを共用とする場合）、以降でコピーするファイルがそこに入ります。 これを最も簡単に行うには、/etc/fstab 内に /boot 用のエントリーを生成します（詳細は前節を参照してください）。 そして chroot 環境 内の root ユーザーになって、以下のコマンドを実行します。

```
mount /boot
```

コマンド実行にあたっては、デバイスノードへのパスは省略します。 これは mount コマンドが /etc/fstab から読み込むからです。

カーネルイメージへのパスは、利用しているプラットフォームによってさまざまです。 そのファイル名は、好みにより自由に変更して構いません。 ただし 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-6.16.9-lfs-r12.4-29-systemd
```

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-6.16.9
```

カーネル設定ファイル .config は、上で実行した make menuconfig によって生成されます。 このファイル内には、今コンパイルしたカーネルの設定項目の情報がすべて保持されています。 将来このファイルを参照する必要が出てくるかもしれないため、このファイルを保存しておきます。

```
cp -iv .config /boot/config-6.16.9
```

Linux カーネルのドキュメントをインストールします。

```
cp -r Documentation -T /usr/share/doc/linux-6.16.9
```

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-6.16.9 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。

保持しておいたカーネルソースを使って、カーネル設定の更新およびカーネルの再ビルドを行う場合、普通は make mrproper コマンドは実行しないでください。 このコマンドを実行すると、前回のビルド時に生成された .config ファイルと、拡張子 .o のファイルすべてを削除します。 .config だけなら /boot からコピーすれば簡単に復元できます。 しかし .o ファイルをすべて削除すると、またビルドに時間を要することになります。 たとえば単純な設定を変更するだけであったなら、(再)生成すべき .o ファイルは少ないはずであり、それ以外の .o ファイルは残しておけば、カーネルビルドシステムは適切にビルドをスキップしてくれます。

それとは逆に GCC のアップグレードを行っていた場合には make clean を実行して、前回ビルドされた .o ファイルは削除しておかなければなりません。 これを行わなかった場合、新たなビルドが失敗する可能性があります。

警告

カーネルを説明する書の中には、カーネルのソースディレクトリに対してシンボリックリンク /usr/src/linux の生成を勧めているものがあります。 これはカーネル 2.6 系以前におけるものであり LFS システム上では生成してはなりません 。 ベースとなる LFS システムを構築し、そこに新たなパッケージを追加していこうとした際に、そのことが問題となるからです。

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-6.16.9, vmlinuz-6.16.9-lfs-r12.4-29-systemd, and System.map-6.16.9
インストールディレクトリ: /lib/modules, /usr/share/doc/linux-6.16.9

概略説明

config-6.16.9	カーネルの設定をすべて含みます。
vmlinuz-6.16.9-lfs-r12.4-29-systemd	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-6.16.9	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、本ページに示す手順は読み飛ばしてください。 ただし grub.cfg の文法を学ぶ場合や、ファイル内にあるパーティションの指定方法を学ぶ場合は確認しておいてください。 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（resucue)」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（hdn,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は 5 から）数え始めます。 かつてのバージョンでは共にゼロから数え始めていましたが、今はそうではないので注意してください。 例えば 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。 こうしておくと、この後に LFS であろうが商用ディストリビューションであろうが、システム導入する際に同一のブートファイルを利用することが可能です。 つまりどのようなブートシステムからでもアクセスが可能となります。 この方法をとるなら、新たなパーティションをマウントした上で、現在 /boot ディレクトリにある全ファイルを（例えば前節にてビルドした Linux カーネルも）新しいパーティションに移動させる必要があります。 そしていったんパーティションをアンマウントし、再度 /boot としてマウントしなおすことになります。 これを行った後は/etc/fstab を適切に書き換えてください。

現時点での LFS パーティションにて /boot を残しておいて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当であるならコマンドを実行しないでください。 例えばマスターブートレコード (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ー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 part_gpt
insmod ext2
set root=(hd0,2)
set gfxpayload=1024x768x32

menuentry "GNU/Linux, Linux 6.16.9-lfs-r12.4-29-systemd" {
        linux   /boot/vmlinuz-6.16.9-lfs-r12.4-29-systemd root=/dev/sda2 ro
}
EOF
```

insmod コマンドは GRUB モジュールである `part_gpt` と `ext2` をロードします。 そしてその名前こそ `ext2` となっていますが、このモジュールは実際には `ext2`, `ext3`, `ext4` の各ファイルシステムをサポートしています。 grub-install コマンドによっていくつかのモジュールは、メインの (MBR または GRUB BIOS パーティションにインストールされる) GRUB イメージ内に埋め込まれており、鶏が先か卵が先かという問題を生じさせることなく、そこから (`/boot/grub/i386-pc` にある) 他モジュールへのアクセスを可能としています。 したがってごく普通の設定を行っていれば、上述の 2 つのもジュールはすでに埋め込まれていることとなり、insmod コマンドは何も行わないことになります。 そうなったとしても何も問題はありませんが、特殊な設定を行った際には必要となるかもしれません。

set gfxpayload=1024x768x32 コマンドは VESA フレームバッファーの解像度と色の深さを設定するものであり、これがカーネルに受け渡されます。 VESA フレームバッファー向けにカーネルの SimpleDRM ドライバーを用いる場合にこの指定が必要になります。 モニター画面に最適な解像度や色深さを選んでください。

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 指定子が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ょう。

/etc/lfs-release というファイルをここで作成することにします。 このファイルを作っておけば、どのバージョンの LFS をインストールしたのか、すぐに判別できます。（もしあなたが質問を投げた時には、我々もすぐに判別できることになります。）以下のコマンドによりこのファイルを生成します。

```
echo r12.4-29-systemd > /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="r12.4-29-systemd"
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="r12.4-29-systemd"
ID=lfs
PRETTY_NAME="Linux From Scratch r12.4-29-systemd"
VERSION_CODENAME="<your name here>"
HOME_URL="https://www.linuxfromscratch.org/lfs/"
RELEASE_TYPE="development"
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。 以下にその内容を示します。

- 利用するハードウェア用のカーネルドライバーが、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。
- root ユーザーのパスワードが設定されていることを確認してください。
- 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/fstab
  - /etc/hosts

- /etc/inputrc
- /etc/profile
- /etc/resolv.conf (optional)
- /etc/vimrc

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
mountpoint -q $LFS/dev/shm && umount -v $LFS/dev/shm
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/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
```

システムを再起動します。

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 r12.4-29-systemd 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。 起動後に見えるのは「login: 」という単純なプロンプトです。 ここからは BLFS ブック に進んでいき、利用したいソフトウェアをいろいろと追加していくことができます。

再起動がうまく できなかった 場合は、解消していきます。 初期起動時の問題を解決するヒントとして、https://www.linuxfromscratch.org/lfs/troubleshooting.html を参考にしてください。

11.4. さらなる情報

本書をお読み頂き、ありがとうございます。 本書が皆さんにとって有用なものとなり、システムの構築方法について十分に学んで頂けたものと思います。

LFS システムをインストールしたら「次は何を？」とお考えになるかもしれません。 その質問に答えるために以下に各種の情報をまとめます。

- 保守

  あらゆるソフトウェアにおいて、バグやセキュリティの情報は日々報告されています。 LFS システムはソースコードからコンパイルしていますので、そのような報告を見逃さずにおくことは皆さんの仕事となります。 そのような報告をオンラインで提供する情報の場がありますので、いくつかを以下に示しましょう。

  - LFS セキュリティアドバイザリー

    LFS ブックを公開した後に発見されたセキュリティぜい弱性の一覧です。

  - オープンソースセキュリティメーリングリスト

    オープンソースコミュニティにおいて、セキュリティ不備、捉え方、実践などを議論するメーリングリストです。

- 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。 https://www.tldp.org/

11.5. LFS の次に向けて

11.5.1. 次に何をやるのか

ここに LFS が完成して起動可能なシステムを手にしました。 ここから何をしますか？ 次はこれをどう使うかを決めることです。 一般的には大きく２つの方法があります。 ワークステーションとするのかサーバーとするのかです。 実のところ、両者は別々とする必要はありません。 それぞれにとって必要となるアプリケーションは、同じシステム内に含めることができます。 もっとも以下では、それぞれを個別に見ていくことにします。

サーバーとすることは比較的簡単です。 一般には Apache HTTP Server のようなウェブサーバーと、MariaDB のようなデータベースサーバーから構成されます。 ただし他のサービスを含めても構いません。 使い捨てデバイスに埋め込まれているオペレーティングシステムは、ここに分類されます。

これに比べてワークステーションは、やや複雑です。 一般には LXDE, XFCE, KDE, Gnome といったグラフィカルユーザー環境が必要であり、これらは グラフィック環境 や Firefox ウェブブラウザー, Thunderbird Email クライアント, LibreOffice office スイート といったグラフィックベースのアプリケーションによって成り立っています。 こういったアプリケーションは、実に多くのパッケージ（所定機能の実現のために何百もの依存パッケージ）によるアプリケーションやライブラリを必要としています。

上に加えて、全システム向けにシステムを管理するアプリケーション群があります。 そういったアプリケーションは BLFS ブックに掲載しています。 環境による話であって、そのアプリケーションをすべて必要とするものではありません。 例として dhcpcd は、サーバーにおいては普通は不要のものですし、wireless_tools は、ラップトップシステムにのみ必要となるのが通常です。

11.5.2. 基本的な LFS 環境での作業

LFS を初めて起動すると、追加するパッケージをビルドするための内部ツールはすべて含まれています。 ただしユーザー環境は十分なものではありません。 これを充足させていくには、いくつかの方法があります。

11.5.2.1. LFS ホストからの chroot による作業

この方法を使えば、完全なグラフィック環境を扱うことができ、充実したブラウザーを利用してコピー/ペースト機能が活用できます。 またホスト内にある wget のようなアプリケーションを使うことができるため、パッケージソースをダウンロードして、chroot 環境内で作業可能な場所に配置することができます。

chroot 環境内で適切にパッケージビルドを行うためには、仮想ファイルシステムのマウントを忘れずに行っておく必要があります。 これを実現する１つの方法として、以下のようなスクリプトを HOST システム内に生成して利用することです。

```
cat > ~/mount-virt.sh << "EOF"
#!/bin/bash

function mountbind
{
   if ! mountpoint $LFS/$1 >/dev/null; then
     $SUDO mount --bind /$1 $LFS/$1
     echo $LFS/$1 mounted
   else
     echo $LFS/$1 already mounted
   fi
}

function mounttype
{
   if ! mountpoint $LFS/$1 >/dev/null; then
     $SUDO mount -t $2 $3 $4 $5 $LFS/$1
     echo $LFS/$1 mounted
   else
     echo $LFS/$1 already mounted
   fi
}

if [ $EUID -ne 0 ]; then
  SUDO=sudo
else
  SUDO=""
fi

if [ x$LFS == x ]; then
  echo "LFS not set"
  exit 1
fi

mountbind dev
mounttype dev/pts devpts devpts -o gid=5,mode=620
mounttype proc    proc   proc
mounttype sys     sysfs  sysfs
mounttype run     tmpfs  run
if [ -h $LFS/dev/shm ]; then
  install -v -d -m 1777 $LFS$(realpath /dev/shm)
else
  mounttype dev/shm tmpfs tmpfs -o nosuid,nodev
fi

#mountbind usr/src
#mountbind boot
#mountbind home
EOF
```

なおこのスクリプト内の最後の３つのコマンドはコメントアウトしています。 こういったディレクトリがホストシステム上の個別パーティションにマウントされていて、LFS/BLFS システムの起動時にマウントする必要がある場合に利用します。

このスクリプトは、一般ユーザー（これを推奨）または root ユーザーにて bash ~/mount-virt.sh として実行します。 一般ユーザーとして実行する場合には、ホストシステム上に sudo が必要です。

もう一つ、このスクリプトにおいて指摘するポイントとして、ダウンロードしたパッケージファイルをどこに保存するのかという点があります。 その場所については任意です。 たとえば一般ユーザーのホームディレクトリ配下の ~/sources といった場所にすることができます。 あるいはグローバルな場所として /usr/src とすることもできます。 ここで推奨したいのは、(chroot 環境から見て) /sources といったディレクトリに、BLFS と LFS のソースを混ぜないようにすることです。 どのようにするにせよ、パッケージソースは chroot 環境内部からアクセスできるようにしなければなりません。

ここで紹介する機能の最後は、chroot 環境に入る手順を効率化することです。 これは、ホストシステム内のユーザー向け ~/.bashrc ファイルにエイリアスを設けることで実現します。

```
alias lfs='sudo /usr/sbin/chroot /mnt/lfs /usr/bin/env -i HOME=/root TERM="$TERM" PS1="\u:\w\\\\
PATH=/usr/bin:/usr/sbin /bin/bash --login'
```

このエイリアスは多少トリッキーなところがあります。 それはクォートと重複するバックスラッシュ文字があるところです。 これらは単一行にすべて記述しなければなりません。 上で示したコマンド記述は、見やすさを考慮して二行に分けているに過ぎません。

11.5.2.2. ssh 経由のリモート作業

この方法はグラフィック環境下においても利用できます。 まず何よりも sshd を LFS システムにインストールすることが必要です。 これは通常 chroot 環境にて行います。 また 2 つめのコンピューターも必要です。 この方法は、複雑な chroot 環境を必要としないことから、単純であるという利点があります。 追加導入するパッケージに対しても、LFS からビルドしたカーネルを用いていくことになるので、インストールパッケージに対しても完全なシステム構成を保証し続けることになります。

LFS システム上においてビルドするソースパッケージを scp コマンドによってアップロードすることができます。 LFS システム上に直接ダウンロードするのであれば、chroot 環境内において libtasn1, p11-kit, make-ca, wget をインストールしてください。 (あるいは LFS システムの起動後に、それらのソースを scp を使ってアップロードしてください。)

11.5.2.3. LFS コマンドラインからの作業

この方法を用いるには chroot 環境において libtasn1, p11-kit, make-ca, wget, gpm, links (または lynx) をインストールしておき、再起動して新たな LFS システムに入ることが必要です。 その時点において、システムにはデフォルトで 6 つの仮想コンソールが存在します。 コンソールの切り替えは簡単で、 Alt+Fx のキー組み合わせを利用します。 ここで Fx は F1 から F6 までのキーを表します。 別のキー組み合わせ Alt+← と Alt+→ を使ってコンソールを切り替えることもできます。

この後に 2 つの異なる仮想コンソールにログインして、1 つのコンソール上では links または lynx ブラウザーを開き、もう 1 つのコンソールでは bash を起動します。 GPM があることで、ブラウザー上のコマンドを左マウスボタンによりコピーすることができます。 したがってコンソールを移って、そのコマンドをペーストすることができます。

注記

注記にして示しておくと、X Windows インスタンスから仮想コンソールを切り替えるには、 Ctrl+Alt+Fx のキー組み合わせを用います。 ただしマウスによるコピー操作は、グラフィックインターフェースと仮想コンソール間では動作しません。 X Windows ディスプレイに戻るため Ctrl+Alt+Fx の組み合わせを用いてください。 ここで Fx は一般的には F1 ですが F7 の場合もあります。

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。 逆に訳語に適当なものがないと思われるものは、無理に訳出せず原語だけを示すことにします。 この判断はあくまで訳者によるものであるため、不適切・不十分な個所についてはご指摘ください。

ABI	アプリケーション バイナリ インターフェース (Application Binary Interface)
ALFS	Automated Linux From Scratch
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; 以前
- Jamenson Espindula <jafesp@gmail.com> - ポルトガルの LFS 翻訳プロジェクト; 2022年から現在
- 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

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 McMurchy <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。

依存関係として4つめに示すのは任意のパッケージであり LFS では説明していないものです。 しかし皆さんにとっては有用なパッケージであるはずです。 それらのパッケージは、さらに別のパッケージを必要としていたり、互いに依存し合っていることがあります。 そういった依存関係があるため、それらをインストールする場合には、LFS をすべて仕上げた後に再度 LFS 内のパッケージを再構築する方法をお勧めします。 再インストールに関しては、たいていは BLFS にて説明しています。

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, Patch
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, Coreutils
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: Coreutils
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, Pkgconf, Sed, Texinfo, Zlib, Zstd
実行時依存パッケージ: Glibc, Zlib, Zstd
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, Jansson

Bison

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, 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, Libelf
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: Autoconf, Automake, 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, Findutils, Man-DB, Systemd
任意依存パッケージ: Expect.pm, IO::Tty

D-Bus

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Pkgconf, Sed, Systemd, Util-linux
実行時依存パッケージ: Glibc, Systemd
テストスイート依存パッケージ: BLFS におけるパッケージ数種
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 ライブラリ

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, Pkgconf, Sed, Systemd, 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

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

Flit-Core

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: Packaging, Wheel
任意依存パッケージ: pytest, testpath

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: GDC, 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: libunistring , libxml2

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, Pcre2, 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: なし

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: なし
事前インストールパッケージ: Man-DB
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, Pkgconf, 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, libbpf, libmnl, libtirpc

Jinja2

インストール依存パッケージ: MarkupSafe, Python, Setuptools, Wheel
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, 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: Linux-PAM

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, Pkgconf, 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: Systemd
任意依存パッケージ: scdoc (man ページのため)

Less

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Pcre2, 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: Gzip
任意依存パッケージ: なし

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, Bzip2, Coreutils, GCC, Glibc, Make, Xz, Zlib, Zstd
実行時依存パッケージ: Bzip2, Glibc, Xz, Zlib, Zstd
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: Pkgconf
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

Libxcrypt

インストール依存パッケージ: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, 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: なし
事前インストールパッケージ: Perl, Python, Shadow, Systemd
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, LLVM (Clang 込み), Rust-bindgen

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

Lz4

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
テストスイート依存パッケージ: Python
事前インストールパッケージ: Zstd, Systemd
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, Pkgconf, Sed, Systemd, 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, po4a

Man-Pages

インストール依存パッケージ: Bash, Coreutils, Make, 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

MarkupSafe

インストール依存パッケージ: Python, Setuptools, Wheel
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, Setuptools, Wheel
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: cmake
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, Systemd
任意依存パッケージ: なし

Packaging

インストール依存パッケージ: Flit-core, 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: Wheel
任意依存パッケージ: pytest

Patch

インストール依存パッケージ: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed
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: 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

Pcre2

インストール依存パッケージ: Bash, Binutils, Bzip2, Coreutils, GCC, Glibc, GZip, Make, Readline
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: Grep
事前インストールパッケージ: Grep, Less
任意依存パッケージ: Valgrind, libedit

Perl

インストール依存パッケージ: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Libxcrypt, Make, Sed, Zlib
実行時依存パッケージ: GDBM, Glibc, Libxcrypt
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

Pkgconf

インストール依存パッケージ: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed, Sqlite
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: Binutils, D-Bus, E2fsprogs, IProute2, Kmod, Man-DB, Procps-ng, Python, Systemd, Util-linux
任意依存パッケージ: なし

Procps-ng

インストール依存パッケージ: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses, Pkgconf, Systemd
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: 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

Python

インストール依存パッケージ: Bash, Binutils, Coreutils, Expat, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Libxcrypt, Make, Ncurses, OpenSSL, Pkgconf, Sed, Util-linux
実行時依存パッケージ: Bzip2, Expat, Gdbm, Glibc, Libffi, Libxcrypt, 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, 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

Setuptools

インストール依存パッケージ: Python, Wheel
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, MarkupSafe, Meson
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, Libxcrypt, Make, Sed
実行時依存パッケージ: Glibc, Libxcrypt
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

Pcre2

インストール依存パッケージ: Bash, Binutils, GCC, Glibc, Gzip, Make, Ncurses, Readline
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
任意依存パッケージ: libarchive, libedit

Systemd

インストール依存パッケージ: Acl, Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Gperf, Grep, Jinja2, Libcap, Libxcrypt, Lz4, Meson, OpenSSL, Pcre2, Pkgconf, Sed, Util-linux, Zstd
実行時依存パッケージ: Acl, Glibc, Libcap, Libxcrypt, OpenSSL, Util-linux, Xz, Zlib, Zstd
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: D-Bus, E2fsprogs, Man-DB, Procps-ng, Util-linux
任意依存パッケージ: AppArmor, audit-userspace, bash-completion, btrfs-progs, cURL, cryptsetup, docbook-xml, docbook-xsl-nons, Git, GnuTLS, iptables, jekyll, kexec-tools, libbpf, libdw, libfido2, libgcrypt, libidn2, libmicrohttpd, libpwquality, libseccomp, libxkbcommon, libxslt, Linux-PAM, lxml, make-ca, p11-kit, pefile, Polkit, pyelftools, qemu, qrencode, quota-tools, rpm, rsync, SELinux, Sphinx, systemtap, tpm2-tss, Valgrind, Xen, zsh

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, File, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Pkgconf, Sed, Systemd, Zlib
実行時依存パッケージ: Glibc, Ncurses, Readline, Systemd, 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: Asciidoctor, Libcap-NG, libeconf, libuser, libutempter, Linux-PAM, smartmontools, po4a, slang

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, Flit-core, packaging
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, MarkupSafe, Meson, Setuptools
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: LWP::UserAgent

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: File, GRUB, Kmod, Libelf, Man-DB, Systemd
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, Libelf, 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, Lz4, Make, Xz, Zlib
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: Binutils, GCC, Libelf, Systemd
任意依存パッケージ: なし

付録D 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。

D.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. 6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

重要

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/.

D.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 © 1999-2025 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

プログラム

ライブラリ

スクリプト

その他