Linux From Scratch

版本 12.4-systemd-中文翻译版

发布于 2025 年 9 月 1 日

由 Gerard Beekmans 原著
总编辑：Bruce Dubbs
编辑：Douglas R. Reno
编辑：DJ Lucas

Linux From Scratch: 版本 12.4-systemd-中文翻译版: 发布于 2025 年 9 月 1 日

由 由 Gerard Beekmans 原著、总编辑：Bruce Dubbs、编辑：Douglas R. Reno和编辑：DJ Lucas

目录

序言

前言

从 1998 年起，我踏上了学习和深入理解 Linux 的旅程。当时我刚刚安装了我的第一个 Linux 发行版，并迅速被 Linux 背后的整个设计理念和哲学所折服。

为了完成一项工作，人们总是能提出很多不同的方法。对于 Linux 发行版来说，情况也是这样。多年来诞生了许多发行版，其中一些仍然生存，另外一些已经被其他发行版吸收，还有的已经消亡，成为我们的回忆。这些发行版各有特色，以满足它们的目标人群的需求。因为这些发行版都是已经存在的，能够达成同一目的的手段，我开始意识到并不需要将自己的思维约束在发行版这一种实现方法上。在发现 Linux 之前，我们只能忍受其他操作系统的种种不足，因为我们没有其他选择，操作系统的行为不以我们的意志为转移。然而，自由选择的理念随着 Linux 的诞生而出现。如果你不喜欢某种行为，就可以自由地改变它。这在 Linux 世界中甚至是受到鼓励的。

我曾经尝试了许多发行版，但无法做出最终决定。它们各有特色，都是很不错的系统。这里不存在对与错的问题，而是系统是否符合个人口味的问题。在各种选择中，看上去并没有一种发行版能完美地符合我的要求。因此我开始创造自己的 Linux 系统，以完全符合我的个人品味。

为了构建出真正属于我自己的系统，我决定从源代码编译所有东西，而不使用预先编译的二进制包。这个“完美的” Linux 系统将会兼具不同系统的优点，同时扬弃它们的不足。这个想法初听上去非常可怕。然而，我仍然坚信这个系统可以被构建出来。

在整理并解决了循环依赖和编译错误等问题后，我终于构建出自己定制的 Linux 系统。它完全可以工作，并且像当时的其他 Linux 系统一样完美可用。不同的是，这是我自己的创造，亲手组装出这样的系统是非常有成就感的。唯一能够让我更开心的事情是亲自编写一个软件系统。

当我向其他 Linux 社区成员推广我的目标和经验时，大家似乎对这些想法很有兴趣。显而易见，这些自行定制的 Linux 系统不仅能够满足用户的特殊需求，而且对于程序员和系统管理员来说是提高 Linux 技能的理想学习机会。随着越来越多的人对这一主题的关注，Linux From Scratch 项目诞生了。

这本 Linux From Scratch 手册是这一项目的核心内容，它将提供亲自设计和构建系统所需的背景知识和操作步骤。本书提供了一个构建能够正常工作的系统的样板，您可以自由地调整本书中的命令，来满足您自己的要求，这也是本项目的重要组成部分。您始终掌握自己的系统，我们只是在您起步时提供微小的帮助。

我真诚地祝愿您能够在您自己的 Linux From Scratch 系统上体验快乐，并享受拥有这样一个真正属于自己的系统所带来的各种乐趣。

--
Gerard Beekmans
gerard@linuxfromscratch.org

本书面向的读者

您可能有许多阅读本书的理由。许多人首先会问：“为什么要不辞辛苦地手工从头构建一个 Linux 系统，而不是直接下载并且安装一个现成的？”

LFS 项目存在的一项重要原因是，它能够帮助您学习 Linux 系统的内部是如何运作的。构建 LFS 系统的过程将展示 Linux 系统的工作原理，以及其各组成部分的协作和依赖关系。最棒的是，有了这些经验，您将能够定制 Linux 系统，使其满足您独一无二的需求。

LFS 的另一个关键优势是，它允许您掌控您的系统，而不用依赖于其他人的 Linux 实现。在使用 LFS 时，您就像坐在驾驶座上一样，亲自掌控系统的各个部分。

LFS 允许您创建非常紧凑的 Linux 系统。在安装其他发行版时，您往往不得不安装一大堆永远不会用到或者意义不明的程序。它们会浪费系统资源。您可能以为，有了现代的大容量硬盘和高速 CPU，就不需要考虑资源浪费的问题。然而，在一些情况下，即使不考虑其他问题，仅仅存储空间的约束就十分紧张。典型的代表有可引导 CD，USB 启动盘，以及嵌入式系统。在这些领域中，LFS 是十分有用的。

自行定制的 Linux 系统在安全方面也具有优势。在从源码编译整个系统的过程中，您有机会审核所有的代码，并安装您需要的安全补丁。您不需要像往常那样等待其他人编译一个修复了安全漏洞的二进制包。另外，除非您亲自检查并应用了补丁，您无法保证新的二进制包在编译过程中没有出问题，并且正确修补了安全漏洞。

Linux From Scratch 的目标是构建一个完整并基本可用的系统。如果您不想从零构建您自己的 Linux 系统，那么您可能不会从本书提供的信息中受益。

此外，构建 LFS 系统还有很多好处，这里就不一一列举了。总而言之，学习仍然是最重要的使用 LFS 的原因。在您编译和使用 LFS 的实践过程中，您将学到很多威力巨大的信息和知识。

LFS 的目标架构

LFS 的主要目标架构是 AMD/Intel 的 x86 (32 位) 和 x86_64 (64 位) CPU。此外，如果对本书中的一些指令作适当的修改，它们也应该适用于 Power PC 和 ARM 架构的 CPU。无论在其中哪种 CPU 上，构建 LFS 都至少需要一个现有的 Linux 系统，例如已经构建好的 LFS 系统，Ubuntu，Rad Hat/Fedora，SuSE，或者其他支持您的硬件架构的发行版，后文中还会介绍其他前提条件。(另外，32 位发行版也能在 64 位的 AMD/Intel 计算机上正常运行，并作为 LFS 的构建环境。)

构建 64 位系统相较于 32 位系统而言只会获得很小的收益。例如，在使用 Core i7-4790 CPU 的 4 个 CPU 核心测试构建 LFS-9.1 时，我们得到的实验数据为：

```
架构          构建时间         系统大小
32 位         239.9 分钟       3.6 GB
64 位         233.2 分钟       4.4 GB
```

可以看出，在相同的硬件上，64 位系统的构建仅仅比 32 位快 3% (但占用的磁盘空间却大 22%)。如果您准备用 LFS 系统运行 LAMP 服务器，或者防火墙，那么 32 位 CPU 足以满足需求。然而，BLFS 中的一些软件包在构建或运行过程中可能需要超过 4GB 的内存，因此如果您准备将 LFS 作为桌面系统，LFS 作者推荐构建 64 位系统。

完全按照本书构建的 LFS 系统是一个“纯粹的” 64 位系统。换句话说，它只能运行 64 位可执行程序。构建一个“multi-lib” 系统需要将许多应用程序编译两次，一次编译为 32 位，另一次编译为 64 位。本书不提供这方面的内容，因为本书的教学目的是提供简洁的基本 Linux 系统的构建方法，讨论 multilib 会和这一目标发生冲突。一些 LFS/BLFS 编辑维护了 LFS 的 multilib 版本，可以在 https://www.linuxfromscratch.org/~thomas/multilib/index.html 查阅。但这是一个比较复杂的主题。

阅读本书需要的背景知识

构建 LFS 系统不是一项简单的任务。它需要您运用足够丰富的 Unix 系统管理知识来解决构建过程中的问题，并正确执行本书给出的命令。特别是，您至少需要知道如何使用命令行 (shell) 来复制或移动文件和目录，列出目录或文件的内容，以及切换当前工作目录。另外，我们预期您知道如何使用和安装 Linux 软件。

由于本书假定您至少具备上述基本技能，任何 LFS 支持论坛不太可能在这些领域为您提供帮助。关于这些基础知识的问题一般会被忽略 (或者被提供一份 LFS 背景知识预习书单作为答案)。

在您开始构建 LFS 系统之前，我们建议您阅读下列材料：

- Software-Building-HOWTO https://tldp.org/HOWTO/Software-Building-HOWTO.html

  这是一份关于在 Linux 环境编译和安装“常规的”Unix 软件包的详细指南。虽然这份文档比较老，但是它较好地总结了编译和安装软件的基本技巧。

· Beginner's Guide to Installing from Source https://moi.vonos.net/linux/beginners-installing-from-source/

这份指南很好地总结了从源代码编译软件的基本技能和技巧。

LFS 和标准

LFS 的结构尽可能遵循 Linux 的各项标准。主要的标准有：

· POSIX.1-2008.

· Filesystem Hierarchy Standard (FHS) Version 3.0

· Linux Standard Base (LSB) Version 5.0 (2015)

LSB 由四个独立的规范组成：Core，Desktop，Languages，以及 Imaging。Core 和 Desktop 规范中一些部分是架构相关的。另外，还有两个处于试用阶段的规范：Gtk3 和 Graphics。LFS 试图遵循 LSB 对前一节讨论的 IA32 (32 位 x86) 和 AMD64 (x86_64) 架构的要求。

注意

许多人不认同 LSB 的要求。定义 LSB 的主要目的是保证专有软件能够在满足 LSB 的系统上正常安装并运行。然而 LFS 是基于源代码的，用户完全知道自己需要什么软件包；您可以选择不安装 LSB 要求的一些软件包。

“从零开始”创建一个能够通过 LSB 认证测试的完整系统是可行的，但需要安装大量超过 LFS 范畴的额外软件包。在 BLFS 中可以找到其中一些软件包的安装说明。

LSB 要求的，由 LFS 提供的软件包

LSB Core:	Bash, Bc, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Systemd, Tar, Util-linux, Zlib
LSB Desktop:	无
LSB Languages:	Perl
LSB Imaging:	无
LSB Gtk3 和 LSB Graphics (试用):	无

LSB 要求的，由 BLFS 提供的软件包

LSB Core:	At, Batch (At 的一部分), BLFS Bash 启动文件, Cpio, Ed, Fcrontab, LSB-Tools, NSPR, NSS, Linux-PAM, Pax, Sendmail (或 Postfix，或 Exim), Time
LSB Desktop:	Alsa, ATK, Cairo, Desktop-file-utils, Freetype, Fontconfig, Gdk-pixbuf, Glib2, GTK+2, Icon-naming-utils, Libjpeg-turbo, Libxml2, Mesa, Pango, Xdg-utils, Xorg
LSB Languages:	Libxml2, Libxslt
LSB Imaging:	CUPS, Cups-filters, Ghostscript, SANE
LSB Gtk3 和 LSB Graphics (试用):	GTK+3

LSB 要求的，LFS 和 BLFS 均不提供或仅可选提供的组件

LSB Core:	**install_initd**, `libcrypt.so.1` (可以按照 LFS Libxcrypt 软件包的可选说明提供), `libncurses.so.5` (可以按照 LFS Ncurses 软件包的可选说明提供), `libncursesw.so.5` (但 LFS Ncurses 软件包提供了 `libncursesw.so.6`)
LSB Desktop:	`libgdk-x11-2.0.so` (但 BLFS GTK+-3 软件包提供了 `libgdk-3.so`), `libgtk-x11-2.0.so` (但 BLFS GTK+-3 软件包和 GTK-4 软件包提供了 `libgtk-3.so` 和 `libgtk-4.so`), `libpng12.so` (但 BLFS libpng 软件包提供了 `libpng16.so`), `libQt*.so.4` (但 BLFS Qt6 软件包提供了 `libQt6*.so.6`), `libtiff.so.4` (但 BLFS libtiff 软件包提供了 `libtiff.so.6`)
LSB Languages:	**/usr/bin/python** (LSB 要求 Python2，但 LFS 和 BLFS 只提供 Python3)
LSB Imaging:	无
LSB Gtk3 和 LSB Graphics (试用):	`libpng15.so` (但 BLFS libpng 软件包提供了 `libpng16.so`)

本书选择软件包的逻辑

LFS 的目标是构建一个完整且基本可用的系统 —— 包括所有再次构建 LFS 系统本身所需的软件包 —— 以便用户在这个相对较小的系统基础上，根据自己的选择，定制一个更完整的系统。但是，LFS 并不是最小可用系统。严格来说，LFS 中一些软件包并不是必须安装的。下面列出了选择每个软件包的理由。

· Acl

这个软件包包含管理访问控制列表 (ACL) 的工具，用来对文件和目录提供细粒度的访问权限控制。

· Attr

这个软件包包含管理文件系统对象的扩展属性的程序。

· Autoconf

这个软件包提供能根据软件开发者提供的模板，自动生成配置源代码的 shell 脚本的程序。如果修改了软件包的构建过程，一般需要该软件包的支持才能重新构建被修改的软件包。

· Automake

这个软件包包含能根据软件开发者提供的模板，自动生成 Makefile 的程序。如果修改了软件包的构建过程，一般需要该软件包的支持才能重新构建被修改的软件包。

· Bash

这个软件包为系统提供一个 LSB core 要求的 Bourne Shell 接口。它是较为常用的 shell 软件包，且具有一定的的扩展能力，因此在各种 shell 软件包中选择了它。

· Bc

这个软件包提供了一个任意精度数值处理语言。在编译 Linux 内核时需要该软件包。

· Binutils

该软件包提供链接器、汇编器，以及其他处理目标文件的工具。编译 LFS 系统中的大多数软件包都需要这些程序。

· Bison

这个软件包包含 yacc (Yet Another Compiler Compiler) 的 GNU 版本。一些 LFS 程序的编译过程需要该软件包。

· Bzip2

这个软件包包含用于压缩和解压缩文件的程序。许多 LFS 软件包的解压需要该软件包。

· Check

这个软件包提供其他程序使用的测试环境。

· Coreutils

这个软件包包含一些用于查看和操作文件和目录的基本程序。这些程序被用于在命令行下管理文件，以及每个 LFS 软件包的安装过程。

· D-Bus

这个软件包包含用于提供消息总线系统的程序，作为一种应用程序之间通信的简单方式。

· DejaGNU

这个软件包提供用于测试其他程序的框架。

· Diffutils

这个软件包包含用于显示文件或目录之间的差异的程序。这些程序可以被用于创建补丁，很多软件包的编译过程也需要该软件包。

· E2fsprogs

这个软件包提供用于处理 ext2, ext3 和 ext4 文件系统的工具。它们是 Linux 支持的最常用且久经考验的文件系统。

· Expat

这个软件包提供一个相对轻量级的 XML 解析库。Perl 模块 XML::Parser 需要该软件包。

· Expect

这个软件包包含一个自动和其他交互程序交互的脚本执行程序。一般用它测试其他程序。

· File

这个软件包包含用于判定给定文件类型的工具。一些软件包的构建脚本需要它。

· Findutils

这个软件包提供用于在文件系统中寻找文件的程序。它被许多软件包的编译脚本使用。

· Flex

这个软件包包含用于生成词法分析器的程序。它是 lex (lexical analyzer) 程序的 GNU 版本。许多 LFS 软件包的编译过程需要该软件包。

· Gawk

这个软件包提供用于操作文本文件的程序。它是 awk (Aho-Weinberg-Kernighan) 的 GNU 版本。它被许多其他软件包的构建脚本使用。

· GCC

这是 GNU 编译器的集合。它包含 C 和 C++ 编译器，以及其他一些在 LFS 中不会涉及的编译器。

· GDBM

这个软件包包含 GNU 数据库管理库。LFS 的另一个软件包 Man-DB 需要该软件包。

· Gettext

这个软件包提供用于许多其他软件包的国际化和本地化的工具和库。

· Glibc

这个软件包包含主要的 C 语言库。Linux 程序没有该软件包的支持根本无法运行。

· GMP

这个软件包提供数学库，这些库支持用于任意精度算术的函数。编译 GCC 需要该软件包。

· Gperf

这个软件包提供一个能够根据键值集合生成完美散列函数的程序。 Systemd 需要该软件包。

· Grep

这个软件包包含在文本中搜索指定模式的程序。它被多数软件包的编译脚本所使用。

· Groff

这个软件包提供用于处理和格式化文本的程序。它们的一项重要功能是格式化手册页。

· GRUB

这个软件包是 Grand Unified Boot Loader。Linux 可以使用其他引导加载器，但 GRUB 最灵活。

· Gzip

这个软件包包含用于压缩和解压缩文件的程序。许多 LFS 软件包的解压需要该软件包。

· Iana-etc

这个软件包包含网络服务和协议的描述数据。网络功能的正确运作需要该软件包。

· Inetutils

这个软件包提供基本网络管理程序。

· Intltool

这个软件包提供能够从源代码中提取可翻译字符串的工具。

· IProute2

这个软件包提供了用于 IPv4 和 IPv6 网络的基础和高级管理程序。和另一个常见的网络工具包 net-tools 相比，它具有管理 IPv6 网络的能力。

· Jinja2

该软件包是一个处理文本文件模板的 Python 模块。构建 Systemd 需要它。

· Kbd

这个软件包提供键盘映射文件，用于非美式键盘的键盘工具，以及一些控制台字体。

· Kmod

这个软件包提供用于管理 Linux 内核模块的程序。

· Less

这个软件包包含一个很好的文本文件查看器，它支持在查看文件时上下滚动。许多软件包使用它对输出进行分页。

· Libcap

这个软件包实现了用于访问 Linux 内核中 POSIX 1003.1e 权能字功能的用户空间接口。

· Libelf

Elfutils 项目提供了用于 ELF 文件和 DWARF 数据的工具和库。该软件包的大多数工具已经由其他软件包提供，但使用默认 (也是最高效的) 配置构建 Linux 内核时，需要使用该软件包的库。

· Libffi

这个软件包实现了一个可移植的高级编程接口，用于处理不同的调用惯例。某些程序在编译时并不知道如何向函数传递参数，例如解释器在运行时才得到函数的参数个数和类型信息。它们可以使用 libffi 作为解释语言和编译语言之间的桥梁。

· Libpipeline

Libpipeline 提供一个能够灵活、方便地操作子进程流水线的库。Man-DB 软件包需要这个库。

· Libtool

这个软件包包含 GNU 通用库支持脚本。它将共享库的使用封装成一个一致、可移植的接口。在其他 LFS 软件包的测试套件中需要该软件包。

· Libxcrypt

该软件包提供 `libcrypt` 库，一些软件包 (例如 Shadow) 使用该库对密码进行散列操作。它替代 Glibc 中过时的 `libcrypt` 实现。

· Linux Kernel

这个软件包就是操作系统。我们平常说的 “GNU/Linux” 环境中的 “Linux” 就指的是它。

· M4

这个软件包提供通用的文本宏处理器。它被其他程序用作构建工具。

· Make

这个软件包包含用于指导软件包编译过程的程序。LFS 中几乎每个软件包都需要它。

· MarkupSafe

该软件包是一个安全地处理 HTML/XHTML/XML 中的字符串的 Python 模块。Jinja2 需要该软件包。

· Man-DB

这个软件包包含用于查找和浏览手册页的程序。与 man 软件包相比，该软件包的国际化功能更为强大。该软件包提供了 man 程序。

· Man-pages

这个软件包提供基本的 Linux 手册页的实际内容。

· Meson

这个软件包一个自动化软件构建过程的工具。它的设计目标是最小化软件开发者不得不用于配置构建系统的时间。该软件包在构建 Systemd 和很多 BLFS 软件包时是必要的。

· MPC

这个软件包提供用于复数算术的函数。GCC 需要该软件包。

· MPFR

这个软件包包含用于多精度算术的函数。GCC 需要该软件包。

· Ninja

这个软件包提供一个注重执行速度的小型构建系统。它被设计为读取高级构建系统生成的输入文件，并以尽量高的速度运行。Meson 需要该软件包。

· Ncurses

这个软件包包含用于处理字符界面的不依赖特定终端的库。它一般被用于为菜单系统提供光标控制。一些 LFS 软件包需要该软件包。

· Openssl

这个软件包包含关于密码学的管理工具和库，它们为 Linux 内核等其他软件包提供密码学功能。

· Patch

这个软件包包含一个通过 补丁 文件修改或创建文件的程序。补丁文件通常由 diff 程序创建。一些 LFS 软件包的编译过程需要该软件包。

· Perl

这个软件包是运行时语言 PERL 的解释器。几个 LFS 软件包的安装和测试过程需要该软件包。

· Pkgconf

这个软件包包含一个为开发库配置编译和链接选项的程序。该程序可以直接替代 **pkg-config** 命令，许多软件包的构建系统都需要该命令。它的维护比原始的 Pkg-config 软件包更积极，而且运行速度稍快一些。

· Procps-NG

这个软件包包含用于监控系统进程的程序，对系统管理非常有用。另外 LFS 引导脚本也需要该软件包。

· Psmisc

这个软件包提供一些显示当前运行的系统进程信息的程序。这些程序对系统管理非常有用。

· Python 3

这个软件包提供了一种在设计时强调代码可读性的解释性语言支持。

· Readline

这个软件包是一组库，提供命令行编辑和历史记录支持。Bash 需要该软件包。

· Sed

这个软件包可以在没有文本编辑器的情况下编辑文本文件。另外，许多 LFS 软件包的配置脚本需要该软件包。

· Shadow

这个软件包包含用于安全地处理密码的程序。

· Systemd

这个软件包提供一个init程序，和一些附加的引导和系统控制支持。它能够替代 SysVinit。许多商业发行版使用该软件包。

· Tar

这个软件包提供存档和提取功能，几乎每个 LFS 软件包都需要它才能被提取。

· Tcl

这个软件包包含在测试套件中广泛使用的工具控制语言 (Tool Command Language)。

· Texinfo

这个软件包提供用于阅读、编写和转换 info 页面的程序。许多 LFS 软件包的安装过程需要使用它。

· Util-linux

这个软件包包含许多工具程序，其中有处理文件系统、终端、分区和消息的工具。

· Vim

这个软件包提供一个编辑器。由于它与经典的 vi 编辑器相兼容，且拥有许多强大的功能，我们选择这个编辑器。编辑器的选择是非常主观的。如果希望的话，读者可以用其他编辑器替代它。

· Wheel

该软件包提供一个 Python 模块，该模块是 Python wheel 软件包标准格式的参考实现。

· XML::Parser

这个软件包是和 Expat 交互的 Perl 模块。

· XZ Utils

这个软件包包含用于压缩和解压缩文件的程序。在所有这类程序中，该软件包提供了最高的压缩率。该软件包被用于解压 XZ 或 LZMA 格式的压缩文件。

· Zlib

这个软件包包含一些程序使用的压缩和解压缩子程序。

· Zstd

这个软件包提供一些程序使用的压缩和解压缩子程序。它具有较高的压缩比，以及很宽的压缩比/速度权衡范围。

排版约定

为了使得本书更容易阅读，首先说明本书的排版惯例。本节包含本书中若干排版格式的示例。

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

类似上面这样排版的文字应当被绝对准确地输入，除非上下文另有说明。在解释命令的含义时，我们也用这种格式给出被解释的命令。

有时，我们会将一个逻辑行拆分成两行或者多行，此时行末需要使用反斜线。

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

请注意反斜线之后必须紧跟换行符。反斜线后如果存在空格或者制表符等其他空白字符，会导致不正确的结果。

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

以上格式的文本 (等宽字体) 展示屏幕输出，通常是某个命令执行的结果。如果正在阅读 HTML 格式 (而非 PDF) 的手册，这些文本应该是蓝色的。

等宽字体文本也用于展示文件名，例如 `/etc/ld.so.conf`。

注意

请配置浏览器以使用合适的等宽字体，以便区分 `Il1` 或 `O0` 的字形。

强调的文本

以上格式的文本在本书中被用于一些目的。主要目的是强调重点。

https://www.linuxfromscratch.org/

以上格式的文本是超链接，可能指向 LFS 社区内部或外部的页面。外部页面包括 HOWTO，下载地址，以及网站。

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

这种格式在创建配置文件时使用，第一行的命令告诉系统使用键盘输入的后续各行内容创建 `$LFS/etc/group` 文件，直到遇到文件结束序列 (EOF)。因此，通常应该将整段命令原封不动地输入。

`<需要替换的文本>`

不应该直接输入或复制粘贴这种尖括号包含的文本，而应该将其替换成合适内容。

`[可选的文本]`

方括号包含的文本是可选的，根据您的需要决定是否输入。

passwd(5)

以上格式被用于引用特定的手册页 (man page)。数字表明页面来自系统手册中的某一节。例如，**passwd** 有两个手册页。LFS 安装命令将它们安装在 `/usr/share/man/man1/passwd.1` 和 `/usr/share/man/man5/passwd.5`。当本书使用 passwd(5) 时，它特指手册页 `/usr/share/man/man5/passwd.5`。**man passwd** 会显示它找到的第一个匹配 “passwd” 的手册页，即 `/usr/share/man/man1/passwd.1`。对于本例，您需要执行 **man 5 passwd** 才能阅读指定的手册页。多数手册页在各个章节中并不存在重名。因此，**man** **`<程序名>`** 一般是足够的。在本书中，这些对手册页的引用同时也是超链接，因此点击它们时，会打开 Arch Linux 手册页 提供的，已渲染为 HTML 的手册页。

本书结构

本书被分为以下三个部分。

第一部分 - 引言

第一部分解释了一些安装 LFS 时的重要注意事项。同时，提供了本书的基本信息。

第二部分 - 准备工作

第二部分描述了如何进行构建的准备工作，包括分区、下载软件包、编译临时工具链等。

第三部分 - 构建 LFS 交叉工具链和临时工具

第三部分提供在最终构建 LFS 系统时需要使用的工具的构建方法。

第四部分 - 构建 LFS 系统

第四部分引导读者完成 LFS 系统的构建 — 逐个安装和编译所有需要的软件包，设定引导脚本，以及安装内核。得到的 Linux 系统是一个基本系统，在它之上可以继续编译其他软件，以扩展系统，更好地满足需求。在本书的最后，给出了一个便于使用的引用列表，包括本书中安装的所有软件、库和其他重要文件。

第五部分 - 附录

第五部分提供关于本书本身的信息，如缩写和用语，致谢，软件包依赖信息，LFS 引导脚本的代码清单，本书的许可和发行信息，以及软件包、程序、库和引导脚本的完整索引。

勘误和安全公告

用于构建 LFS 系统的软件处于不断更新和改进的过程中。在本书发布后，一些软件可能公布安全警告和漏洞修复补丁。为了确认本书提供的软件包版本或者构建命令是否需要修正 —— 以修复安全漏洞或其他 bug，请在开始构建 LFS 之前阅读 https://www.linuxfromscratch.org/lfs/errata/12.4-systemd/。请记录勘误表列出的所有修正项，并据此在构建过程中对本书相关章节给出的操作进行修正。

另外，Linux From Scratch 编辑维护了在手册发布后发现的安全缺陷列表。在进行构建前，请访问 https://www.linuxfromscratch.org/lfs/advisories/ 以检查是否存在需要处理的安全缺陷。请根据安全公告的建议，在构建过程中对本书相关章节给出的操作进行修正。另外，如果要将 LFS 系统实际用作桌面或服务器系统，那么即使在 LFS 系统构建完成后，也要继续关注安全公告并修复列出的所有安全缺陷。

第 I 部分 概述

第 1 章 概述

1.1. 如何构建 LFS 系统

LFS 系统必须在一个已经安装好的 Linux 发行版 (如 Debian、OpenMandriva、Fedora 或者 openSUSE) 中构建。这个安装好的 Linux 系统 (称为宿主) 提供包括编译器、链接器和 shell 在内的必要程序，作为构建新系统的起点。请在安装发行版的过程中选择“development” (开发) 选项，以保证系统包含这些工具。

注意

有许多安装 Linux 发行版的方式，但默认方式对于构建 LFS 来说往往不是最好的。如果需要有关安装商业发行版的建议，参阅：https://www.linuxfromscratch.org/hints/downloads/files/partitioning-for-lfs.txt.

除了安装一个完整的发行版外，也可以使用某个商业发行版的 LiveCD。

本书的第 2 章描述了如何创建一个新的 Linux 本地分区和文件系统，以在其中编译和安装新的 LFS 系统。第 3 章列举了在构建 LFS 系统的过程中必须下载的软件包和补丁，并解释了在新文件系统中存储它们的方法。第 4 章讨论工作环境的正确配置。请仔细阅读第 4 章，因为它解释了您在开始第 5 章及后续章节的工作前必须了解的一些重要问题。

第 5 章解释初始工具链 (binutils, gcc, 以及 glibc) 的安装过程，在安装过程中使用交叉编译技术，将新的工具与宿主系统完全隔离。

第 6 章向您展示如何使用刚刚构建的交叉工具链，交叉编译一些基本工具。

之后在第 7 章中，进入一个 "chroot" 环境，在其中使用刚刚构建的工具，继续构建 LFS 构建过程中需要的其余工具。

我们努力将新构造的系统从宿主发行版分离出来。这个过程看上去很繁琐，我们将会在工具链技术说明中完整地从技术上解释这样做的必要性。

在第 8 章中我们将构建完整的 LFS 系统。使用 chroot 环境的另一项优势是，在构建 LFS 的过程中，您可以继续使用宿主系统。这样，在等待软件包编译的过程中，您可以继续正常使用计算机。

为了完成安装，我们在第 9 章中进行系统的基本设置，在第 10 章中配置内核和引导加载器。最后，第 11 章包含在阅读完本书后继续体验 LFS 的相关信息。在完成本书的所有流程后，重启计算机即可进入新的 LFS 系统。

以上是 LFS 构建过程的简要介绍，针对特定步骤的详细信息将在之后章节以及软件包的简介中讨论。在您踏上 LFS 的构建之旅后，就能逐步理清这些目前看上去很复杂的步骤，每一步都将变得非常清晰。

1.2. 自上次发布以来的更新

这里列出本书上一次发布之后发生变化的软件包。

已升级：

·

· Automake-1.18.1
· Bash-5.3
· Binutils-2.45
· Coreutils-9.7
· D-Bus-1.16.2

· Diffutils-3.12
· E2fsprogs-1.47.3
· Expat-2.7.1
· Flit-Core-3.12.0
· Gawk-5.3.2
· GCC-15.2.0
· GDBM-1.26
· Gettext-0.26
· Glibc-2.42
· Gperf-3.3
· Grep-3.12
· Gzip-1.14
· Iana-Etc-20250807
· IPRoute2-6.16.0
· Jinja2-3.1.6
· Kbd-2.8.0
· Kmod-34.2
· Less-679
· LFS-Bootscripts-20250827
· Libcap-2.76
· Elfutils-0.193 中的 Libelf
· Libffi-3.5.2
· Linux-6.16.1
· M4-1.4.20
· Man-DB-2.13.1
· Man-pages-6.15
· Meson-1.8.3
· MPFR-4.2.2
· Ncurses-6.5-20250809
· Ninja-1.13.1
· OpenSSL-3.5.2
· Patch-2.8
· Perl-5.42.0
· Pkgconf-2.5.1
· Python-3.13.7
· Readline-8.3
· Setuptools-80.9.0
· Shadow-4.18.0
· Systemd-257.8

· Tzdata-2025b
· Util-linux-2.41.1
· Vim-9.1.1629
· Wheel-0.46.1
· Xz-5.8.1

已添加：

·

· Packaging-25.0
· coreutils-9.7-upstream_fix-1.patch

已移除：

·

· Check-0.15.2

1.3. 更新日志

这是 Linux From Scratch 手册的 12.4-systemd 版本，发布于 2025 年 9 月 1 日。如果该版本已经发布了六个月或更久，可能已经发布了更好的新版本。如果要查询是否有新版本，通过 https://www.linuxfromscratch.org/mirrors.html 访问一个 LFS 镜像站。

下面是本书自上一版本发布以来的更新日志。

更新日志记录：

· 2025 年 9 月 1 日
  · [bdubbs] — LFS-12.4 发布。
· 2025 年 8 月 15 日
  · [bdubbs] — 引入上游对 glibc 进行的一项修改，以修复与 valgrind 不兼容的问题。修复 #5778。
  · [bdubbs] — 更新到 iana-etc-20250807。处理 #5006。
  · [bdubbs] — 更新到 vim-9.1.1829。处理 #4500。
  · [bdubbs] — 更新到 ncurses-6.5-20250809。修复 #5780。
  · [bdubbs] — 更新到 Python-3.13.7 (安全更新)。修复 #5779。
  · [bdubbs] — 更新到 linux-6.16.1。修复 #5758。
  · [bdubbs] — 更新到 iproute2-6.16.0。修复 #5773。
  · [bdubbs] — 更新到 systemd-257.8。修复 #5751。
· 2025 年 8 月 8 日
  · [bdubbs] — 更新到 Python-3.13.6 (安全更新)。修复 #5776。
  · [bdubbs] — 更新到 openssl-3.5.2。修复 #5775。
  · [bdubbs] — 更新到 libffi-3.5.2。修复 #5772。
  · [bdubbs] — 更新到 gcc-15.2.0。修复 #5777。
· 2025 年 8 月 5 日
  · [bdubbs] — 为 Python 修复 CVE-2025-8194。修复 #5774。
· 2025 年 8 月 1 日

- [bdubbs] — 更新到 meson-1.8.3。修复 #5771。
- [bdubbs] — 更新到 gdbm-1.26。修复 #5770。
- [bdubbs] — 更新到 binutils-2.45。修复 #5766。
- [bdubbs] — 更新到 gettext-0.26。修复 #5762。
- [bdubbs] — 更新到 glibc-2.42。修复 #5765。
- [bdubbs] — 更新到 man-pages-6.15。修复 #5763。

· 2025 年 7 月 15 日
- [bdubbs] — 更新到 vim-9.1.1552 (安全更新)。修复 #5760。
- [bdubbs] — 更新到 readline-8.3。修复 #5755。
- [bdubbs] — 更新到 openssl-3.5.1。修复 #5723。
- [bdubbs] — 更新到 ninja-1.13.1。修复 #5759。
- [bdubbs] — 更新到 linux-6.15.6。修复 #5757。
- [bdubbs] — 更新到 gettext-0.25.1。修复 #5753。
- [bdubbs] — 更新到 e2fsprogs-1.47.3。修复 #5758。
- [bdubbs] — 更新到 bash-5.3。修复 #5754。

· 2025 年 7 月 1 日
- [bdubbs] — 更新到 iana-etc-20250618。处理 #5006。
- [bdubbs] — 更新到 vim-9.1.1497。处理 #4500。
- [bdubbs] — 更新到 util-linux-2.41.1。修复 #5749。
- [bdubbs] — 更新到 pkgconf-2.5.1。修复 #5746。
- [bdubbs] — 更新到 ninja-1.13.0。修复 #5745。
- [bdubbs] — 更新到 linux-6.15.4。修复 #5748。
- [bdubbs] — 更新到 less-679。修复 #5747。
- [bdubbs] — 更新到 automake-1.18.1。修复 #5752。

· 2025 年 6 月 15 日
- [bdubbs] — 更新到 meson-1.8.2。修复 #5742。
- [bdubbs] — 更新到 linux-6.15.2。修复 #5725。
- [bdubbs] — 更新到 libffi-3.5.1。修复 #5741。
- [bdubbs] — 更新到 iproute2-6.15.0。修复 #5732。
- [bdubbs] — 更新到 Python-3.13.5。修复 #5743。

· 2025 年 6 月 4 日
- [bdubbs] — 更新到 ncurses-6.5-20250531。修复 #5737。
- [bdubbs] — 更新到 readline-8.3-rc2。修复 #5738。
- [bdubbs] — 更新到 bash-5.3-rc2。修复 #5738。
- [bdubbs] — 更新到 Python-3.13.4。修复 #6739。

· 2025 年 6 月 1 日
- [bdubbs] — 更新到 iana-etc-20250519。处理 #5006。
- [bdubbs] — 更新到 vim-9.1.1418。处理 #4500。

- [bdubbs] — 更新到 kbd-2.8.0。修复 #5736。
- [bdubbs] — 更新到 systemd-257.6。修复 #5674。
- [bdubbs] — 更新到 setuptools-80.9.0。修复 #5728。
- [bdubbs] — 更新到 meson-1.8.1。修复 #5731。
- [bdubbs] — 更新到 automake-1.18。修复 #5734。
- [bdubbs] — 更新 bc，expect，ncurses，以及 gmp 的构建命令，以适应 gcc-15。
- [bdubbs] — 更新到 gcc-15.1.0。修复 #5707。
- [bdubbs] — 更新到 less-678。修复 #5724。
- [bdubbs] — 更新到 readline-8.3-rc1。修复 #5726。
- [bdubbs] — 更新到 bash-5.3-rc1。修复 #5714。

· 2025 年 5 月 15 日
- [bdubbs] — 更新到 setuptools-80.7.1。修复 #5715。
- [bdubbs] — 更新到 man-pages-6.14。修复 #5720。
- [bdubbs] — 更新到 man-db-2.13.1。修复 #5719。
- [bdubbs] — 更新到 m4-1.4.20。修复 #5722。
- [bdubbs] — 更新到 linux-6.14.6。修复 #5717。
- [bdubbs] — 更新到 gettext-0.25。修复 #5718。

· 2025 年 5 月 1 日
- [bdubbs] — 更新到 vim-9.1.1353。处理 #4500。
- [bdubbs] — 更新到 setuptools-80.0.1。修复 #5710。
- [bdubbs] — 更新到 packaging-25.0。修复 #5706。
- [bdubbs] — 更新到 meson-1.8.0。修复 #5713。
- [bdubbs] — 更新到 linux-6.14.4。修复 #5709。
- [bdubbs] — 更新到 iana-etc-20250407。处理 #5006。
- [bdubbs] — 更新到 gperf-3.3。修复 #5708。
- [bdubbs] — 更新到 elfutils-0.193。修复 #5711。

· 2025 年 4 月 15 日
- [bdubbs] — 更新到 libcap-2.76。修复 #5704。
- [bdubbs] — 更新到 perl-5.40.2 (安全更新)。修复 #5703。
- [bdubbs] — 加入 packaging-24.2 (Python 模块)。Wheel 需要它。
- [bdubbs] — 更新到 xz-5.8.1。修复 #5694。
- [bdubbs] — 更新到 wheel-0.46.1 (Python 模块)。修复 #5693。
- [bdubbs] — 更新到 sysklogd-2.7.2。修复 #5690。
- [bdubbs] — 更新到 Python3-3.13.3。修复 #5697。
- [bdubbs] — 更新到 openssl-3.5.0。修复 #5701。
- [bdubbs] — 更新到 meson-1.7.2。修复 #5691。
- [bdubbs] — 更新到 linux-6.14.2。修复 #5680。
- [bdubbs] — 更新到 iproute2-6.14.0。修复 #5682。

- [bdubbs] — 更新到 gzip-1.14。修复 #5699。
- [bdubbs] — 更新到 grep-3.12。修复 #5702。
- [bdubbs] — 更新到 gperf-3.2.1。修复 #5695。
- [bdubbs] — 更新到 gawk-5.3.2。修复 #5692。
- [bdubbs] — 更新到 diffutils-3.12。修复 #5696。
- [bdubbs] — 更新到 coreutils-9.7。修复 #5698。

· 2025 年 4 月 1 日
- [bdubbs] — 更新到 vim-9.1.1263。处理 #4500。
- [bdubbs] — 更新到 iana-etc-20250328。处理 #5006。
- [bdubbs] — 更新到 xz-5.8.0。修复 #5684。
- [bdubbs] — 更新到 util-linux-2.41。修复 #5648。
- [bdubbs] — 更新到 tzdata-2025b。修复 #5681。
- [bdubbs] — 更新到 setuptools-78.1.0。修复 #5676。
- [bdubbs] — 更新到 patch-2.8。修复 #5689。
- [bdubbs] — 更新到 mpfr-4.2.2。修复 #5677。
- [bdubbs] — 更新到 kmod-34.2。修复 #5688。
- [bdubbs] — 更新到 gdbm-1.25。修复 #5679。
- [bdubbs] — 更新到 flit_core-3.12.0。修复 #5683。
- [bdubbs] — 更新到 expat-2.7.1。修复 #5685。

· 2025 年 3 月 15 日
- [bdubbs] — 更新到 vim-9.1.1202。处理 #4500。
- [bdubbs] — 更新到 iana-etc-20250304。处理 #5006。
- [bdubbs] — 更新到 setuptools-76.0.0。修复 #5665。
- [bdubbs] — 更新到 pkgconf-2.4.3。修复 #5672。
- [bdubbs] — 更新到 man-pages-6.13。修复 #5673。
- [bdubbs] — 更新到 linux-6.13.7。修复 #5664。
- [bdubbs] — 更新到 libcap-2.75。修复 #5667。
- [bdubbs] — 更新到 kmod-34.1。修复 #5671。
- [bdubbs] — 更新到 jinja2-3.1.6。修复 #5670。
- [bdubbs] — 更新到 expat-2.7.0。修复 #5675。
- [bdubbs] — 更新到 dbus-1.16.2。修复 #5663。

· 2025 年 3 月 5 日
- [bdubbs] — LFS-12.3 发布。

1.4. 相关资源

1.4.1. FAQ

如果在构建 LFS 的过程中您遇到了任何问题，或是存在疑问，或者觉得书中存在拼写错误，请先参考常见问题 (FAQ) 列表，它位于 https://www.linuxfromscratch.org/faq/。

1.4.2. 邮件列表

服务器 `linuxfromscratch.org` 管理了若干用于 LFS项目开发过程的邮件列表，其中有主要的开发列表和技术支持列表，以及其他辅助列表。如果 FAQ 不能解决您的问题，您可以访问 https://www.linuxfromscratch.org/search.html 在邮件列表中进行搜索。

如果希望了解各个邮件列表的信息，如订阅方法、过往邮件存档等，访问 https://www.linuxfromscratch.org/mail.html。

1.4.3. IRC

一些 LFS 社区成员通过互联网中继聊天系统 (IRC) 提供支援。在使用这一渠道之前，首先保证您的问题并没有被 LFS FAQ 和邮件列表解决。您可以在 `irc.libera.net` 找到 IRC 网络，支持频道的名字是 #lfs-support。

1.4.4. 镜像站

LFS 项目在全世界分布着若干镜像站，您可以通过这些镜像站更容易地访问 LFS 网站，并下载需要的软件包。请访问 LFS 网站 https://www.linuxfromscratch.org/mirrors.html 获取最新的镜像站点列表。

1.4.5. 联系信息

请直接将您的问题和评论发送到某个 LFS 邮件列表 (上面已经给出)。

1.5. 如何求助

注意

如果您在按照 LFS 的指示构建某个软件包时出现了问题，我们强烈反对在未通过第 1.4 节 “相关资源”给出的 LFS 支持通道进行讨论前，直接将问题发布到上游支持通道。这样做通常十分低效，因为上游维护者很少熟悉 LFS 的构建过程。即使您真的遇到了上游导致的问题，LFS 社区仍然能够帮您提取上游维护者需要的信息，并正确地报告问题。

如果您一定要通过上游支持通道直接提问，请至少注意许多上游项目的支持通道和 bug 管理系统是分离的。对于这些项目，用于提问的 “bug” 报告会被视为无效，并可能令上游开发者感到不快。

如果您在按照本书工作的过程中遇到任何问题或者疑问，请先阅读位于 https://www.linuxfromscratch.org/faq/#generalfaq 的常见问题列表，一般来说可以找到答案。如果您的问题没有被 FAQ 解决，试着找到问题的根源。这个指南指出了一些疑难问题的排查思路：https://www.linuxfromscratch.org/hints/downloads/files/errors.txt。

如果 FAQ 中没有您的问题，访问 https://www.linuxfromscratch.org/search.html，在邮件列表中搜索。

我们也有一个不错的 LFS 社区，社区成员愿意通过邮件列表和 IRC (见第 1.4 节 “相关资源”) 提供支援。然而，我们每天都会得到一大堆明明在 FAQ 或者邮件列表中能找到答案的问题。因此，为了使得技术支持的效能最大化，您需要自己先对问题进行一些研究。这样，我们就能够集中精力解决最特殊的支援需求。如果您的研究得不到结果，请您在求助时附带下面列出的全部相关信息。

1.5.1. 需要提供的信息

除了简要描述您遇到的问题外，您应该在求助邮件中附带下列必要信息。

· LFS 手册的版本 (您正在阅读的版本是 12.4-systemd)

· 构建 LFS 时使用的宿主发行版名称和版本
· 宿主系统需求脚本的输出
· 出现问题的软件包或书内章节
· 程序输出的原始错误消息，或者对问题的清晰描述
· 您是否进行了超出本书内容的操作

注意

有超出本书内容的操作，并不意味着我们就不会协助您。无论如何，LFS 强调个人体验。在求助信中说明您对本书给出构建过程的改动，有助于我们猜测和判定问题的可能原因。

1.5.2. 配置脚本的问题

如果在运行 **configure** 脚本的过程中出现问题，请阅读日志文件 `config.log`。它可能包含**configure** 运行时没有输出到屏幕的具体问题。求助时请附带日志文件中与问题 相关 的部分。

1.5.3. 编译错误

屏幕上的输出和一些文件的内容对于确认编译错误的原因都很有用。屏幕输出来自于 **configure** 脚本和 **make** 命令。您不用附带所有输出内容，只要包含足够相关信息即可。例如，这里给出从 **make** 的屏幕输出中截取的一段:

```
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
```

对于本例来说，许多人会只附带靠下的一行:

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

这一行只告诉我们某些事情出问题了，而完全没有说明哪里出了问题。而上面的一段输出包含了出现问题的命令行和相关的错误消息，这是我们需要的。

可以在线阅读一篇关于如何在网络上提问的精彩文章：http://catb.org/~esr/faqs/smart-questions.html。请阅读它并遵循它的建议。这样可以增加您得到帮助的可能性。

第 II 部分 准备工作

第 2 章 准备宿主系统

2.1. 概述

在本章中，我们会检查那些构建 LFS 系统必须的宿主工具，如果必要的话就安装它们。之后我们会准备一个容纳 LFS 系统的分区。我们将亲自建立这个分区，在分区上建立文件系统，并挂载该文件系统。

2.2. 宿主系统需求

2.2.1. 硬件

LFS 编辑建议使用有四个以上 CPU 核心和至少 8 GB 内存的硬件进行构建。不满足上述条件的老旧系统仍能完成构建，但构建软件包所需的时间可能大大超过本书给出的估计。

2.2.2. 软件

您的宿主系统必须拥有下列软件，且版本不能低于我们给出的最低版本。对于大多数现代 Linux 发行版来说这不成问题。要注意的是，很多发行版会把软件的头文件放在单独的软件包中，这些软件包的名称往往是 `<软件包名>-devel` 或者 `<软件包名>-dev`。如果您的发行版为下列软件提供了这类软件包，一定要安装它们。

比下列最低版本更古老的版本可能正常工作，但作者没有进行测试。

- **Bash-3.2** (/bin/sh 必须是到 bash 的符号链接或硬连接)
- **Binutils-2.13.1** (比 2.45 更新的版本未经测试，不推荐使用)
- **Bison-2.7** (/usr/bin/yacc 必须是到 bison 的链接，或者是一个执行 bison 的小脚本)
- **Coreutils-8.1**
- **Diffutils-2.8.1**
- **Findutils-4.2.31**
- **Gawk-4.0.1** (/usr/bin/awk 必须是到 gawk 的链接)
- **GCC-5.4**，包括 C++ 编译器 **g++** (比 15.2.0 更新的版本未经测试，不推荐使用)。C 和 C++ 标准库 (包括头文件) 也必须可用，这样 C++ 编译器才能构建宿主环境的程序
- **Grep-2.5.1a**
- **Gzip-1.3.12**
- **Linux Kernel-5.4**

  内核版本的要求是为了符合我们在第 5 章和第 8 章中编译 glibc 时的设置，这样可以禁用为旧版内核设计的变通措施，使得编译得到的 glibc 运行稍快，且占用空间稍小。截至 2024 年 12 月，5.4 是内核开发者仍然提供支持的最老内核版本。一些比 5.4 更老的内核可能被第三方团队支持，但它们不被视为上游正式发布的内核版本；详见 https://kernel.org/category/releases.html。

  如果宿主内核比 5.4 更早，您需要将内核升级到较新的版本。升级内核有两种方法，如果您的发行版供应商提供了 5.4 或更新的内核软件包，您可以直接安装它。如果供应商没有提供一个足够新的内核包，或者您不想安装它，您可以自己编译内核。编译内核和配置启动引导器 (假设宿主使用 GRUB) 的步骤在第 10 章中。

  我们需要宿主系统支持 UNIX 98 伪终端 (PTY)。这项功能在所有使用 Linux 5.4 或更新内核的桌面或服务器发行版中应该已经启用。如果您要构建自定义的宿主内核，需要确认在内核配置中 `CONFIG_UNIX98_PTYS` 选项被设为 `y`。
- **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

上面要求的符号链接是根据本书构建 LFS 的充分条件，不是必要条件。链接指向其他软件 (如 dash 或 mawk 等) 可能不会引发问题，但 LFS 开发团队没有尝试过这种做法，也无法提供帮助。对于一些软件包来说，您可能需要修改本书中的指令或者使用额外的补丁，才能在这类宿主环境成功构建。

为了确定您的宿主系统拥有每个软件的合适版本，且能够编译程序，请运行下列命令:

```
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. 分阶段构建 LFS

LFS 被设计为在一次会话中构建完成。换句话说，本书的指令假设，在整个编译过程中，系统不会关闭或重启。当然，构建过程不需要严格地一气呵成，但是要注意如果在重新启动后继续编译 LFS，根据构建进度的不同，可能需要再次进行某些操作。

2.3.1. 第 1–4 章

这些章节在宿主系统运行命令。在重启后，注意下列事项:

· 在第 2.4 节之后，以 root 用户身份执行的步骤要求 LFS 环境变量已经为 root 用户设置好。

2.3.2. 第 5–6 章

- /mnt/lfs 分区需要重新挂载。
- 这两章的步骤必须由用户 `lfs` 完成。在进行这些步骤时，必须先执行 **su - lfs** 命令。否则，您可能会将软件包安装到宿主系统上，这可能导致宿主系统无法使用。
- 编译过程的一般说明中的过程是关键的。如果无法确定一个软件包是否正确安装，首先确认之前解压的源码包已经被删除，然后重新解压源码包的文件，重新执行该软件包对应章节的所有命令。

2.3.3. 第 7–10 章

- /mnt/lfs 分区需要重新挂载。
- 从 “准备虚拟内核文件系统” 到 “进入 Chroot 环境” 的一些操作必须以 `root` 身份完成，且 LFS 环境变量必须为 `root` 用户设定。
- 在进入 chroot 环境时，LFS 环境变量必须为 `root` 设置好。在进入 chroot 环境后就不需要 LFS 变量。
- 虚拟文件系统必须挂载好。在进入 chroot 环境之前，请切换到一个宿主系统的虚拟终端，以 `root` 身份执行第 7.3.1 节 “挂载和填充 /dev”和第 7.3.2 节 “挂载虚拟内核文件系统”中的命令。

2.4. 创建新的分区

像其他操作系统那样，LFS 一般也被安装在一个专用的分区。我们推荐您为 LFS 选择一个可用的空分区，或者在有充足未划分空间的情况下，创建一个新分区。

一个最小的系统需要大小约 10 吉字节 (GB) 的分区。这足够保存所有源代码压缩包，并且编译所有软件包。然而，如果希望用 LFS 作为日常的 Linux 系统，很可能需要安装额外软件，需要更多空间。一个 30 GB 的分区是比较合理的。LFS 系统本身用不了太多空间，但大分区可以提供足够的临时存储空间，以及在 LFS 构建完成后增添附加功能需要的空间。另外，编译软件包可能需要大量磁盘空间，但在软件包安装完成后可以回收这些空间。

计算机未必有足够满足编译过程要求的内存 (RAM) 空间，因此可以使用一个小的磁盘分区作为 `swap` 空间。内核使用此分区存储很少使用的数据，从而为活动进程留出更多内存。LFS 的 `swap` 分区可以和宿主系统共用，这样就不用专门为 LFS 创建一个。

启动一个磁盘分区程序，例如 **cfdisk** 或者**fdisk**。在启动分区程序时需要一个命令行参数，表示希望创建新分区的硬盘，例如主硬盘 `/dev/sda`。创建一个 Linux 原生分区，如果有必要的话再创建一个 `swap` 分区。请参考 cfdisk(8) 或者 fdisk(8) 来学习如何使用分区程序。

注意

有经验的用户可以尝试其他分区架构。LFS 系统可以被构建在软件 RAID 阵列或 LVM 逻辑卷上。然而，一些分区架构需要 initramfs，这是一个比较复杂的话题。对于初次构建 LFS 的用户来说，不推荐采用这些分区方法。

牢记新分区的代号 (例如 `sda5`)。本书将这个分区称为 LFS 分区。还需要记住 `swap` 分区的代号。之后在设置 `/etc/fstab` 文件时要用到这些代号。

2.4.1. 其他分区问题

经常有人在 LFS 邮件列表询问如何进行系统分区。这是一个相当主观的问题。许多发行版在默认情况下会使用整个磁盘，只留下一个小的 swap 分区。对于 LFS 来说，这往往不是最好的方案。它削弱了系统的灵活性，使得我们难以在多个发行版或 LFS 系统之间共享数据，增加系统备份时间，同时导致文件系统结构的不合理分配，浪费磁盘空间。

2.4.1.1. 根分区

一个 LFS 根分区 (不要与 `/root` 目录混淆) 一般分配 20 GB 的空间就足以保证多数系统的运行。它提供了构建 LFS 以及 BLFS 的大部分软件包的充足空间，但又不太大，因此能够创建多个分区，多次尝试构建 LFS 系统。

2.4.1.2. 交换 (Swap) 分区

许多发行版自动创建交换空间。一般来说，推荐采用两倍于物理内存的交换空间，然而这几乎没有必要。如果磁盘空间有限，可以创建不超过 2GB 的交换空间，并注意它的使用情况。

如果您希望使用 Linux 的休眠功能 (挂起到磁盘)，它会在关机前将内存内容写入到交换分区。这种情况下，交换分区的大小应该至少和系统内存相同。

交换到磁盘从来就不是一件好事。对于机械硬盘，通过听硬盘的工作噪声，同时观察系统的响应速度，就能分辨出系统是否在交换。对于 SSD，您无法听到工作噪声，但可以使用 **top** 或 **free** 命令查看使用了多少交换空间。应该尽量避免在 SSD 中建立交换分区。一旦发生交换，首先检查是否输入了不合理的命令，例如试图编辑一个 5GB 的文件。如果交换时常发生，最好的办法是为你的系统添置内存。

2.4.1.3. Grub Bios 分区

如果启动磁盘采用 GUID 分区表 (GPT)，那么必须创建一个小的，一般占据 1MB 的分区，除非它已经存在。这个分区不能格式化，在安装启动引导器时必须能够被 GRUB 发现。这个分区在使用 **fdisk** 命令时显示为 'BIOS Boot' 分区，在使用 **gdisk** 命令时分区类型代号显示为为 EF02。

注意

Grub Bios 分区必须位于 BIOS 引导系统使用的磁盘上。这个磁盘未必是存放 LFS 根分区的磁盘。不同磁盘可以使用不同分区表格式，只有引导盘采用 GPT 时才必须创建该分区。

2.4.1.4. 常用分区

还有其他几个并非必须，但在设计磁盘布局时应当考虑的分区。下面的列表并不完整，但可以作为一个参考。

- /boot – 高度推荐。这个分区可以存储内核和其他引导信息。为了减少大磁盘可能引起的问题，建议将 /boot 分区设为第一块磁盘的第一个分区。为它分配 200 MB 就绰绰有余。
- /boot/efi – EFI 系统分区，在使用 UEFI 引导系统时是必要的。阅读 BLFS 页面以获得详细信息。
- /home – 高度推荐。独立的 /home 分区可以在多个发行版或 LFS 系统之间共享 home 目录和用户设置。它的尺寸一般很大，取决于硬盘的可用空间。
- /usr – 在 LFS 中，`/bin`，`/lib`，以及 `/sbin` 是指向 `/usr` 中对应目录的符号链接。因此，`/usr` 包含系统运行需要的所有二进制程序和库。对于 LFS，通常不需要为 `/usr` 创建单独的分区。如果仍然需要这种配置，需要为其建立一个能够容纳系统中所有程序和库的分区。同时，在这种配置下，根分区可以非常小 (可能只需要一吉字节)，因此它适用于瘦客户端或者无盘工作站 (此时 `/usr` 从远程服务器挂载)。然而，需要注意的是，必须使用 initramfs (LFS 没有包含)，才能引导具有单独的 `/usr` 分区的系统。
- /opt – 这个目录往往被用于在 BLFS 中安装 KDE 或 Texlive 等大型软件，以免把大量文件塞进 /usr 目录树。如果将它划分为独立分区，5 到 10 GB 一般就足够了。
- /tmp – 默认情况下，systemd 在这里挂载一个 `tmpfs`。如果希望覆盖默认行为，在配置 LFS 系统系统时按照第 9.10.3 节 “禁止将 tmpfs 挂载到 /tmp”进行操作。
- /usr/src – 将它划分为独立分区，可以用于存储 BLFS 源代码，并在多个 LFS 系统之间共享它们。它也可以用于编译 BLFS 软件包。30-50 GB 的分区可以提供足够的空间。

如果您希望在启动时自动挂载任何独立的分区，就要在 /etc/fstab 文件中指定。指定挂载分区的详细过程将在第 10.2 节 “创建 /etc/fstab 文件”中讨论。

2.5. 在分区上建立文件系统

分区只是由分区表中记录的边界确定的一段扇区。在操作系统使用分区存储文件之前，必须格式化该分区，以在分区中建立一个文件系统。文件系统通常包含标签，目录块，数据块，用于定位文件的索引结构等。文件系统也帮助操作系统记录分区中的可用空间，在创建文件或增大已有文件的大小时保留需要的扇区，并回收利用已删除的文件的数据空间。一些文件系统还提供数据冗余和错误恢复功能。

LFS 可以使用 Linux 内核能够识别的任何文件系统，但最常用的是 ext3 和 ext4。文件系统的选型是一个复杂的问题，要综合考虑分区的大小，以及其中所存储文件的特征。例如：

ext2
: 适用于不经常更新的小分区，例如 /boot。

ext3
: 是 ext2 的升级版本，拥有日志系统，能够在非正常关机的情况下恢复分区的正常状态。它被广泛用于一般场合。

ext4
: 是 ext 文件系统家族的最新成员，它支持纳秒精度时间戳、创建或使用超大文件 (最大 16 TB) 支持等新功能，速度也更快。

其他文件系统，包括 FAT32, NTFS, JFS 和 XFS 在特定场合也很有用。关于它们和其他文件系统的更多信息可以在 https://en.wikipedia.org/wiki/Comparison_of_file_systems 找到。

LFS 假设根文件系统 (/) 采用 ext4 文件系统。输入以下命令在 LFS 分区创建一个 ext4 文件系统：

```
mkfs -v -t ext4 /dev/<xxx>
```

命令中 <xxx> 应该替换成 LFS 分区的名称。

如果您拥有一个现成的 swap 分区，就不需要格式化它。如果新创建了一个 swap 分区，需要执行以下命令以初始化它：

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

命令中 <yyy> 应该替换成 swap 分区的名称。

2.6. 设置 $LFS 环境变量和 Umask

在本书中，我们经常使用环境变量 LFS。您应该保证，在构建 LFS 的全过程中，该变量都被定义且设置为您构建 LFS 使用的目录 —— 我们使用 /mnt/lfs 作为例子，但您可以任意选择目录名。如果您在一个独立的分区上构建 LFS，那么这个目录将用作该分区的挂载点。选择一个目录，然后用以下命令设置环境变量：

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

设置该环境变量的好处是，我们可以直接输入书中的命令，例如 **mkdir -v $LFS/tools**。Shell 在解析命令时会自动将 “$LFS” 替换成 “/mnt/lfs” (或是您设置的其他值)。

现在设定创建文件时使用的访问权限模式掩码 (umask) 为 022，以防个别宿主发行版使用不同的默认值：

```
umask 022
```

将 umask 设定为 022，保证只有文件所有者可以写新创建的文件和目录，但任何人都可读取或搜索 (仅针对目录) 它们 (如果 open(2) 系统调用使用默认模式，则新文件将具有权限模式 644，而新目录具有权限模式 755)。过于宽松的默认值可能在 LFS 系统中遗留安全问题，而过于严格的默认值可能在构建或使用 LFS 时引发奇怪的问题。

小心

无论何时，如果您离开并重新进入了工作环境，一定要再次确认 LFS 已经正确设定，且 umask 被设为 022 (例如，使用 **su** 切换到 root 或者其他用户时)。以下命令可用于检查 LFS 的设置是否正确:

```
echo $LFS
```

确认上述命令输出用于构建 LFS 系统的位置，如果使用了本书的例子，就是 /mnt/lfs。

以下命令可用于检查 umask 是否设定正确:

```
umask
```

输出可能是 0022 或 022 (先导零的个数依赖于宿主发行版)。

如果上述两条命令中任意一条的输出不正确，则需要按照本页面之前给出的命令，将 $LFS 设为正确的目录名，并将 umask 设为 022。

注意

确保 LFS 变量和 umask 始终正确的一种方法是：编辑您的主目录中的 .bash_profile，以及/root/.bash_profile，为它们加入上述 **export** 和 **umask** 命令。另外，在 /etc/passwd 中，每个需要使用 LFS 变量的用户的 shell 都必须是 bash，以保证每次登录时都执行 .bash_profile 中的命令。

另外还要考虑登录宿主系统的方式，如果您使用图形显示管理器登录，再启动虚拟终端，那么 .bash_profile 一般不会被虚拟终端执行。此时，应该将上述命令加入到您使用的用户和 root 用户的 .bashrc 文件中。另外，一些发行版的 .bashrc 中使用 "if" 命令，使其在非交互 bash 的启动过程中不执行其余命令。此时必须将上述命令添加到交互性检测之前。

2.7. 挂载新的分区

我们已经在分区上建立了文件系统，为了从宿主系统访问分区，我们需要把分区挂载到选定的挂载点上。正如前一节所述，本书假设将文件系统挂载到 LFS 环境变量指定的目录中。

严格来说，我们不能“挂载一个分区”。我们挂载的是该分区中的文件系统。但是，由于一个分区最多只包含一个文件系统，人们经常不加区分地用“分区”代表分区中的文件系统。

输入以下命令以创建挂载点，并挂载 LFS 文件系统:

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

命令中 <xxx> 应该替换成 LFS 分区的名称。

如果为 LFS 创建了多个分区 (例如一个作为 /，另一个作为 /home)，那么它们都需要被挂载:

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

将 <xxx> 和 <yyy> 替换成对应的分区代号。

将 $LFS 目录 (即为 LFS 系统新创建的文件系统的根目录) 的所有者设为 root，访问权限设为 755，以防个别宿主发行版中 **mkfs** 被配置为使用与此不同的默认值:

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

请确认在挂载新分区时没有使用过于严格的安全限制 (比如 nosuid 或者 nodev 等选项)。直接执行不带任何参数的 **mount** 命令，检查挂载好的 LFS 分区被指定了哪些选项。如果 nodev 或者 nosuid 被设置了，就必须重新挂载分区。

警告

上面的命令假设您在构建 LFS 的过程中不会重启计算机。如果您关闭了系统，那么您要么在继续构建过程时重新挂载分区，要么修改宿主系统的 /etc/fstab 文件，使得系统在引导时自动挂载它们。例如，可以将这一行添加到 /etc/fstab 文件中：

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

如果您使用了多个分区，它们都需要添加到 fstab 中。

如果您使用了 swap 分区，使用 **swapon** 命令启用它：

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

将 <zzz> 替换成 swap 分区的名称。

现在我们准备新的 LFS 分区，可以下载软件包了。

第 3 章 软件包和补丁

3.1. 概述

本章包含了构建基本的 Linux 系统时需要下载的软件包列表。我们给出的版本号对应于已经确定可以正常工作的版本，本书是基于这些版本编写的。我们强烈反对使用不同的版本，这是因为我们为一个版本提供的构建命令未必适用于其他版本，除非 LFS 勘误页面或安全公告指定使用其他版本。最新版本的软件包可能有需要排查的问题，我们会在本书的开发过程中进行排查，将解决方案找到并固定下来。

在一些软件包发布新版本时，可能同时提供正式发布的源代码压缩包与 (Git 或 SVN) 仓库的版本快照，且两种压缩包的文件名可能非常相似，甚至完全相同。正式发布的源码包除了版本快照中的内容外，还包含一些并未在仓库中存储，却仍然十分关键的文件 (例如，**autoconf** 生成的 **configure** 脚本)。本书尽可能地使用正式发布的源码包。如果使用版本快照代替本书指定的源码包，可能会导致构建出现问题。

本书列出的下载位置可能失效。如果本书发布后，某个下载位置发生变化，可以用 Google (https://www.google.com/) 提供的搜索引擎找到大多数软件包。如果搜索不到，尝试 https://www.linuxfromscratch.org/lfs/mirrors.html#files 给出的备用地址。

下载好的软件包和补丁需要保存在一个适当的位置，使得在整个构建过程中都能容易地访问它们。另外，还需要一个工作目录，以便解压和编译软件包。我们可以将 `$LFS/sources` 既用于保存软件包和补丁，又作为工作目录。这样，我们需要的所有东西都在 LFS 分区中，因此在整个构建过程中都能够访问。

为了创建这个目录，在开始下载软件包之前，以`root` 身份执行:

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

下面为该目录添加写入权限和 sticky 标志。“Sticky” 标志使得即使有多个用户对该目录有写入权限，也只有文件所有者能够删除其中的文件。输入以下命令，启用写入权限和 sticky 标志:

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

可以用下列方法获取构建 LFS 必须的软件包和补丁:

- 在后续的两节中，单独下载这些文件。
- 对于本手册的稳定版，从 https://www.linuxfromscratch.org/mirrors.html#files 中列出的某个镜像站下载包含所有所需文件的压缩包。
- 使用 **wget** 和下面描述的 wget-list 下载这些文件。

如果要使用 wget-list-systemd 作为 **wget** 命令的输入，以下载所有软件包和补丁，使用命令:

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

另外，自 LFS-7.0 以来，本书提供一个单独的文件 md5sums，用来检查所有软件包的正确性。将该文件复制到 `$LFS/sources`，运行以下命令即可得到检查结果:

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

使用上面的各种方法获取文件后，都可以执行这项检查。

如果以非 `root` 用户身份下载了软件包和补丁，则下载的文件会属于该用户。文件系统使用 UID 记录文件所有者，而宿主系统中普通用户的 UID 在 LFS 中未被分配。因此，这些文件保留到最终的 LFS 系统后，会属于一个没有命名的 UID 。如果您不准备在 LFS 系统中为您的用户分配相同的 UID，现在就将这些文件的所有者改为 `root`，以避免这一问题:

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

3.2. 全部软件包

注意

在下载软件包之前，阅读安全公告以了解是否应该对某个软件包使用较新的版本，以避免安全缺陷。

上游发布源可能移除旧的发布版本，特别是在这些旧版本存在安全缺陷的情况下。如果下面列出的某个 URL 无法访问，应该首先阅读安全公告，以确认是否应该使用更新的 (已经修复安全缺陷的) 版本。如果情况并非如此，可以尝试从镜像站下载被移除的软件包。尽管即使在某个发布版本因为安全缺陷被移除时仍然可能通过镜像站下载它，最好不要在构建系统时使用已知有安全缺陷的软件包版本。

下载或者用其他方法获取下列软件包。

- **Acl (2.3.2) - 363 KB:**
主页：https://savannah.nongnu.org/projects/acl
下载地址：https://download.savannah.gnu.org/releases/acl/acl-2.3.2.tar.xz
MD5 校验和：`590765dee95907dbc3c856f7255bd669`

- **Attr (2.5.2) - 484 KB:**
主页：https://savannah.nongnu.org/projects/attr
下载地址：https://download.savannah.gnu.org/releases/attr/attr-2.5.2.tar.gz
MD5 校验和：`227043ec2f6ca03c0948df5517f9c927`

- **Autoconf (2.72) - 1,360 KB:**
主页：https://www.gnu.org/software/autoconf/
下载地址：https://ftp.gnu.org/gnu/autoconf/autoconf-2.72.tar.xz
MD5 校验和：`1be79f7106ab6767f18391c5e22be701`

- **Automake (1.18.1) - 1,614 KB:**
主页：https://www.gnu.org/software/automake/
下载地址：https://ftp.gnu.org/gnu/automake/automake-1.18.1.tar.xz
MD5 校验和：`cea31dbf1120f890cbf2a3032cfb9a68`

- **Bash (5.3) - 11,089 KB:**
主页：https://www.gnu.org/software/bash/
下载地址：https://ftp.gnu.org/gnu/bash/bash-5.3.tar.gz
MD5 校验和：`977c8c0c5ae6309191e7768e28ebc951`

- **Bc (7.0.3) - 464 KB:**
主页：https://github.com/gavinhoward
下载地址：https://github.com/gavinhoward/bc/releases/download/7.0.3/bc-7.0.3.tar.xz
MD5 校验和：`ad4db5a0eb4fdbb3f6813be4b6b3da74`

- **Binutils (2.45) - 27,216 KB:**
主页：https://www.gnu.org/software/binutils/
下载地址：https://sourceware.org/pub/binutils/releases/binutils-2.45.tar.xz
MD5 校验和：`dee5b4267e0305a99a3c9d6131f45759`

- **Bison (3.8.2) - 2,752 KB:**
主页：https://www.gnu.org/software/bison/
下载地址：https://ftp.gnu.org/gnu/bison/bison-3.8.2.tar.xz
MD5 校验和：`c28f119f405a2304ff0a7ccdcc629713`

· **Bzip2 (1.0.8) - 792 KB:**
下载地址：https://www.sourceware.org/pub/bzip2/bzip2-1.0.8.tar.gz
MD5 校验和：`67e051268d0c475ea773822f7500d0e5`

· **Coreutils (9.7) - 6,015 KB:**
主页：https://www.gnu.org/software/coreutils/
下载地址：https://ftp.gnu.org/gnu/coreutils/coreutils-9.7.tar.xz
MD5 校验和：`6b7285faf7d5eb91592bdd689270d3f1`

· **D-Bus (1.16.2) - 1,090 KB:**
主页：https://www.freedesktop.org/wiki/Software/dbus
下载地址：https://dbus.freedesktop.org/releases/dbus/dbus-1.16.2.tar.xz
MD5 校验和：`97832e6f0a260936d28536e5349c22e5`

· **DejaGNU (1.6.3) - 608 KB:**
主页：https://www.gnu.org/software/dejagnu/
下载地址：https://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.3.tar.gz
MD5 校验和：`68c5208c58236eba447d7d6d1326b821`

· **Diffutils (3.12) - 1,894 KB:**
主页：https://www.gnu.org/software/diffutils/
下载地址：https://ftp.gnu.org/gnu/diffutils/diffutils-3.12.tar.xz
MD5 校验和：`d1b18b20868fb561f77861cd90b05de4`

· **E2fsprogs (1.47.3) - 9,851 KB:**
主页：https://e2fsprogs.sourceforge.net/
下载地址：https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.47.3/e2fsprogs-1.47.3.tar.gz
MD5 校验和：`113d7a7ee0710d2a670a44692a35fd2e`

· **Elfutils (0.193) - 11,695 KB:**
主页：https://sourceware.org/elfutils/
下载地址：https://sourceware.org/ftp/elfutils/0.193/elfutils-0.193.tar.bz2
MD5 校验和：`ceefa052ded950a4c523688799193a44`

· **Expat (2.7.1) - 485 KB:**
主页：https://libexpat.github.io/
下载地址：https://github.com/libexpat/libexpat/releases/download/R_2_7_1/expat-2.7.1.tar.xz
MD5 校验和：`9f0c266ff4b9720beae0c6bd53ae4469`

· **Expect (5.45.4) - 618 KB:**
主页：https://core.tcl.tk/expect/
下载地址：https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz
MD5 校验和：`00fce8de158422f5ccd2666512329bd2`

· **File (5.46) - 1,283 KB:**
主页：https://www.darwinsys.com/file/
下载地址：https://astron.com/pub/file/file-5.46.tar.gz
MD5 校验和：`459da2d4b534801e2e2861611d823864`

· **Findutils (4.10.0) - 2,189 KB:**
主页：https://www.gnu.org/software/findutils/
下载地址：https://ftp.gnu.org/gnu/findutils/findutils-4.10.0.tar.xz
MD5 校验和：`870cfd71c07d37ebe56f9f4aaf4ad872`

· **Flex (2.6.4) - 1,386 KB:**
主页：https://github.com/westes/flex
下载地址：https://github.com/westes/flex/releases/download/v2.6.4/flex-2.6.4.tar.gz
MD5 校验和：`2882e3179748cc9f9c23ec593d6adc8d`

· **Flit-core (3.12.0) - 53 KB:**
主页：https://pypi.org/project/flit-core/
下载地址：https://pypi.org/packages/source/f/flit-core/flit_core-3.12.0.tar.gz
MD5 校验和：`c538415c1f27bd69cbbbf3cdd5135d39`

· **Gawk (5.3.2) - 3,662 KB:**
主页：https://www.gnu.org/software/gawk/
下载地址：https://ftp.gnu.org/gnu/gawk/gawk-5.3.2.tar.xz
MD5 校验和：`b7014650c5f45e5d4837c31209dc0037`

· **GCC (15.2.0) - 98,688 KB:**
主页：https://gcc.gnu.org/
下载地址：https://ftp.gnu.org/gnu/gcc/gcc-15.2.0/gcc-15.2.0.tar.xz
MD5 校验和：`b861b092bf1af683c46a8aa2e689a6fd`

· **GDBM (1.26) - 1,198 KB:**
主页：https://www.gnu.org/software/gdbm/
下载地址：https://ftp.gnu.org/gnu/gdbm/gdbm-1.26.tar.gz
MD5 校验和：`aaa600665bc89e2febb3c7bd90679115`

· **Gettext (0.26) - 9,926 KB:**
主页：https://www.gnu.org/software/gettext/
下载地址：https://ftp.gnu.org/gnu/gettext/gettext-0.26.tar.xz
MD5 校验和：`8e14e926f088e292f5f2bce95b81d10e`

· **Glibc (2.42) - 19,464 KB:**
主页：https://www.gnu.org/software/libc/
下载地址：https://ftp.gnu.org/gnu/glibc/glibc-2.42.tar.xz
MD5 校验和：`23c6f5a27932b435cae94e087cb8b1f5`

注意
Glibc 开发者维护了一个 Git 分支，包含那些被认为值得包含在 Glibc-2.42 中，但不幸地在 Glibc-2.42 发布后才完成开发的补丁。LFS 编辑在有安全修补被加入该分支时会发布一条安全公告，但不会为其他新加入该分支的补丁做出任何反应。您可以自行审查这些补丁，并在构建时合入您认为重要的补丁。

· **GMP (6.3.0) - 2,046 KB:**
主页：https://www.gnu.org/software/gmp/
下载地址：https://ftp.gnu.org/gnu/gmp/gmp-6.3.0.tar.xz
MD5 校验和：`956dc04e864001a9c22429f761f2c283`

· **Gperf (3.3) - 1,789 KB:**
主页：https://www.gnu.org/software/gperf/
下载地址：https://ftp.gnu.org/gnu/gperf/gperf-3.3.tar.gz
MD5 校验和：`31753b021ea78a21f154bf9eecb8b079`

· **Grep (3.12) - 1,874 KB:**
主页：https://www.gnu.org/software/grep/
下载地址：https://ftp.gnu.org/gnu/grep/grep-3.12.tar.xz
MD5 校验和：`5d9301ed9d209c4a88c8d3a6fd08b9ac`

· **Groff (1.23.0) - 7,259 KB:**
主页：https://www.gnu.org/software/groff/
下载地址：https://ftp.gnu.org/gnu/groff/groff-1.23.0.tar.gz
MD5 校验和：`5e4f40315a22bb8a158748e7d5094c7d`

· **GRUB (2.12) - 6,524 KB:**
主页：https://www.gnu.org/software/grub/
下载地址：https://ftp.gnu.org/gnu/grub/grub-2.12.tar.xz
MD5 校验和：`60c564b1bdc39d8e43b3aab4bc0fb140`

· **Gzip (1.14) - 865 KB:**
主页：https://www.gnu.org/software/gzip/
下载地址：https://ftp.gnu.org/gnu/gzip/gzip-1.14.tar.xz
MD5 校验和：`4bf5a10f287501ee8e8ebe00ef62b2c2`

· **Iana-Etc (20250807) - 592 KB:**
主页：https://www.iana.org/protocols
下载地址：https://github.com/Mic92/iana-etc/releases/download/20250807/iana-etc-20250807.tar.gz
MD5 校验和：`de0a909103d4ff59d1424c5ec7ac9e4a`

· **Inetutils (2.6) - 1,724 KB:**
主页：https://www.gnu.org/software/inetutils/
下载地址：https://ftp.gnu.org/gnu/inetutils/inetutils-2.6.tar.xz
MD5 校验和：`401d7d07682a193960bcdecafd03de94`

· **Intltool (0.51.0) - 159 KB:**
主页：https://freedesktop.org/wiki/Software/intltool
下载地址：https://launchpad.net/intltool/trunk/0.51.0/+download/intltool-0.51.0.tar.gz
MD5 校验和：`12e517cac2b57a0121cda351570f1e63`

· **IPRoute2 (6.16.0) - 910 KB:**
主页：https://www.kernel.org/pub/linux/utils/net/iproute2/
下载地址：https://www.kernel.org/pub/linux/utils/net/iproute2/iproute2-6.16.0.tar.xz
MD5 校验和：`80e1f91bf59d572acc15d5c6eb4f3e7c`

· **Jinja2 (3.1.6) - 240 KB:**
主页：https://jinja.palletsprojects.com/en/3.1.x/
下载地址：https://pypi.org/packages/source/J/Jinja2/jinja2-3.1.6.tar.gz
MD5 校验和：`66d4c25ff43d1deaf9637ccda523dec8`

· **Kbd (2.8.0) - 1,448 KB:**
主页：https://kbd-project.org/
下载地址：https://www.kernel.org/pub/linux/utils/kbd/kbd-2.8.0.tar.xz
MD5 校验和：`24b5d24f7483726b88f214dc6c77aa41`

· **Kmod (34.2) - 434 KB:**
主页：https://github.com/kmod-project/kmod
下载地址：https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-34.2.tar.xz
MD5 校验和：`36f2cc483745e81ede3406fa55e1065a`

· **Less (679) - 857 KB:**
主页：https://www.greenwoodsoftware.com/less/
下载地址：https://www.greenwoodsoftware.com/less/less-679.tar.gz
MD5 校验和：`0386dc14f6a081a94dfb4c2413864eed`

· **Libcap (2.76) - 195 KB:**
主页：https://sites.google.com/site/fullycapable/
下载地址：https://www.kernel.org/pub/linux/libs/security/linux-privs/libcap2/libcap-2.76.tar.xz
MD5 校验和：`449ade7d620b5c4eeb15a632fbaa4f74`

· **Libffi (3.5.2) - 1,390 KB:**
主页：https://sourceware.org/libffi/
下载地址：https://github.com/libffi/libffi/releases/download/v3.5.2/libffi-3.5.2.tar.gz
MD5 校验和：`92af9efad4ba398995abf44835c5d9e9`

· **Libpipeline (1.5.8) - 1046 KB:**
主页：https://libpipeline.nongnu.org/
下载地址：https://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.8.tar.gz
MD5 校验和：`17ac6969b2015386bcb5d278a08a40b5`

· **Libtool (2.5.4) - 1,033 KB:**
主页：https://www.gnu.org/software/libtool/
下载地址：https://ftp.gnu.org/gnu/libtool/libtool-2.5.4.tar.xz
MD5 校验和：`22e0a29df8af5fdde276ea3a7d351d30`

· **Libxcrypt (4.4.38) - 612 KB:**
主页：https://github.com/besser82/libxcrypt/
下载地址：https://github.com/besser82/libxcrypt/releases/download/v4.4.38/libxcrypt-4.4.38.tar.xz
MD5 校验和：`1796a5d20098e9dd9e3f576803c83000`

· **Linux (6.16.1) - 149,042 KB:**
主页：https://www.kernel.org/
下载地址：https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.16.1.tar.xz
MD5 校验和：`32d45755e4b39d06e9be58f6817445ee`

注意
Linux 内核的更新十分频繁，多数情况下是为了解决新发现的安全问题。除非勘误页明确说明，应该使用内核的最新稳定版本。
对于那些上网很慢或者流量很贵的用户来说，可以分别下载内核的基线版本和补丁。这可以节省内核修订号更新时的下载时间和网费。

· **Lz4 (1.10.0) - 379 KB:**
主页：https://lz4.org/
下载地址：https://github.com/lz4/lz4/releases/download/v1.10.0/lz4-1.10.0.tar.gz
MD5 校验和：`dead9f5f1966d9ae56e1e32761e4e675`

· **M4 (1.4.20) - 1,997 KB:**
主页：https://www.gnu.org/software/m4/
下载地址：https://ftp.gnu.org/gnu/m4/m4-1.4.20.tar.xz
MD5 校验和：`6eb2ebed5b24e74b6e890919331d2132`

· **Make (4.4.1) - 2,300 KB:**
主页：https://www.gnu.org/software/make/
下载地址：https://ftp.gnu.org/gnu/make/make-4.4.1.tar.gz
MD5 校验和：`c8469a3713cbbe04d955d4ae4be23eeb`

· **Man-DB (2.13.1) - 2,061 KB:**
主页：https://www.nongnu.org/man-db/
下载地址：https://download.savannah.gnu.org/releases/man-db/man-db-2.13.1.tar.xz
MD5 校验和：`b6335533cbeac3b24cd7be31fdee8c83`

· **Man-pages (6.15) - 1,817 KB:**
主页：https://www.kernel.org/doc/man-pages/
下载地址：https://www.kernel.org/pub/linux/docs/man-pages/man-pages-6.15.tar.xz
MD5 校验和：`16f68d70139dd2bbcae4102be4705753`

· **MarkupSafe (3.0.2) - 21 KB:**
主页：https://palletsprojects.com/p/markupsafe/
下载地址：https://pypi.org/packages/source/M/MarkupSafe/markupsafe-3.0.2.tar.gz
MD5 校验和：`cb0071711b573b155cc8f86e1de72167`

· **Meson (1.8.3) - 2,282:**
主页：https://mesonbuild.com
下载地址：https://github.com/mesonbuild/meson/releases/download/1.8.3/meson-1.8.3.tar.gz
MD5 校验和：`08221d2f515e759686f666ff6409a903`

· **MPC (1.3.1) - 756 KB:**
主页：https://www.multiprecision.org/
下载地址：https://ftp.gnu.org/gnu/mpc/mpc-1.3.1.tar.gz
MD5 校验和：`5c9bc658c9fd0f940e8e3e0f09530c62`

· **MPFR (4.2.2) - 1,471 KB:**
主页：https://www.mpfr.org/
下载地址：https://ftp.gnu.org/gnu/mpfr/mpfr-4.2.2.tar.xz
MD5 校验和：`7c32c39b8b6e3ae85f25156228156061`

· **Ncurses (6.5-20250809) - 3,703 KB:**
主页：https://www.gnu.org/software/ncurses/
下载地址：https://invisible-mirror.net/archives/ncurses/current/ncurses-6.5-20250809.tgz
MD5 校验和：`679987405412f970561cc85e1e6428a2`

· **Ninja (1.13.1) - 286 KB:**
主页：https://ninja-build.org/
下载地址：https://github.com/ninja-build/ninja/archive/v1.13.1/ninja-1.13.1.tar.gz
MD5 校验和：`c35f8f55f4cf60f1a916068d8f45a0f8`

· **OpenSSL (3.5.2) - 51,934 KB:**
主页：https://www.openssl-library.org/
下载地址：https://github.com/openssl/openssl/releases/download/openssl-3.5.2/openssl-3.5.2.tar.gz
MD5 校验和：`890fc59f86fc21b5e4d1c031a698dbde`

· **Packaging (25.0) - 162 KB:**
主页：https://pypi.org/project/packaging/
下载地址：https://files.pythonhosted.org/packages/source/p/packaging/packaging-25.0.tar.gz
MD5 校验和：`ab0ef21ddebe09d1803575120d3f99f8`

· **Patch (2.8) - 886 KB:**
主页：https://savannah.gnu.org/projects/patch/
下载地址：https://ftp.gnu.org/gnu/patch/patch-2.8.tar.xz
MD5 校验和：`149327a021d41c8f88d034eab41c039f`

· **Perl (5.42.0) - 14,084 KB:**
主页：https://www.perl.org/
下载地址：https://www.cpan.org/src/5.0/perl-5.42.0.tar.xz
MD5 校验和：`7a6950a9f12d01eb96a9d2ed2f4e0072`

· **Pkgconf (2.5.1) - 321 KB:**
主页：https://github.com/pkgconf/pkgconf
下载地址：https://distfiles.ariadne.space/pkgconf/pkgconf-2.5.1.tar.xz
MD5 校验和：`3291128c917fdb8fccd8c9e7784b643b`

· **Procps (4.0.5) - 1,483 KB:**
主页：https://gitlab.com/procps-ng/procps/
下载地址：https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-4.0.5.tar.xz
MD5 校验和：`90803e64f51f192f3325d25c3335d057`

· **Psmisc (23.7) - 423 KB:**
主页：https://gitlab.com/psmisc/psmisc
下载地址：https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.7.tar.xz
MD5 校验和：`53eae841735189a896d614cba440eb10`

· **Python (3.13.7) - 22,236 KB:**
主页：https://www.python.org/
下载地址：https://www.python.org/ftp/python/3.13.7/Python-3.13.7.tar.xz
MD5 校验和：`256cdb3bbf45cdce7499e52ba6c36ea3`

· **Python 文档 (3.13.7) - 10,183 KB:**
下载地址：https://www.python.org/ftp/python/doc/3.13.7/python-3.13.7-docs-html.tar.bz2
MD5 校验和：`b84c0d81b2758398bb7f5b7411d3d908`

· **Readline (8.3) - 3,340 KB:**
主页：https://tiswww.case.edu/php/chet/readline/rltop.html
下载地址：https://ftp.gnu.org/gnu/readline/readline-8.3.tar.gz
MD5 校验和：`25a73bfb2a3ad7146c5e9d4408d9f6cd`

· **Sed (4.9) - 1,365 KB:**
主页：https://www.gnu.org/software/sed/
下载地址：https://ftp.gnu.org/gnu/sed/sed-4.9.tar.xz
MD5 校验和：`6aac9b2dbafcd5b7a67a8a9bcb8036c3`

· **Setuptools (80.9.0) - 1,290 KB:**
主页：https://pypi.org/project/setuptools/
下载地址：https://pypi.org/packages/source/s/setuptools/setuptools-80.9.0.tar.gz
MD5 校验和：`82e1d67883b713f9493659b50d13b436`

· **Shadow (4.18.0) - 2,293 KB:**
主页：https://github.com/shadow-maint/shadow/
下载地址：https://github.com/shadow-maint/shadow/releases/download/4.18.0/shadow-4.18.0.tar.xz
MD5 校验和：`30ef46f54363db1d624587be68794ef2`

· **Systemd (257.8) - 16,002 KB:**
主页：https://www.freedesktop.org/wiki/Software/systemd/
下载地址：https://github.com/systemd/systemd/archive/v257.8/systemd-257.8.tar.gz
MD5 校验和：`25fe5d328e22641254761f1baa74cee0`

· **Systemd Man 页面 (257.8) - 736 KB:**
主页：https://www.freedesktop.org/wiki/Software/systemd/
下载地址：https://anduin.linuxfromscratch.org/LFS/systemd-man-pages-257.8.tar.xz
MD5 校验和：a44063e2ec0cf4adfd2ed5c9e9e095c5

注意
Linux From Scratch 团队自行从 systemd 源码生成了其手册页的压缩包，以避免不必要的依赖项。

· **Tar (1.35) - 2,263 KB:**
主页：https://www.gnu.org/software/tar/
下载地址：https://ftp.gnu.org/gnu/tar/tar-1.35.tar.xz
MD5 校验和：a2d8042658cfd8ea939e6d911eaf4152

· **Tcl (8.6.16) - 11,406 KB:**
主页：https://tcl.sourceforge.net/
下载地址：https://downloads.sourceforge.net/tcl/tcl8.6.16-src.tar.gz
MD5 校验和：eaef5d0a27239fb840f04af8ec608242

· **Tcl 文档 (8.6.16) - 1,169 KB:**
下载地址：https://downloads.sourceforge.net/tcl/tcl8.6.16-html.tar.gz
MD5 校验和：750c221bcb6f8737a6791c1fbe98b684

· **Texinfo (7.2) - 6,259 KB:**
主页：https://www.gnu.org/software/texinfo/
下载地址：https://ftp.gnu.org/gnu/texinfo/texinfo-7.2.tar.xz
MD5 校验和：11939a7624572814912a18e76c8d8972

· **Time Zone Data (2025b) - 454 KB:**
主页：https://www.iana.org/time-zones
下载地址：https://www.iana.org/time-zones/repository/releases/tzdata2025b.tar.gz
MD5 校验和：ad65154c48c74a9b311fe84778c5434f

· **Util-linux (2.41.1) - 9,382 KB:**
主页：https://git.kernel.org/pub/scm/utils/util-linux/util-linux.git/
下载地址：https://www.kernel.org/pub/linux/utils/util-linux/v2.41/util-linux-2.41.1.tar.xz
MD5 校验和：7e5e68845e2f347cf96f5448165f1764

· **Vim (9.1.1629) - 18,317 KB:**
主页：https://www.vim.org
下载地址：https://github.com/vim/vim/archive/v9.1.1629/vim-9.1.1629.tar.gz
MD5 校验和：4f856c3233c1c4570bc17572e4f9e8e4

注意
Vim 的版本每天都会升级。如果需要最新版本，访问 https://github.com/vim/vim/tags。

· **Wheel (0.46.1) - 54 KB:**
主页：https://pypi.org/project/wheel/
下载地址：https://pypi.org/packages/source/w/wheel/wheel-0.46.1.tar.gz
MD5 校验和：65e09ee84af36821e3b1e9564aa91bd5

· **XML::Parser (2.47) - 276 KB:**
主页：https://github.com/chorny/XML-Parser
下载地址：https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.47.tar.gz
MD5 校验和：89a8e82cfd2ad948b349c0a69c494463

· **Xz Utils (5.8.1) - 1,428 KB:**
主页：https://tukaani.org/xz
下载地址：https://github.com//tukaani-project/xz/releases/download/v5.8.1/xz-5.8.1.tar.xz
MD5 校验和：`cf5e1feb023d22c6bdaa30e84ef3abe3`

· **Zlib (1.3.1) - 1,478 KB:**
主页：https://zlib.net/
下载地址：https://zlib.net/fossils/zlib-1.3.1.tar.gz
MD5 校验和：`9855b6d802d7fe5b7bd5b196a2271655`

· **Zstd (1.5.7) - 2,378 KB:**
主页：https://facebook.github.io/zstd/
下载地址：https://github.com/facebook/zstd/releases/download/v1.5.7/zstd-1.5.7.tar.gz
MD5 校验和：`780fc1896922b1bc52a4e90980cdda48`

以上软件包的总大小：约 NaN MB

3.3. 必要的补丁

除了软件包外，我们还需要一些补丁。有些补丁解决了本应由维护者修复的问题，有些则对软件包进行微小的修改，使得它们更容易使用。构建 LFS 系统需要下列补丁：

· **Bzip2 文档补丁 - 1.6 KB:**
下载地址：https://www.linuxfromscratch.org/patches/lfs/12.4/bzip2-1.0.8-install_docs-1.patch
MD5 校验和：`6a5ac7e89b791aae556de0f745916f7f`

· **Coreutils 上游修复补丁 - 4.1 KB:**
下载地址：https://www.linuxfromscratch.org/patches/lfs/12.4/coreutils-9.7-upstream_fix-1.patch
MD5 校验和：`96382a5aa85d6651a74f94ffb61785d9`

· **Coreutils 国际化修复补丁 - 159 KB:**
下载地址：https://www.linuxfromscratch.org/patches/lfs/12.4/coreutils-9.7-i18n-1.patch
MD5 校验和：`33ebfad32b2dfb8417c3335c08671206`

· **Expect GCC15 补丁 - 12 KB:**
下载地址：https://www.linuxfromscratch.org/patches/lfs/12.4/expect-5.45.4-gcc15-1.patch
MD5 校验和：`0ca4d6bb8d572fbcdb13cb36cd34833e`

· **Glibc FHS 补丁 - 2.8 KB:**
下载地址：https://www.linuxfromscratch.org/patches/lfs/12.4/glibc-2.42-fhs-1.patch
MD5 校验和：`9a5997c3452909b1769918c759eff8a2`

· **Kbd 退格/删除修复补丁 - 12 KB:**
下载地址：https://www.linuxfromscratch.org/patches/lfs/12.4/kbd-2.8.0-backspace-1.patch
MD5 校验和：`f75cca16a38da6caa7d52151f7136895`

以上补丁的总大小：约 191.5 KB

除了上述必要的补丁外，LFS 社区还创建了一些可选补丁。它们有的解决了一些微小的问题，有的启用了一些默认没有启用的功能。您可以浏览 https://www.linuxfromscratch.org/patches/downloads/ 查询 LFS 补丁库，并获取各种适合您需求的补丁。

第 4 章 最后准备工作

4.1. 概述

在本章中，我们将为构建临时系统进行一些额外的准备工作。我们将在 `$LFS` 中创建一些目录 (用于安装临时工具)，增加一个非特权用户并为该用户建立合适的构建环境。我们还将解释 LFS 软件包构建时间长度的测量单位 (“SBU”) 的概念，并给出关于软件包测试套件的一些信息。

4.2. 在 LFS 文件系统中创建有限目录布局

在本节中，我们开始将组成最终的 Linux 系统的内容填充到 LFS 文件系统中。首先，在 LFS 分区中创建一个有限的目录树，使得在第 6 章中编译的程序 (以及第 5 章中的 glibc 和 libstdc++) 可以被安装到它们的最终位置。这样，在第 8 章中重新构建它们时，就能直接覆盖这些临时程序。

以 `root` 身份，执行以下命令创建所需的目录布局：

```
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
```

在第 6 章中，会使用交叉编译器编译程序 (更多细节可以在工具链技术说明一节找到)。这个交叉编译器会被安装到一个专用的目录中，从而将其和其他程序分离。仍然以 `root` 用户身份，执行以下命令创建该目录：

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

注意

LFS 编辑团队特意决定不使用 `/usr/lib64` 目录。本书中的一些步骤保证工具链不使用该目录。如果该目录被创建，无论原因如何 (可能是您在使用书中的命令时出现了偏差，或您在完成 LFS 构建后安装了一个创建该目录的二进制包)，则它可能破坏您的系统。您应该经常检查并确认该目录不存在。

4.3. 添加 LFS 用户

在作为 `root` 用户登录时，一个微小的错误就可能损坏甚至摧毁整个系统。因此，我们建议在后续两章中，以非特权用户身份编译软件包。您或许可以使用自己的系统用户，但为了更容易地建立一个干净的工作环境，我们将创建一个名为 `lfs` 的新用户，以及它从属于的一个新组 (组名也是 `lfs`)，并在安装过程中以 `lfs` 身份执行命令。为了创建新用户，以 `root` 身份执行以下命令：

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

命令行中各选项的含义：

`-s /bin/bash`
设置 **bash** 为用户 `lfs` 的默认 shell。

`-g lfs`
添加用户 `lfs` 到组 `lfs`。

`-m`
为用户 `lfs` 创建一个主目录。

`-k /dev/null`
将模板目录设置为空设备文件，防止从默认模板目录 (`/etc/skel`) 复制文件到新的主目录。

`lfs`
这是新用户的名称。

如果希望以 `lfs` 身份登录，或从一个非 `root` 用户切换到 `lfs` 用户 (这与从 `root` 切换到 `lfs` 不同，后者不需要 `lfs` 用户设有密码)，则需要为 `lfs` 用户设置密码。以 `root` 用户身份，执行以下命令设置密码：

```
passwd lfs
```

将 `lfs` 设为 `$LFS` 中所有目录的所有者，使 `lfs` 对它们拥有完全访问权：

```
chown -v lfs $LFS/{usr{,/*},var,etc,tools}
case $(uname -m) in
  x86_64) chown -v lfs $LFS/lib64 ;;
esac
```

注意

在某些宿主系统上，下面的 **su** 命令不会正确完成，而会将 `lfs` 用户的登录会话挂起到后台。如果提示符 “lfs:~$” 没有很快出现，输入 **fg** 命令以修复这个问题。

下面启动一个以 `lfs` 身份运行的 shell。这可以通过在虚拟控制台登录 `lfs` 用户完成，也可以使用下面的命令切换用户：

```
su - lfs
```

参数 “-” 使得 **su** 启动一个登录 shell，而不是非登录 shell。bash(1) 和 **info bash** 详细介绍了它们的区别。

4.4. 配置环境

为了配置一个良好的工作环境，我们为 **bash** 创建两个新的启动脚本。以 `lfs` 的身份，执行以下命令，创建一个新的 `.bash_profile`：

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

在以 `lfs` 用户登录或从其他用户使用带 “-” 选项的 **su** 命令切换到 `lfs` 用户时，初始的 shell 是一个登录 shell。它读取宿主系统的 `/etc/profile` 文件 (可能包含一些设置和环境变量)，然后读取 `.bash_profile`。我们在 `.bash_profile` 中使用 **exec env -i.../bin/bash** 命令，新建一个除了 `HOME`, `TERM` 以及 `PS1` 外没有任何环境变量的 shell 并替换当前 shell。这可以防止宿主环境中不需要和有潜在风险的环境变量进入构建环境。

新的 shell 实例是 非登录 shell，它不会读取和执行 `/etc/profile` 或者 `.bash_profile` 的内容，而是读取并执行 `.bashrc` 文件。现在我们创建一个 `.bashrc` 文件：

```
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 中设定的含义：

`set +h`

set +h 命令关闭 **bash** 的散列功能。一般情况下，散列是很有用的 —— **bash** 使用一个散列表维护各个可执行文件的完整路径，这样就不用每次都在 PATH 指定的目录中搜索可执行文件。然而，在构建 LFS 时，我们希望总是使用最新安装的工具。关闭散列功能强制 shell 在运行程序时总是搜索 PATH。这样，一旦$LFS/tools/bin 中有新的工具可用，shell 就能够找到它们，而不是使用之前记忆在散列表中，由宿主发行版提供的 /usr/bin 或 /bin 中的工具。

`umask 022`

如同第 2.6 节 “设置 $LFS 环境变量和 Umask”解释的那样设置 umask。

`LFS=/mnt/lfs`

LFS 环境变量必须被设定为之前选择的挂载点。

`LC_ALL=POSIX`

LC_ALL 环境变量控制某些程序的本地化行为，使得它们以特定国家的语言和惯例输出消息。将 LC_ALL 设置为 “POSIX” 或者 “C”(这两种设置是等价的) 可以保证在交叉编译环境中所有命令的行为完全符合预期，而与宿主的本地化设置无关。

`LFS_TGT=$(uname -m)-lfs-linux-gnu`

LFS_TGT变量设定了一个非默认，但与宿主系统兼容的机器描述符。该描述符被用于构建交叉编译器和交叉编译临时工具链。工具链技术说明将提供关于这个描述符的更多信息。

`PATH=/usr/bin`

许多现代 Linux 发行版合并了 /bin 和 /usr/bin。在这种情况下，标准 PATH 变量应该被设定为 /usr/bin，以满足第 6 章所需。否则，后续命令将会增加 /bin 到搜索路径中。

`if [ ! -L /bin ]; then PATH=/bin:$PATH; fi`

如果 /bin 不是符号链接，则它需要被添加到 PATH 变量中。

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

我们将 $LFS/tools/bin 附加在默认的 PATH 环境变量之前，这样在第 5 章中，我们一旦安装了新的程序，shell 就能立刻使用它们。这与关闭散列功能相结合，降低了在第 5 章环境中新程序可用时错误地使用宿主系统中旧程序的风险。

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

在第 5 章和第 6 章中，如果没有设定这个变量，**configure** 脚本可能会从宿主系统的 /usr/share/config.site 加载一些发行版特有的配置信息。覆盖这一默认路径，避免宿主系统可能造成的污染。

`export ...`

上述命令设定了一些变量，为了让所有子 shell 都能使用这些变量，需要导出它们。

重要

一些商业发行版未做文档说明地将 /etc/bash.bashrc 引入 **bash** 初始化过程。该文件可能修改 lfs 用户的环境，并影响 LFS 关键软件包的构建。为了保证 lfs 用户环境的纯净，检查 /etc/bash.bashrc 是否存在，如果它存在就将它移走。以 root 用户身份，运行：

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

当不再需要 lfs 用户时 (第 7 章开始后)，您 (如果希望的话) 可以复原 /etc/bash.bashrc 文件。

注意我们将会在第 8.36 节 “Bash-5.3”中构建的 LFS Bash 软件包未被配置为读取或执行 /etc/bash.bashrc，因此它在完整的 LFS 系统中没有作用。

对于许多拥有多个处理器（或 CPU 核心）的系统，可以使用命令行选项或环境变量指定 make 程序可用的处理器核心数，以进行 "并行 make"，从而减少构建软件包所需的时间。例如，一块 Intel Core i9-13900K 处理器有 8 个 P (性能) 核与 16 个 E (能效) 核，且每个 P 核能同时运行两个线程，因此 Linux 内核将每个 P 核抽象为两个逻辑核心。因此，该处理器共有 32 个逻辑核心。一种显而易见的利用这些逻辑核心的方法是允许 **make** 生成至多 32 个构建任务。为此，可以将 `-j32` 选项传递给 **make**：

```
make -j32
```

或者，设置环境变量 `MAKEFLAGS`，它的内容会被 **make** 自动视为命令行选项：

```
export MAKEFLAGS=-j32
```

重要

绝对不要将一个没有数字的 `-j` 选项传递给 **make** 或在 `MAKEFLAGS` 中设定这样的选项。这样做会导致 **make** 生成无限多的构建任务并导致系统稳定性问题。

为了在构建第 5 章和第 6 章中的软件包时使用所有可用的逻辑 CPU 核心，现在将 `MAKEFLAGS` 的设置写入 `.bashrc` 中：

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

如果不希望使用全部逻辑 CPU 核心，将 `$(nproc)` 替换为希望使用的核心数。

最后，为了保证构建临时工具所需的环境准备就绪，强制 **bash** shell 读取刚才创建的配置文件：

```
source ~/.bash_profile
```

4.5. 关于 SBU

许多人想在编译和安装各个软件包之前，了解这一过程大概需要多少时间。由于 Linux From Scratch 可以在许多不同系统上构建，我们无法直接给出估计时间。例如，最大的软件包 (gcc) 在最快的系统上只要大约 5 分钟就能构建好，而在一些较慢的系统上需要若干天！因此，我们不提供实际时间，而是以标准构建单位 (SBU) 衡量时间。

下面给出标准构建单位的测量方法。本书中构建的第一个软件包是第 5 章中的 Binutils，定义使用单个 CPU 核心编译它需要的时间为标准构建单位，缩写为 SBU。其他软件包的编译时间用 SBU 为单位表示。

例如，考虑一个编译时间是 4.5 SBU 的软件包。如果在您的系统上，需要 4 分钟来编译和安装第一轮的 Binutils，那么大概需要 18 分钟才能构建这个软件包。幸运的是，多数软件包的构建时间少于 1 SBU。

SBU 不是完全准确的，这是由于它受到许多因素的影响，包括宿主系统的 GCC 版本。SBU 只能用来估计安装一个软件包可能需要的时间，估计结果的误差在个别情况下可能达到几十分钟。

在一些较新的系统中，主板能够控制系统时钟频率，可以使用 **powerprofilesctl** 或类似命令调整主板控制系统时钟的策略。LFS 不包含这类命令，但宿主系统可能提供它。在 LFS 构建完成后，可以按照 BLFS power-profiles-daemon 页面的说明，为系统加入这一功能。在测量软件包的构建时间前，最好将系统电源策略设为最大性能模式 (这也会允许系统功率达到上限)。否则测量的 SBU 值可能不准确，这是因为系统在构建第一遍的 Binutils 和其他软件包时的反应速度可能不同。注意即使在构建它们时使用了相同的电源策略，SBU 值仍然可能明显偏离正常值，因为一些电源策略会在系统空闲时降低其反应速度。将电源策略设为“性能模式”可以尽量减轻这一问题。当然，这样做还会使得构建 LFS 更快。

如果宿主系统提供 **powerprofilesctl** 命令，执行 **powerprofilesctl set performance** 命令以选择 `performance` (性能) 策略。一些宿主发行版使用 **tuned-adm** 命令代替 **powerprofilesctl** 命令管理电源策略，在这些发行版中，执行 **tuned-adm profile throughput-performance** 命令以选择 `throughput-performance` (优化吞吐量) 策略。

注意

如上使用多个处理器时，使用 SBU 估计构建时间会出现更大的误差。某些情况下，还会导致 make 命令失败。另外，分析构建过程的的输出也会变得困难，因为不同进程的输出行会交错在一起。如果在构建过程中出现问题，需要使用单处理器进行构建，才能更好地分析错误消息。

我们在计算 SBU 值时，除了对于第一遍的 Binutils 本身使用单个 CPU 核心外，都使用四个 CPU 核心 (-j4) 计时。第 8 章中的时间还会包含运行软件包退化测试的时间，除非另有说明。

4.6. 关于测试套件

多数软件包提供测试套件，一般来说，为新构建的软件包运行测试套件是个好主意，这可以进行一次“完整性检查”，从而确认所有东西编译正确。如果测试套件中的所有检验项目都能通过，一般就可以证明这个软件包像开发者期望的那样运行。然而，这并不保证软件包完全没有错误。

某些软件包的测试套件比其他的更为重要。例如，组成核心工具链的几个软件包 — GCC、Binutils 和 Glibc 的测试套件就最为重要，因为这些软件包在系统的正常工作中发挥中心作用。GCC 和 Glibc 的测试套件需要运行很长时间，特别是在较慢的硬件上，但我们仍然强烈推荐运行它们。

注意

在第 5 章和第 6 章中运行测试套件没有意义，因为测试程序是使用交叉编译器编译的，可能无法在构建它们的宿主系统运行。

在运行 Binutils 和 GCC 的测试套件时，较常见的问题是伪终端 (PTY) 被耗尽。这会导致大量测试出现失败结果。这种现象有多种可能原因，但最常见的原因是宿主系统没有正确设置 `devpts` 文件系统。关于这个问题的更多细节在 https://www.linuxfromscratch.org/lfs/faq.html#no-ptys 中进行了讨论。

一些软件包的测试套件由于开发者已经知道的原因而失败，且这些失败已被判定为并不重要。参照 https://www.linuxfromscratch.org/lfs/build-logs/12.4/ 中的构建日志，来检查这些失败是否符合预期。本书中的所有测试结果都可以在该网址查询。

第 III 部分 构建 LFS 交叉工具链和临时工具

重要的提前阅读资料

概述

本书的这一部分被分为三个阶段：首先，构建一个交叉编译器和与之相关的库；然后，使用这个交叉工具链构建一些工具，并使用保证它们和宿主系统分离的构建方法；最后进入 chroot 环境 (它能够进一步提高与宿主的隔离度)，并构建剩余的，在构建最终的系统时必须的工具。

重要

从本节开始，我们将进行构建新系统的实际工作。请非常认真地严格执行本书给出的指示。您应该尽量理解这些操作的含义，无论您急于完成构建的心情多么迫切，都不能不加思考地将命令直接输入。在您无法理解命令时要阅读描述它们的文本。另外，注意跟踪您输入的命令和它们的输出，您可以将输出通过 **tee** 工具发送到文件。这样如果出现了问题，可以更好地进行诊断。

下一节将给出构建过程的技术说明。再下一节包含**非常重要**的通用说明。

工具链技术说明

本节综合地解释构建方法中的逻辑和技术细节。不要试图立刻理解本节的所有内容。在实际完成一次系统构建后，可以更容易地理解本节。在整个构建过程中，您随时可以重新阅读本节。

第 5 章和第 6 章的总目标是构造一个临时环境，它包含一组可靠的，能够与宿主系统完全分离的工具。这样，通过使用 **chroot** 命令，其余各章中执行的命令就被限制在这个临时环境中。这确保我们能够干净、顺利地构建 LFS 系统。整个构建过程被的设计目标是尽量降低新读者可能面临的风险，同时提供尽可能多的教育价值。

构建过程是基于交叉编译过程的。交叉编译通常被用于为一台与本机完全不同的计算机构建编译器及其工具链。这对于 LFS 并不严格必要，因为新系统运行的机器就是构建它时使用的。但是，交叉编译拥有一项重要优势：任何交叉编译产生的程序都不可能依赖于宿主环境。

关于交叉编译

注意

LFS 手册并不是 (也不包含) 一份通用的，构建交叉 (或本地) 工具链的指南。除非您完全明白自己在干什么，请勿使用手册中的命令构建交叉工具链并用于构建 LFS 以外的用途。

已知安装第二遍的 GCC 会破坏交叉工具链。我们认为这并不是 bug，因为第二遍的 GCC 是本书中最后一个交叉编译的软件包，因此除非在未来我们确实需要在第二遍的 GCC 之后交叉编译某个软件包，我们不会“修复”这个问题。

交叉编译涉及一些概念，值得专门用一节讨论。尽管您可以在初次阅读时跳过本节，但在之后重新阅读本节，能帮助您更全面地理解构建过程。

首先我们定义讨论交叉编译时常用的术语。

build
: 指构建程序时使用的机器。注意在某些其他章节，这台机器被称为“host”(宿主)。

host
: 指将来会运行被构建的程序的机器。注意这里说的“host”与其他章节使用的“宿主”(host) 一词不同。

target
只有编译器使用这个术语。编译器为这台机器产生代码。它可能和 build 与 host 都不同。

例如，我们考虑下列场景 (有时称为“Canadian Cross”)。我们仅在一台运行缓慢的机器上有编译器，称这台机器为 A，这个编译器为 ccA。我们还有一台运行较快的机器 (B)，但它没有安装编译器，而我们希望为另一台缓慢的机器 (C) 生成代码。如果要为 C 构建编译器，可以通过三个阶段完成：

阶段	Build	Host	Target	操作描述
1	A	A	B	在机器 A 上，使用 ccA 构建交叉编译器 cc1
2	A	B	C	在机器 A 上，使用 cc1 构建交叉编译器 cc2
3	B	C	C	在机器 B 上，使用 cc2 构建交叉编译器 ccC

这样，我们可以为机器 C 使用 cc2 在快速的机器 B 上构建所有其他程序。需要注意的是，除非 B 能运行为 C 编译的程序，则在 C 上实际运行它们之前，无法测试它们的功能。例如，如果要测试 ccC，我们可能需要增加第四个阶段：

阶段	Build	Host	Target	操作描述
4	C	C	C	在机器 C 上，用 ccC 重新构建它本身，并测试

在上面的例子中，只有 cc1 和 cc2 是交叉编译器，它们为与它们本身运行的机器不同的机器产生代码。而另外的编译器 ccA 和 ccC 为它们本身运行的机器产生代码，它们称为本地编译器。

LFS 的交叉编译实现

注意

本书中涉及交叉编译的软件包都使用基于 autoconf 的构建系统。基于 autoconf 的构建系统使用形如 CPU-供应商-内核-操作系统，称为三元组的名称表示目标系统类型。由于供应商字段通常无关紧要，autoconf 允许省略它。

好奇的读者可能会问，为什么一个“三元组”却包含四个部分。这是由于内核和操作系统两个字段起源于一个“系统”字段。至今，一些系统仍然用三字段的格式准确描述，例如，`x86_64-unknown-freebsd`。但是对于其他一些系统，即使两个系统使用相同的内核，它们也可能截然不同，以至于不能使用相同的三元组。例如，运行在智能手机的 Android 和运行在 ARM64 服务器的 Ubuntu 完全不同，尽管它们使用相同类型的 CPU (ARM64) 和相同的内核 (Linux)。

在没有仿真中间层的情况下，显然不能在智能手机上运行用于服务器的可执行文件，反之亦然。因此，“系统”字段被拆分为内核和操作系统两部分，以准确区分这些系统。对于本例，Android 被表示为 `aarch64-unknown-linux-android`，而 Ubuntu 被表示为 `aarch64-unknown-linux-gnu`。

“三元组”这个词汇被沿用下来。有一种简单方法可以获得您的机器的三元组，即运行许多软件包附带的 **config.guess** 脚本。解压缩 binutils 源码，然后输入 **`./config.guess`** 运行脚本，并观察输出。例如，对于 32 位 Intel 处理器，输出应该是 i686-pc-linux-gnu。而在 64 位系统上输出应该是 x86_64-pc-linux-gnu。在许多 Linux 系统上，更简单的 **gcc -dumpmachine** 命令也会输出类似的信息。

您还需要注意平台的动态链接器的名称，它又被称为动态加载器 (不要和 Binutils 中的普通链接器 **ld** 混淆)。动态链接器由 Glibc 提供，它寻找并加载程序所需的共享库，为程序运行做好准备，然后运行程序。在 32 位 Intel 机器上动态链接器的名称是 `ld-linux.so.2` (在 64 位系统上是 `ld-linux-x86-64.so.2`)。为了准确确定动态链接器名称，可以从宿主系统找一个二进制可执行文件，然后执行：**`readelf -l <二进制文件名> | grep interpreter`** 并观察输出。可以在 Glibc wiki 页面找到包含所有平台的权威参考。

在进行交叉编译时需要注意两个关键问题：

- 在生成和处理应该在 host 执行的机器码时，必须使用交叉工具链。注意构建过程仍然可能调用 build 的本地工具链以生成应该在 build 执行的机器码，例如构建系统可能首先用本地工具链编译一个生成器，然后用该生成器生成一个 C 源代码文件，最后再用交叉工具链编译生成的代码，使其能在 host 运行。

  在使用基于 autoconf 的构建系统时，可以通过 `--host` 选项指定 host 三元组以满足上述需求。该选项使得构建系统使用带有 `<the host triplet>` (host 三元组) 前缀的工具链组件生成和处理用于 host 的机器码；例如，使用 **`<the host triplet>`-gcc** 作为编译器，使用 **`<the host triplet>`-readelf** 作为 **readelf** 工具。

- 构建系统不应试图执行任何应该在 host 运行的机器码。例如，在本地编译某个工具时，可能通过运行它并传递 `--help` 选项，再处理其输出以生成它的手册页，但在交叉编译时通常不能这么做，因为该工具可能根本无法在 build 运行：显然不可能在 x86 CPU 上运行 ARM64 机器码 (如果没有模拟器的话)。

  在使用基于 autoconf 的构建系统时，可以通过所谓的“交叉编译模式”满足这一需求，在此模式下需要在构建时运行为 host 编译的机器码的可选功能会被全部禁用。在显式指定了 host 三元组的情况下，“交叉编译模式”当且仅当 **configure** 无法运行已经为 host 编译的样品程序，或通过 `--build` 显式指定了和 host 三元组不同的 build 三元组时被启用。

为了为 LFS 临时系统交叉编译软件包，首先微调系统三元组，将其"vendor" 字段改为 "lfs" 后写入LFS_TGT 环境变量，并将该变量值通过 --host 指定为 host 三元组，这样在生成和处理作为 LFS 临时系统一部分的机器码时，构建系统就会使用交叉工具链。此外，我们还需要启用“交叉编译模式”：host，即 LFS 临时系统所用的机器码尽管能够在当前 CPU 上运行，却可能使用 build (宿主发行版) 没有提供的库；即使库恰好存在，这些机器码也可能使用不存在或定义不同的代码或数据。在为 LFS 临时系统进行交叉编译时，我们不能期望 **configure** 通过运行样品程序来检测这一问题：样品程序只引用 libc 中的少数组件，而宿主发行版的 libc 很可能提供了这些组件 (或许，除非在宿主发行版使用了不同的 libc 实现，如 Musl)，故样品程序不会像那些真正有功能的程序大概会发生的一样运行失败。故我们必须显式指定 build 三元组，以启用“交叉编译模式”。我们指定的三元组就是默认值，即 **config.guess** 输出的，未经修改的系统三元组，但正如前文所述，“交叉编译模式”依赖于是否显式指定了该三元组。

在构建交叉链接器和交叉编译器时，我们指定 --with-sysroot 选项，以告知它们应该在何处搜索为 host 生成代码所需的文件。这几乎保证了第 6 章中构建的其他程序不会连接到 build 中的库。这里使用“几乎”这个词是因为 **libtool**，一个包装编译器和链接器的“兼容层”，可能自作聪明地传递一些导致链接器能找到 build 库的选项。为了防止这一问题，我们需要删除 libtool 档案 (.la) 文件，并修复 Binutils 代码随附的过时的 libtool 副本。

阶段	Build	Host	Target	操作描述
1	pc	pc	lfs	在 pc 上使用 cc-pc 构建交叉编译器 cc1
2	pc	lfs	lfs	在 pc 上使用 cc1 构建 cc-lfs
3	lfs	lfs	lfs	在 lfs 上使用 cc-lfs 重新构建 (同时可以测试) 它本身

在上表中，“在 pc 上” 意味着命令在已经安装好的发行版中执行。“在 lfs 上” 意味着命令在 chroot 环境中执行。

现在，关于交叉编译，还有更多要处理的问题：C 语言并不仅仅是一个编译器；它还规定了一个标准库。在本书中，我们使用 GNU C 运行库，即 glibc (除此之外，还有名为 "musl" 的另一种 C 运行库实现)。它必须为 lfs 目标机器使用交叉编译器 cc1 编译。但是，编译器本身使用一个库，实现汇编指令集并不支持的一些复杂指令。这个内部库称为 libgcc，它必须链接到 glibc 库才能实现完整功能。另外，C++ 标准库 (libstdc++) 也必须链接到 glibc。为了解决这个”先有鸡还是先有蛋“的问题，只能先构建一个降级的 cc1，它的 libgcc 缺失线程和异常等功能，再用这个降级的编译器构建 glibc (这不会导致 glibc 缺失功能)，再构建 libstdc++。但是这种方法构建的 libstdc++ 会缺失一些依赖于 libgcc 的功能。

上一自然段的结论是 cc1 无法使用功能降级的 libgcc 构建功能完整的 libstdc++，但这是我们在阶段 2 构建 C/C++ 库时唯一可用的编译器。如同前文所述，我们不能在 pc (宿主发行版) 运行 cc-lfs，因为它可能使用 build (宿主发行版) 未提供的库，代码，或数据。因此在构建第二阶段的 gcc 时，我们覆盖库文件搜索路径，以将 libstdc++ 链接到刚刚重新构建的 libgcc，而不是旧的，功能降级的版本。这样重新构建的 libstdc++ 就会具有完整的功能。

在第 8 章 (或者也可以称为“第三阶段”) 中，我们会构建 LFS 需要的所有软件包。即使某个软件包在之前的章节已被构建，我们仍然重新构建它。重新构建的最主要原因是将软件包稳定下来：如果我们在完整的 LFS 系统上重新安装一个 LFS 软件包，则重新安装到系统中的内容应该和第 8 章中初次安装的完全一致。第 6 章和第 7 章中的临时软件包无法满足这一条件，因为它们的一些可选功能由于缺失依赖或使用“交叉编译模式”而被禁用。另外，进行重新构建还有一个次要原因，即运行软件包的测试套件。

构建过程的其他细节

交叉编译器会被安装在独立的 `$LFS/tools` 目录，因为它不属于最终构建的系统。

我们首先安装 Binutils。这是由于 GCC 和 Glibc 的 **configure** 脚本首先测试汇编器和链接器的一些特性，以决定启用或禁用一些软件特性。初看起来这并不重要，但没有正确配置的 GCC 或者 Glibc 会导致工具链中潜伏的故障。这些故障可能到整个构建过程快要结束时才突然爆发，不过在花费大量无用功之前，测试套件的失败通常可以将这类错误暴露出来。

Binutils 将汇编器和链接器安装在两个位置，一个是 `$LFS/tools/bin`，另一个是 `$LFS/tools/$LFS_TGT/bin`。这两个位置中的工具互为硬链接。链接器的一项重要属性是它搜索库的顺序，通过向 **ld** 命令加入 `--verbose` 参数，可以得到关于搜索路径的详细信息。例如，**ld --verbose | grep SEARCH** 会输出当前的搜索路径及其顺序。(注意这条命令只有在以 `lfs` 用户身份操作时才能正常工作。如果在阅读后续章节的过程中复习这里的内容，可能需要将 **$LFS_TGT-ld** 替换为 **ld**。)

下一步安装 GCC。在执行它的 **configure** 脚本时，您会看到类似下面这样的输出：

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

基于我们上面论述的原因，这些输出非常重要。这也说明 gcc 的配置脚本没有在 PATH 变量指定的目录中搜索工具。然而，在 **gcc** 的实际运行中，未必会使用同样的搜索路径。为了查询 **gcc** 会使用哪个链接器，需要执行以下命令：**$LFS_TGT-gcc -print-prog-name=ld**。(同样，如果在阅读后续章节的过程中复习这里的内容，可能需要移除命令中的 **$LFS_TGT-** 前缀。)

通过向 **gcc** 传递 `-v` 参数，可以知道在编译程序时发生的细节。例如，**$LFS_TGT-gcc -v `example.c`** (如果在复习这里的内容，可能需要移除 **$LFS_TGT**) 会输出预处理、编译和汇编阶段中的详细信息，包括 **gcc** 的包含文件搜索路径和顺序。

下一个步骤是：安装“净化的” (sanitized) Linux API 头文件。这些头文件允许 C 标准库 (glibc) 与 Linux 内核提供的各种特性交互。

下一步安装 Glibc。正如前文所述，我们使用 `--host=$LFS_TGT` 选项以保证构建系统使用那些带有 `$LFS_TGT-` 前缀的工具，并使用 `--build=$(../scripts/config.guess)` 选项启用“交叉编译模式”。最后，我们使用 `DESTDIR` 变量，将构建得到的 Glibc 强制安装到 LFS 文件系统中。

正如前文所述，接下来构建 C++ 标准库，然后是第 6 章中的其他程序，必须交叉编译这些程序才能打破构建时的循环依赖。构建并安装这些软件包的步骤类似构建并安装 glibc。

在第 6 章的末尾，构建 LFS 本地编译器。首先使用和其他程序相同的 `DESTDIR` 第二次构建 binutils，然后第二次构建 GCC，构建时省略一些不重要的库。

在第 7 章中，进入 chroot 环境后，临时性地安装工具链的正常工作所必须的程序。此后，核心工具链成为自包含的本地工具链。在第 8 章中，构建，测试，并最终安装所有软件包，它们组成功能完整的系统。

编译过程的一般说明

小心

在 LFS 的开发周期中，我们经常修改本书中的指令，以适应更新的软件包，或利用新版本软件包提供的新特性。混用不同版本 LFS 手册中的指令会导致难以察觉的问题。这种情况通常是重用为之前的某个 LFS 版本创建的脚本所致。我们强烈反对这种重用。如果您出于某种原因，一定要重用为旧版本 LFS 创建的脚本，您必须非常小心地更新脚本，使其内容和当前版本的 LFS 手册一致。

下面是在构建每个软件包时都要注意的事项：

- 某些软件包在编译前需要打补丁，然而补丁只在绕过特定问题时才需要。补丁常常在本章和下一章都要使用，但有时，对于多次构建的软件包，在初次构建时可能并不需要补丁。因此，如果发现本书给出的步骤中没有使用某个下载好的补丁，这是正常的，不必担心。在应用补丁时可能会出现关于 offset 或者 fuzz 的警告信息。不用担心这些警告，补丁还是会成功应用到源码上的。
- 在编译大多数软件包时，屏幕上都会出现一些警告。这是正常的，可以放心地忽略。这些警告就像它们描述的那样，是关于一些过时的，但并不是错误的 C 或 C++ 语法。C 标准经常改变，一些软件包仍然在使用旧的标准。这并不是一个严重的问题，但确实会触发警告。
- 最后确认 `LFS` 环境变量是否配置正确：

```
echo $LFS
```

确认上述命令输出 LFS 分区挂载点的路径，如果使用了本书的例子，就是 `/mnt/lfs`。

- 最后强调两个重要事项：

重要

本书中的命令假设宿主系统需求中的所有内容，包括符号链接，都被正确设置：

- **bash** 是正在使用的 shell。
- **sh** 是指向 **bash** 的符号链接。
- **/usr/bin/awk** 是指向 **gawk** 的符号链接。
- **/usr/bin/yacc** 是指向 **bison** 的符号链接，或者一个执行 bison 的小脚本。

重要

下面给出软件包构建过程的概要。

1. 把所有的源码包和补丁放在一个能够从 chroot 环境访问的目录，例如 `/mnt/lfs/sources/`。
2. 切换到 `/mnt/lfs/sources/` 目录。
3. 对于每个软件包：
   a. 使用 **tar** 程序，解压需要构建的软件包。在第 5 章和第 6 章中解压软件包时，确认您以用户 lfs 的身份登录。

      除了使用 **tar** 命令解压源码包外，不要使用其他任何将源代码目录树置入工作目录的方法。特别需要注意的是，使用 **cp -R** 从其他位置复制源代码目录树会破坏其中的时间戳，从而导致构建失败。
   b. 切换到解压源码包时产生的目录。
   c. 根据指示构建软件包。
   d. 构建完成后，切换回包含所有源码包的目录。
   e. 除非另有说明，删除解压出来的目录。

第 5 章 编译交叉工具链

5.1. 概述

本章展示如何构建交叉编译器和相关工具。尽管本书中的交叉编译是伪装的，但其原理和构建真实的交叉工具链是一致的。

本章中编译的程序会被安装在 `$LFS/tools` 目录中，以将它们和后续章节中安装的文件分开。但是，本章中编译的库会被安装到它们的最终位置，因为这些库在我们最终要构建的系统中也存在。

5.2. Binutils-2.45 - 第一遍

Binutils 包含汇编器、链接器以及其他用于处理目标文件的工具。

估计构建时间: 1 SBU
需要硬盘空间: 678 MB

5.2.1. 安装交叉工具链中的 Binutils

注意

返回并重新阅读编译过程的一般说明一节。仔细理解那些标为“重要”的说明，以防止之后出现问题。

首先构建 Binutils 相当重要，因为 Glibc 和 GCC 都会对可用的链接器和汇编器进行测试，以决定可以启用它们自带的哪些特性。

Binutils 文档推荐创建一个新的目录，以在其中构建 Binutils：

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

注意

为了衡量本书其余部分使用的 SBU 值，需要测量本软件包从配置开始直到第一次安装花费的时间。为了容易地完成测量，可以将命令包装在 **time** 命令中，就像这样：`time { ../configure ... && make && make install; }`。

现在，准备编译 Binutils：

```
../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
```

配置选项的含义：

注意

和其他一些软件包不同，**./configure --help** 不会列出所有配置选项。例如，为了查看 `--with-sysroot` 选项的相关信息，需要运行 **ld/configure --help**。运行 **./configure --help=recursive** 可以列出所有选项。

`--prefix=$LFS/tools`
这告诉配置脚本准备将 Binutils 程序安装在 `$LFS/tools` 目录中。

`--with-sysroot=$LFS`
该选项告诉构建系统，交叉编译时在 $LFS 中寻找目标系统的库。

`--target=$LFS_TGT`
由于 `LFS_TGT` 变量中的机器描述和 **config.guess** 脚本的输出略有不同，这个开关使得 **configure** 脚本调整 Binutils 的构建系统，以构建交叉链接器。

`--disable-nls`
该选项禁用临时工具不需要的国际化功能。

`--enable-gprofng=no`
该选项禁用临时工具不需要的 gprofng 工具。

`--disable-werror`
该选项防止宿主系统编译器警告导致构建失败。

`--enable-new-dtags`
该选项使得链接器使用“runpath”标记在可执行程序和共享库中嵌入库文件搜索路径，而非传统的“rpath”标记。这样能使得调试动态链接的可执行程序更容易，且能绕过一些软件包的测试套件中潜藏的问题。

`--enable-default-hash-style=gnu`
默认情况下，链接器会为共享库和动态链接的可执行文件同时生成 GNU 风格的散列表和经典的 ELF 散列表。散列表仅供动态链接器进行符号查询。LFS 系统的动态链接器 (由 Glibc 软件包提供) 总是使用查询更快的 GNU 风格散列表。因此经典 ELF 散列表完全没有意义。该选项使得链接器在默认情况下只生成 GNU 风格散列表，以避免为生成和存储经典 ELF 散列表浪费时间和空间。

然后编译该软件包：

```
make
```

安装该软件包：

```
make install
```

该软件包的更多信息可以在第 8.20.2 节 “Binutils 的内容”中找到。

5.3. GCC-15.2.0 - 第一遍

GCC 软件包包含 GNU 编译器集合，其中有 C 和 C++ 编译器。

估计构建时间: 3.8 SBU
需要硬盘空间: 5.4 GB

5.3.1. 安装交叉工具链中的 GCC

GCC 依赖于 GMP、MPFR 和 MPC 这三个包。由于宿主发行版未必包含它们，我们将它们和 GCC 一同构建。将它们都解压到 GCC 源码目录中，并重命名解压出的目录，这样 GCC 构建过程就能自动使用它们:

注意

对于本章内容有一些很常见的误解。该软件包的构建过程就像之前（软件包构建说明）解释的那样，首先，解压 gcc-15.2.0 压缩包，然后切换到解压出的目录中。之后才能执行后续的命令。

```
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
```

对于 x86_64 平台，还要设置存放 64 位库的默认目录为 “lib”：

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

注意

这里演示了 `-i.orig` 选项的用法。它使得 **sed** 将 `t-linux64` 文件复制到 `t-linux64.orig`，然后再编辑原本的 `t-linux64` 文件。这样，之后可以使用 **diff -u gcc/config/i386/t-linux64{.orig,}** 命令展示 **sed** 命令所做的修改。对于本书中的其他软件包，我们会简单使用 `-i` (并不复制原始文件，而是直接编辑它)，但如果您需要保留原始文件的一份副本，可以随时将其替换为 `-i.orig`。

GCC 文档建议在一个新建的目录中构建 GCC:

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

准备编译 GCC:

```
../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++
```

配置选项的含义:

`--with-glibc-version=2.42`
该选项指定目标系统将要使用的 Glibc 版本。这与宿主系统的 libc 没有关系，因为第一遍的 GCC 产生的所有代码都会在与宿主系统的 libc 完全隔离的 chroot 环境中运行。

`--with-newlib`
由于现在没有可用的 C 运行库，使用该选项保证构建 libgcc 时 inhibit_libc 常量被定义，以防止编译任何需要 libc 支持的代码。

`--without-headers`
在创建完整的交叉编译器时，GCC 需要与目标系统兼容的标准头文件。由于我们的特殊目的，这些头文件并不必要。这个开关防止 GCC 查找它们。

`--enable-default-pie 和 --enable-default-ssp`
它们使得 GCC 在编译程序时默认启用一些增强安全性的特性 (详见第 8 章中的关于 PIE 和 SSP 的说明)。在本阶段并没有使用它们的必要性，但是尽早使用它们能够使得临时安装和最终安装的软件包更相近，这样构建过程更加稳定。

`--disable-shared`
这个开关强制 GCC 静态链接它的内部库。我们必须这样做，因为动态库需要目标系统中尚未安装的 Glibc。

`--disable-multilib`
在 x86_64 平台上，LFS 不支持 multilib 配置。这个开关对于 x86 来说可有可无。

`--disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx`
这些开关禁用对于线程、libatomic、libgomp、libquadmath、libssp、libvtv，以及 C++ 标准库的支持。在构建交叉编译器时它们可能编译失败，而且在交叉编译临时 libc 时并不需要它们。

`--enable-languages=c,c++`
这个选项保证只构建 C 和 C++ 编译器。目前只需要这两个语言。

执行以下命令编译 GCC:

```
make
```

安装该软件包:

```
make install
```

刚刚构建的 GCC 安装了若干内部系统头文件。其中的 `limits.h` 一般来说，应该包含对应的系统头文件 `limits.h`，在我们的 LFS 环境中，就是 `$LFS/usr/include/limits.h`。然而，在构建 GCC 的时候，`$LFS/usr/include/limits.h` 还不存在，因此 GCC 安装的内部头文件是一个不完整的、自给自足的文件，不包含系统头文件提供的扩展特性。这对于构建临时的 Glibc 已经足够了，但后续工作将需要完整的内部头文件。使用以下命令创建一个完整版本的内部头文件，该命令与 GCC 构建系统在一般情况下生成该头文件的命令是一致的：

注意

下列命令作为实例，展示了命令行代换操作的两种不同写法：反引号和 `$()` 结构。可以将该命令改写为使用一种写法完成两次代换，但我们这里特意展示如何混用两种写法。一般来说 `$()` 这种写法更常用。

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

该软件包的详细信息在第 8.29.2 节 “GCC 的内容”可以找到。

5.4. Linux-6.16.1 API 头文件

Linux API 头文件 (在 linux-6.16.1.tar.xz 中) 导出内核 API 供 Glibc 使用。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 1.7 GB

5.4.1. 安装 Linux API 头文件

Linux 内核需要导出应用程序编程接口 (API) 以供系统的 C 运行库 (例如 LFS 中的 Glibc) 使用。这通过净化内核源码包中提供的若干 C 头文件完成。

确保软件包中没有遗留陈旧的文件:

```
make mrproper
```

下面从源代码中提取用户可见的头文件。我们不能使用推荐的 make 目标“headers_install”，因为它需要 rsync，这个程序在宿主系统中未必可用。头文件会先被放置在 `./usr` 目录中，之后再将它们复制到最终的位置。

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

5.4.2. Linux API 头文件的内容

安装的头文件: /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, 以及 /usr/include/xen/*.h
安装的目录: /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, 以及 /usr/include/xen

简要描述

`/usr/include/asm/*.h`	Linux API 汇编头文件
`/usr/include/asm-generic/*.h`	Linux API 通用汇编头文件
`/usr/include/drm/*.h`	Linux API DRM 头文件
`/usr/include/linux/*.h`	Linux API Linux 头文件
`/usr/include/misc/*.h`	Linux API 杂项头文件
`/usr/include/mtd/*.h`	Linux API MTD 头文件
`/usr/include/rdma/*.h`	Linux API RDMA 头文件
`/usr/include/scsi/*.h`	Linux API SCSI 头文件
`/usr/include/sound/*.h`	Linux API 音频头文件
`/usr/include/video/*.h`	Linux API 视频头文件
`/usr/include/xen/*.h`	Linux API Xen 头文件

5.5. Glibc-2.42

Glibc 软件包包含主要的 C 语言库。它提供用于分配内存、检索目录、打开和关闭文件、读写文件、字符串处理、模式匹配、算术等用途的基本子程序。

估计构建时间: 1.4 SBU
需要硬盘空间: 870 MB

5.5.1. 安装 Glibc

首先，创建一个 LSB 兼容性符号链接。另外，对于 x86_64，创建一个动态链接器正常工作所必须的符号链接：

```
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
```

注意

以上命令是正确的。**ln** 命令有多种语法变式，因此在报告看似错误的命令之前，请先阅读 **info coreutils ln** 和 ln(1)。

一些 Glibc 程序使用与 FHS 不兼容的 `/var/db` 目录存放它们的运行时数据。应用一个补丁，使得这些程序在 FHS 兼容的位置存放运行时数据：

```
patch -Np1 -i ../glibc-2.42-fhs-1.patch
```

Glibc 文档推荐在一个新建的目录中构建 Glibc：

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

确保将 **ldconfig** 和 **sln** 工具安装到 `/usr/sbin` 目录中：

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

下面，准备编译 Glibc：

```
../configure                             \
      --prefix=/usr                      \
      --host=$LFS_TGT                    \
      --build=$(../scripts/config.guess) \
      --disable-nscd                     \
      libc_cv_slibdir=/usr/lib           \
      --enable-kernel=5.4
```

配置选项的含义：

`--host=$LFS_TGT, --build=$(../scripts/config.guess)`
在它们的共同作用下，Glibc 的构建系统将自身配置为使用 `$LFS/tools` 中的交叉链接器和交叉编译器，进行交叉编译。

`--enable-kernel=5.4`
该选项使得编译得到的 Glibc 库只支持 5.4 版或更新版本的 Linux 内核，这样就不会编译那些为更老内核准备的替代方案。

`libc_cv_slibdir=/usr/lib`
在 64 位机器上，这保证将库安装到 /usr/lib，而不是默认的 /lib64。

`--disable-nscd`
不构建目前已经没有作用的命名服务缓存守护程序。

在当前阶段，可能出现下列警告：

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

msgfmt 程序的缺失或不兼容一般是无害的。**msgfmt** 程序是 Gettext 软件包的一部分，宿主发行版应该提供它。

注意

有报告称该软件包在并行构建时可能失败，如果发生了这种情况，加上 `-j1` 选项重新执行 make 命令。

编译该软件包：

```
make
```

安装该软件包：

警告

如果 `LFS` 没有正确设定，而且您不顾本书的建议，以 `root` 用户的身份进行构建，下面的命令会将新构建的 Glibc 安装到您的宿主系统中，这几乎必然导致宿主系统完全无法使用。因此，在运行下面的命令前，请再次检查该环境变量是否已经正确设定，并确认您并非以 `root` 身份操作。

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

make install 选项的含义：

`DESTDIR=$LFS`
多数软件包使用 `DESTDIR` 变量指定软件包应该安装的位置。如果不设定它，默认值为根 (/) 目录。这里我们指定将软件包安装到 `$LFS`，它在第 7.4 节 “进入 Chroot 环境”之后将成为根目录。

改正 **ldd** 脚本中硬编码的可执行文件加载器路径：

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

现在交叉工具链已经就位，重要的是再次确认编译和链接像我们期望的一样正常工作。为此，进行下列完整性检查：

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

上述命令不应该出现错误，最后一行命令输出的结果应该 (不同平台的动态链接器名称可能不同) 是：

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

注意该路径不应包含 `/mnt/lfs` (或者您为 `LFS` 变量自行设置的值)。该路径在编译得到的程序运行时才会被解析，交叉编译得到的程序应该在进入 chroot 环境后才会运行，而在 chroot 环境中内核会将 `$LFS` 视为根目录 (/)。

下面确认我们在使用正确的启动文件：

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

以上命令应该输出：

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

确认编译器能正确查找头文件：

```
grep -B3 "^ $LFS/usr/include" dummy.log
```

该命令应当输出：

```
#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
```

同样要注意，以您的目标三元组命名的目录由于您体系结构的不同，可能和以上不同。

下一步确认新的链接器使用了正确的搜索路径：

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

那些包含 '-linux-gnu' 的路径应该忽略，除此之外，以上命令应该输出：

```
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 位系统可能使用不同的路径，但无论如何，需要关注的重点是所有这些路径都应该以等号 (=) 起始，链接器会自动以为其配置的 sysroot 目录替换等号。

之后确认我们使用了正确的 libc：

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

以上命令应该输出：

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

确认 GCC 使用了正确的动态链接器：

```
grep found dummy.log
```

以上命令应该输出 (不同平台的动态链接器名称可能不同)：

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

如果输出和以上描述不符，或者根本没有输出，那么必然有什么地方出了严重错误。检查并重新跟踪以上步骤，找到问题的原因，并修复它。在继续构建前必须解决这里发现的所有问题。

在确认一切工作良好后，删除测试文件：

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

注意

在下一章中，构建各软件包的过程可以作为对工具链是否正常构建的额外检查。如果 一些软件包，特别是第二遍的 Binutils 或者 GCC 不能构建，说明在之前安装 Binutils，GCC，或者 Glibc 时出了问题。

该软件包的详细信息可以在第 8.5.3 节 “Glibc 的内容”中找到。

5.6. GCC-15.2.0 中的 Libstdc++

Libstdc++ 是 C++ 标准库。我们需要它才能编译 C++ 代码 (GCC 的一部分用 C++ 编写)。但在构建第一遍的 GCC时我们不得不暂缓安装它，因为 Libstdc++ 依赖于当时还没有安装到目标目录的 Glibc。

估计构建时间: 0.2 SBU
需要硬盘空间: 1.3 GB

5.6.1. 安装目标系统的 Libstdc++

注意

Libstdc++ 是 GCC 源代码的一部分。您应该先解压 GCC 源码包并切换到解压出来的 `gcc-15.2.0` 目录。

为 Libstdc++ 创建一个单独的构建目录，并进入它:

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

准备编译 Libstdc++:

```
../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
```

配置选项的含义:

`--host=...`
指定使用我们刚刚构建的交叉编译器，而不是 `/usr/bin` 中编译器。

`--disable-libstdcxx-pch`
这个开关防止安装预编译头文件，在这个阶段不需要它们。

`--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/15.2.0`
该选项指定包含文件的安装路径。因为 Libstdc++ 是 LFS 的 C++ 标准库，这个安装路径应该与 C++ 编译器 (**$LFS_TGT-g++**) 搜索 C++ 标准头的位置一致。在正常的构建过程中，这项信息被构建系统由顶层目录自动传递给 Libstdc++ **configure** 脚本。但我们没有使用顶层目录构建系统，因此需要明确指定该选项。C++ 编译器会将 sysroot 路径 `$LFS` (我们在构建第一遍的 GCC 时指定了它) 附加到包含文件搜索目录之前，因此它实际上会搜索 `$LFS/tools/$LFS_TGT/include/c++/15.2.0`。该选项和后续使用的 `DESTDIR` 变量 (在 **make install** 命令中) 一起，确保将头文件安装到这一路径。

运行以下命令编译 Libstdc++:

```
make
```

安装这个库:

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

移除对交叉编译有害的 libtool 档案文件:

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

该软件包的详细信息在第 8.29.2 节 “GCC 的内容”可以找到。

第 6 章 交叉编译临时工具

6.1. 概述

本章展示如何使用刚刚构建的交叉工具链对基本工具进行交叉编译。这些工具会被安装到它们的最终位置，但现在还无法使用。基本操作仍然依赖宿主系统的工具。尽管如此，在链接时会使用刚刚安装的库。

在下一章，进入“chroot”环境后，就可以使用这些工具。但是在此之前，我们必须将本章中所有的软件包构建完毕。因此现在我们还不能脱离宿主系统。

再一次地，请注意如果 `LFS` 环境变量设置错误，而且使用 `root` 用户的身份进行构建，可能导致您的电脑完全无法使用。本章应该以用户 `lfs` 身份完成，且环境变量应该如同第 4.4 节 “配置环境”所述设置。

6.2. M4-1.4.20

M4 软件包包含一个宏处理器。

估计构建时间: 0.1 SBU
需要硬盘空间: 38 MB

6.2.1. 安装 M4

准备编译 M4:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.13.2 节 “M4 的内容”找到。

6.3. Ncurses-6.5-20250809

Ncurses 软件包包含使用时不需考虑终端特性的字符屏幕处理函数库。

估计构建时间: 0.4 SBU
需要硬盘空间: 54 MB

6.3.1. 安装 Ncurses

首先，运行下列命令，为宿主系统构建 **tic** 程序。我们将其安装到 `$LFS/tools` 中，这样需要该程序时，就能通过 `PATH` 环境变量找到它：

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

准备编译 Ncurses：

```
./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
```

新的配置选项的含义：

`--with-manpage-format=normal`
这防止 Ncurses 安装压缩的手册页面，否则在宿主发行版使用压缩的手册页面时，Ncurses 可能这样做。

`--with-shared`
该选项使得 Ncurses 将 C 函数库构建并安装为共享库。

`--without-normal`
该选项禁止将 C 函数库构建和安装为静态库。

`--without-debug`
该选项禁止构建和安装用于调试的库。

`--with-cxx-shared`
该选项使得 Ncurses 将 C++ 绑定构建并安装为共享库，同时防止构建和安装静态的 C++ 绑定库。

`--without-ada`
这保证不构建 Ncurses 的 Ada 编译器支持，宿主环境可能有 Ada 编译器，但进入 **chroot** 环境后 Ada 编译器就不再可用。

`--disable-stripping`
该选项防止构建过程使用宿主系统的 **strip** 程序。对交叉编译产生的程序使用宿主工具可能导致构建失败。

`AWK=gawk`
该选项防止构建过程使用宿主系统的 **mawk** 程序。一些版本的 **mawk** 会导致该软件包构建失败。

编译该软件包：

```
make
```

安装该软件包：

```
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
```

安装选项的含义：

ln -sv libncursesw.so $LFS/usr/lib/libncurses.so

我们很快将会构建一些需要 `libncurses.so` 库的软件包。创建该符号链接，使得这些包使用 `libncursesw.so` 作为替代。

sed -e 's/^#if.*XOPEN.*$/#if 1/' ...

头文件 `curses.h` 包含若干 Ncurses 数据结构的定义。在使用不同预处理器宏定义时，可能使用两套不同的数据结构定义：一套是用于 8 字节字符的定义，和 `libncurses.so` 兼容；而另一套是用于宽字符的定义，和 `libncursesw.so` 兼容。由于我们使用 `libncursesw.so` 替代 `libncurses.so`，修改这个头文件，使之总是使用与 `libncursesw.so` 兼容的宽字符数据结构定义。

该软件包的详细信息可以在第 8.30.2 节 “Ncurses 的内容”中找到。

6.4. Bash-5.3

Bash 软件包包含 Bourne-Again Shell。

估计构建时间: 0.2 SBU
需要硬盘空间: 72 MB

6.4.1. 安装 Bash

准备编译 Bash:

```
./configure --prefix=/usr                      \
            --build=$(sh support/config.guess) \
            --host=$LFS_TGT                    \
            --without-bash-malloc
```

配置选项的含义:

`--without-bash-malloc`
该选项禁用 Bash 自己的内存分配 (`malloc`) 函数，因为已知它会导致段错误。这样，Bash 就会使用 Glibc 的更加稳定的 `malloc` 函数。

编译该软件包:

```
make
```

安装该软件包:

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

为那些使用 **sh** 命令运行 shell 的程序考虑，创建一个链接:

```
ln -sv bash $LFS/bin/sh
```

该软件包的详细信息可以在第 8.36.2 节 “Bash 的内容”中找到。

6.5. Coreutils-9.7

Coreutils 软件包包含各种操作系统都需要提供的基本工具程序。

估计构建时间: 0.3 SBU
需要硬盘空间: 181 MB

6.5.1. 安装 Coreutils

准备编译 Coreutils:

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

配置选项的含义:

`--enable-install-program=hostname`
该选项表示构建 **hostname** 程序并安装它 —— 默认情况下它被禁用，但 Perl 测试套件需要它。

编译该软件包:

```
make
```

安装该软件包:

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

将程序移动到它们最终安装时的正确位置。在临时环境中这看似不必要，但一些程序会硬编码它们的位置，因此必须进行这步操作:

```
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
```

该软件包的详细信息可以在第 8.59.2 节 “Coreutils 的内容”中找到。

6.6. Diffutils-3.12

Diffutils 软件包包含显示文件或目录之间差异的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 35 MB

6.6.1. 安装 Diffutils

准备编译 Diffutils:

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

配置选项的含义:

`gl_cv_func_strcasecmp_works=y`
该选项为一项针对 `strcasecmp` 的检查指定结果。这项检查需要编译并运行一个 C 程序，而在交叉编译时这是不可行的，因为通常来说交叉编译得到的程序不能在宿主发行版运行。正常情况下，**configure** 脚本在交叉编译时应该对这些检查回落到一个缺省值，但这项检查缺失了缺省值设定，因此 **configure** 脚本完全无法得出检查结果，并报错退出。上游开发者已经修复了这一问题，但应用修复后需要运行 宿主发行版未必提供的 **autoconf**。因此我们不使用上游修复，而是直接指定检查结果 (为 `y`，因为我们已经知道Glibc-2.42 中的 `strcasecmp` 函数能够正常工作)，这样 **configure** 就会直接使用我们给定的值，跳过有问题的检查。

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.60.2 节 “Diffutils 的内容”中找到。

6.7. File-5.46

File 软件包包含用于确定给定文件类型的工具。

估计构建时间: 0.1 SBU
需要硬盘空间: 42 MB

6.7.1. 安装 File

宿主系统 **file** 命令的版本必须和正在构建的软件包相同，才能在构建过程中创建必要的特征数据文件。运行以下命令，构建 **file** 命令的一个临时副本：

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

新的配置选项的含义：

`--disable-*`
如果相关的库文件存在，配置脚本企图使用宿主发行版的一些软件包。当库文件存在，但对应的头文件不存在时，这会导致编译失败。该选项防止使用这些来自宿主系统的非必要功能。

准备编译 File：

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

编译该软件包：

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

安装该软件包：

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

移除对交叉编译有害的 libtool 档案文件：

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

该软件包的详细信息可以在第 8.11.2 节 “File 的内容”中找到。

6.8. Findutils-4.10.0

Findutils 软件包包含用于查找文件的程序。这些程序能直接搜索目录树中的所有文件，也可以创建、维护和搜索文件数据库 (一般比递归搜索快，但在数据库最近没有更新时不可靠)。Findutils 还提供了 **xargs** 程序，它能够对一次搜索列出的所有文件执行给定的命令。

估计构建时间: 0.2 SBU
需要硬盘空间: 48 MB

6.8.1. 安装 Findutils

准备编译 Findutils:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.62.2 节 “Findutils 的内容”中找到。

6.9. Gawk-5.3.2

Gawk 软件包包含操作文本文件的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 49 MB

6.9.1. 安装 Gawk

首先，确保不安装某些不需要的文件：

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

准备编译 Gawk：

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

编译该软件包：

```
make
```

安装该软件包：

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

该软件包的详细信息可以在第 8.61.2 节 “Gawk 的内容”中找到。

6.10. Grep-3.12

Grep 软件包包含在文件内容中进行搜索的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 32 MB

6.10.1. 安装 Grep

准备编译 Grep:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.35.2 节 “Grep 的内容”中找到。

6.11. Gzip-1.14

Gzip 软件包包含压缩和解压缩文件的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 12 MB

6.11.1. 安装 Gzip

准备编译 Gzip:

```
./configure --prefix=/usr --host=$LFS_TGT
```

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.65.2 节 “Gzip 的内容”中找到。

6.12. Make-4.4.1

Make 软件包包含一个程序，用于控制从软件包源代码生成可执行文件和其他非源代码文件的过程。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 15 MB

6.12.1. 安装 Make

准备编译 Make:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.69.2 节 “Make 的内容”中找到。

6.13. Patch-2.8

Patch 软件包包含通过应用 “补丁” 文件，修改或创建文件的程序，补丁文件通常是 **diff** 程序创建的。

估计构建时间: 0.1 SBU
需要硬盘空间: 14 MB

6.13.1. 安装 Patch

准备编译 Patch:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.70.2 节 “Patch 的内容”中找到。

6.14. Sed-4.9

Sed 软件包包含一个流编辑器。

估计构建时间: 0.1 SBU
需要硬盘空间: 21 MB

6.14.1. 安装 Sed

准备编译 Sed:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.31.2 节 “Sed 的内容”中找到。

6.15. Tar-1.35

Tar 软件包提供创建 tar 归档文件，以及对归档文件进行其他操作的功能。Tar 可以对已经创建的归档文件进行提取文件，存储新文件，更新文件，或者列出文件等操作。

估计构建时间: 0.1 SBU
需要硬盘空间: 42 MB

6.15.1. 安装 Tar

准备编译 Tar:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.71.2 节 “Tar 的内容”中找到。

6.16. Xz-5.8.1

Xz 软件包包含文件压缩和解压缩工具，它能够处理 lzma 和新的 xz 压缩文件格式。使用 **xz** 压缩文本文件，可以得到比传统的 **gzip** 或 **bzip2** 更好的压缩比。

估计构建时间: 0.1 SBU
需要硬盘空间: 23 MB

6.16.1. 安装 Xz

准备编译 Xz:

```
./configure --prefix=/usr                       \
            --host=$LFS_TGT                     \
            --build=$(build-aux/config.guess) \
            --disable-static                    \
            --docdir=/usr/share/doc/xz-5.8.1
```

编译该软件包:

```
make
```

安装该软件包:

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

移除对交叉编译有害的 libtool 档案文件:

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

该软件包的详细信息可以在第 8.8.2 节 “Xz 的内容”中找到。

6.17. Binutils-2.45 - 第二遍

Binutils 包含汇编器、链接器以及其他用于处理目标文件的工具。

估计构建时间: 0.4 SBU
需要硬盘空间: 548 MB

6.17.1. 安装 Binutils

Binutils 构建系统依赖附带的 libtool 拷贝链接内部静态库，但源码包内附带的 libiberty 和 zlib 不使用 libtool。这个区别可能导致构建得到的二进制程序和库错误地链接到宿主发行版的库。绕过这个问题：

```
sed '6031s/$add_dir//' -i ltmain.sh
```

再次创建一个独立的构建目录：

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

准备编译 Binutils：

```
../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
```

新的配置选项的含义：

`--enable-shared`
将 `libbfd` 构建为共享库。

`--enable-64-bit-bfd`
(在字长较小的宿主平台上) 启用 64 位支持。该选项在 64 位平台上可能不必要，但无害。

编译该软件包：

```
make
```

安装该软件包：

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

移除对交叉编译有害的 libtool 档案文件，同时移除不必要的静态库：

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

该软件包的更多信息可以在第 8.20.2 节 “Binutils 的内容”中找到。

6.18. GCC-15.2.0 - 第二遍

GCC 软件包包含 GNU 编译器集合，其中有 C 和 C++ 编译器。

估计构建时间: 4.5 SBU
需要硬盘空间: 6.0 GB

6.18.1. 安装 GCC

如同第一次构建 GCC 时一样，需要使用 GMP、MPFR 和 MPC 三个包。解压它们的源码包，并将它们移动到 GCC 要求的目录名:

```
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
```

如果正在 x86_64 上进行构建，修改存放 64 位库的默认路径为 “lib”:

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

覆盖 libgcc 和 libstdc++ 头文件的构建规则，以允许在构建它们时启用 POSIX 线程支持:

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

再次创建一个独立的构建目录:

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

在开始构建 GCC 前，记得清除所有覆盖默认优化开关的环境变量。

现在准备编译 GCC:

```
../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
```

新的配置选项的含义:

`--with-build-sysroot=$LFS`
通常，指定 `--host` 即可保证使用交叉编译器构建 GCC，这个交叉编译器知道它应该在 `$LFS` 中查找头文件和库。然而，GCC 构建系统还会使用一些不知道该位置的工具。因此需要该选项，才能使得这些工具在 `$LFS` 中查找需要的文件，而非在宿主系统中查找。

`--target=$LFS_TGT`
我们正在交叉编译 GCC，因此无法使用这一遍构建的 GCC 二进制程序 —— 它们无法在宿主系统运行 —— 为目标系统构建运行库 (`libgcc` 和 `libstdc++`)。GCC 构建系统在默认情况下会试图使用宿主系统提供的 C 和 C++ 编译器来绕过这个问题。但是，用不同版本的 GCC 构建 GCC 运行库不受支持，所以使用宿主系统的编译器可能导致构建失败。该选项保证使用第一遍构建的 GCC 编译运行库。

`LDFLAGS_FOR_TARGET=...`
允许 `libstdc++` 使用即将构建的 `libgcc`，而不是之前在第一遍的 GCC 中构建的版本。之前构建的版本无法正确支持 C++ 异常处理，因为它在构建时缺乏 libc 的支持。

`--disable-libsanitizer`
禁用 GCC 清理检查运行库。临时性安装 GCC 不需要它们。在第一遍的 GCC 中，`--disable-libstdcxx` 隐含了该选项，这里我们可以显式指定它。

编译该软件包：

```
make
```

安装该软件包：

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

最后，还需要创建一个符号链接。许多程序和脚本运行 **cc** 而不是 **gcc**，因为前者能够保证程序的通用性，使它可以在所有 UNIX 系统上使用，无论是否安装了 GNU C 编译器。运行 **cc** 可以将选择 C 编译器的权力留给系统管理员：

```
ln -sv gcc $LFS/usr/bin/cc
```

该软件包的详细信息在第 8.29.2 节 “GCC 的内容”可以找到。

第 7 章 进入 Chroot 并构建其他临时工具

7.1. 概述

本章展示如何构建临时系统最后缺失的部分：在构建一些软件包时必要的工具。由于已经解决了所有循环依赖问题，现在即可使用“chroot”环境进行构建，它与宿主系统 (除正在运行的内核外) 完全隔离。

为了隔离环境的正常工作，必须它与正在运行的内核之间建立一些通信机制。这些通信机制通过所谓的虚拟内核文件系统实现，我们将在进入 chroot 环境前挂载它们。您可能希望用 **findmnt** 命令检查它们是否挂载好。

从现在开始，直到第 7.4 节 “进入 Chroot 环境”，所有命令必须以 `root` 用户身份执行，且 `LFS` 变量必须正确设定。在进入 chroot 之后，仍然以 `root` 身份执行所有命令，但幸运的是此时无法访问您构建 LFS 的计算机的宿主系统。不过仍然要小心，因为错误的命令很容易摧毁整个 LFS 系统。

7.2. 改变所有者

注意

本书中后续的所有命令都应该在以 `root` 用户登录的情况下完成，而不是 `lfs` 用户。另外，请再次检查 `$LFS` 变量已经在 `root` 用户的环境中设定好。

目前，`$LFS` 中整个目录树的所有者都是 `lfs`，这个用户只在宿主系统存在。如果不改变 `$LFS` 中文件和目录的所有权，它们会被一个没有对应账户的用户 ID 所有。这是危险的，因为后续创建的新用户可能获得这个用户 ID，并成为 `$LFS` 中全部文件的所有者，从而产生恶意操作这些文件的可能。

为了避免这样的问题，执行以下命令，将 `$LFS/*` 目录的所有者改变为 `root`：

```
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. 准备虚拟内核文件系统

用户态程序使用内核创建的一些文件系统和内核通信。这些文件系统是虚拟的：它们并不占用磁盘空间。它们的内容保留在内存中。必须将它们被挂载到 $LFS 目录树中，这样 chroot 环境中的程序才能找到它们。

首先创建这些文件系统的挂载点：

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

7.3.1. 挂载和填充 /dev

在 LFS 系统的正常引导过程中，内核自动挂载 `devtmpfs` 到 `/dev`，并在引导过程中，或对应设备被首次发现或访问时动态地创建设备节点。udev 守护程序可能修改内核创建的设备节点的所有者或访问权限，或创建一些新的设备节点或符号链接，以简化发行版维护人员或系统管理员的工作。(详见第 9.3.2.2 节 “设备节点的创建”。) 如果宿主系统支持 `devtmpfs`，我们可以简单地将 `devtmpfs` 挂载到 `$LFS/dev` 并依靠内核填充其内容。

但是一些宿主系统的内核可能不支持 `devtmpfs`；这些宿主系统使用其他方法填充 `/dev`。因此，为了在任何宿主系统上都能填充 `$LFS/dev`，只能绑定挂载宿主系统的 `/dev` 目录。绑定挂载是一种特殊挂载类型，它允许通过不同的位置访问一个目录树或一个文件。运行以下命令进行绑定挂载：

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

7.3.2. 挂载虚拟内核文件系统

现在挂载其余的虚拟内核文件系统:

```
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 文件系统挂载选项的含义:

`gid=5`
该选项使得所有通过 devpts 文件系统创建的设备节点属于编号为 5 的组。我们将会为 `tty` 组分配这个编号。因为宿主系统可能为 `tty` 组分配不同的编号，此处指定组编号而不是组名。

`mode=0620`
该选项使得所有通过 devpts 创建的设备节点的权限模式为 0620 (所属用户可读写，所属组可写)。该选项和前一选项共同保证 devpts 创建的设备节点符合 grantpt() 函数的要求，这样就不需要 Glibc 的 **pt_chown** 辅助程序 (默认不会安装该程序)。

在某些宿主系统上，`/dev/shm` 是一个符号链接，通常指向 `/run/shm` 目录。我们已经在 /run 下挂载了 tmpfs 文件系统，因此在这里只需要创建一个访问权限符合要求的目录。

在其他宿主系统上，`/dev/shm` 是一个 tmpfs 的挂载点。此时，绑定挂载 /dev 只会在 chroot 环境中生成 /dev/shm 目录。这样，我们必须显式挂载一个 tmpfs:

```
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. 进入 Chroot 环境

现在已经准备好了所有继续构建其余工具时必要的软件包，可以进入 chroot 环境并完成临时工具的安装。在安装最终的系统时，会继续使用该 chroot 环境。以 `root` 用户身份，运行以下命令以进入当前只包含临时工具的 chroot 环境:

```
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
```

如果不希望使用所有可用的逻辑 CPU 核心，将 `$(nproc)` 替换为在本章和后续章节中构建软件包时所希望使用的核心数。第 8 章 中的一些软件包 (如 Autoconf, Libtool, 以及 Tar) 的测试套件不受 `MAKEFLAGS` 控制，它们使用另一个环境变量 `TESTSUITEFLAGS`。我们在此也设置这个变量，以便用多个 CPU 核心运行测试套件。

通过传递 `-i` 选项给 **env** 命令，可以清除 chroot 环境中的所有环境变量。随后，只重新设定 `HOME`，`TERM`，`PS1`，以及 `PATH` 变量。参数 `TERM=$TERM` 将 chroot 环境中的 `TERM` 变量设为和 chroot 环境外相同的值。一些程序需要这个变量才能正常工作，例如 **vim** 和 **less**。如果需要设定其他变量，例如 `CFLAGS` 或 `CXXFLAGS`，也可以在这里设定。

从现在开始，就不再需要使用 `LFS` 环境变量，因为所有工作都被局限在 LFS 文件系统内。这是由于 **chroot** 命令启动 Bash 时，已经将根目录 (`/`) 设置为 `$LFS`。

注意 `/tools/bin` 不在 `PATH` 中。这意味着不再使用交叉工具链。

另外，注意 **bash** 的提示符会包含 `I have no name!`。这是正常的，因为现在还没有创建 `/etc/passwd` 文件。

注意

本章剩余部分和后续各章中的命令都要在 chroot 环境中运行。如果您因为一些原因 (如重新启动计算机) 离开了该环境，必须确认虚拟内核文件系统如第 7.3.1 节 “挂载和填充 /dev”和第 7.3.2 节 “挂载虚拟内核文件系统”所述挂载好，然后重新进入 chroot 环境，才能继续安装 LFS。

7.5. 创建目录

现在可以在 LFS 文件系统中创建完整的目录结构。

注意

本节提到的一些目录已经在之前使用命令创建，或者在安装一些软件包时被创建。这里出于内容完整性的考虑，仍然给出它们。

首先，执行命令，创建一些位于根目录中的目录，它们不属于之前章节需要的有限目录结构：

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

执行以下命令，为这些直接位于根目录中的目录创建次级目录结构：

```
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
```

默认情况下，新创建的目录具有权限模式 755，但这并不适用于所有情况。在以上命令中，两个目录的访问权限被修改 —— 一个是 root 的主目录，另一个是包含临时文件的目录。

第一个修改能保证不是所有人都能进入 /root —— 一般用户也可以为他/她的主目录设置同样的 0750 权限模式。第二个修改保证任何用户都可写入 /tmp 和 /var/tmp 目录，但不能从中删除其他用户的文件，因为所谓的 “粘滞位” (sticky bit)，即八进制权限模式 1777 的最高位 (1) 阻止这样做。

7.5.1. FHS 兼容性注记

这个目录树是基于 Filesystem Hierarchy Standard (FHS) (可以在 https://refspecs.linuxfoundation.org/fhs.shtml 查阅) 建立的。FHS 标准还规定了某些可选的目录，例如 /usr/local/games 和 /usr/share/games。在 LFS 中，我们只创建必要的目录。不过，如果您需要的话可以自己创建这些可选目录。

警告

FHS 不要求 /usr/lib64目录，而且 LFS 编辑团队决定不使用它。LFS 和 BLFS 中的一些命令只有在该目录不存在时才能正常工作。您应该经常检查并确认该目录不存在，因为往往容易无意地创建该目录，而它的存在可能破坏您的系统。

7.6. 创建必要的文件和符号链接

历史上，Linux 曾在 /etc/mtab 维护已经挂载的文件系统的列表。现代内核在内部维护该列表，并通过 /proc 文件系统将它展示给用户。为了满足一些仍然使用 /etc/mtab 的工具，执行以下命令，创建符号链接：

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

创建一个基本的 /etc/hosts 文件，一些测试套件，以及 Perl 的一个配置文件将会使用它：

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

为了使得 root 能正常登录，而且用户名 “root” 能被正常识别，必须在文件 /etc/passwd 和 /etc/groups 中写入相关的条目。

执行以下命令创建 /etc/passwd 文件：

```
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
```

我们以后再设置 root 用户的实际密码。

执行以下命令，创建 /etc/group 文件：

```
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:
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
```

这里创建的用户组并不属于任何标准 —— 它们一部分是为了满足第 9 章中 Udev 配置的需要，另一部分借鉴了一些 Linux 发行版的通用惯例。另外，某些测试套件需要特定的用户或组。Linux Standard Base (LSB，可以在 https://refspecs.linuxfoundation.org/lsb.shtml 查看) 标准只推荐以组 ID 0 创建用户组 root，以及以组 ID 1 创建用户组 bin。组 ID 5 被几乎所有发行版分配给 tty 组，而且 systemd 为 devpts 文件系统直接指定了数值 5。其他组名和组 ID 由系统管理员自由分配，因为好的程序不会依赖组 ID 的数值，而是使用组名。

编号 65534 被内核用于 NFS 和用户命名空间，以表示未映射的用户或组 (它们存在于 NFS 服务器或上一级用户命名空间，但是在当前机器或命名空间中“不存在”)。我们为 nobody 和 nogroup 分配该编号，以避免出现未命名的编号。但是其他发行版可能用不同方式处理这个编号，因此需要移植的程序不能依赖于这里给出的分配方式。

第 8 章中的一些测试需要使用一个非特权用户。我们这里创建一个用户，在那一章的末尾再删除该用户。

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

为了移除 “I have no name!” 提示符，需要打开一个新 shell。由于已经创建了文件 /etc/passwd 和 /etc/group，用户名和组名现在就可以正常解析了：

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

login、**agetty** 和 **init** 等程序使用一些日志文件，以记录登录系统的用户和登录时间等信息。然而，这些程序不会创建不存在的日志文件。初始化日志文件，并为它们设置合适的访问权限：

```
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
```

文件 /var/log/wtmp 记录所有的登录和登出，文件 /var/log/lastlog 记录每个用户最后登录的时间，文件 /var/log/faillog 记录所有失败的登录尝试，文件 /var/log/btmp 记录所有错误的登录尝试。

注意

wtmp，btmp，以及 lastlog 文件使用 32 位整数作为时间戳，因此在 2038 年后它们将完全无法使用。许多软件包已经不再使用这些文件，且其他软件包也将停止使用它们。最好将它们视为已经弃用。

7.7. Gettext-0.26

Gettext 软件包包含国际化和本地化工具，它们允许程序在编译时加入 NLS (本地语言支持) 功能，使它们能够以用户的本地语言输出消息。

估计构建时间: 1.5 SBU
需要硬盘空间: 463 MB

7.7.1. 安装 Gettext

对于我们的临时工具，只要安装 Gettext 中的三个程序即可。

准备编译 Gettext:

```
./configure --disable-shared
```

配置选项的含义:

`--disable-shared`
现在我们不需要安装 Gettext 的任何共享库，因此不用构建它们。

编译该软件包:

```
make
```

安装 **msgfmt**，**msgmerge**，以及 **xgettext** 这三个程序:

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

该软件包的详细信息可以在第 8.33.2 节 “Gettext 的内容”中找到。

7.8. Bison-3.8.2

Bison 软件包包含语法分析器生成器。

估计构建时间: 0.2 SBU
需要硬盘空间: 58 MB

7.8.1. 安装 Bison

准备编译 Bison:

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

新的配置选项的含义:

`--docdir=/usr/share/doc/bison-3.8.2`
该选项告诉构建系统将 Bison 文档安装到带有版本号的目录中。

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.34.2 节 “Bison 的内容”中找到。

7.9. Perl-5.42.0

Perl 软件包包含实用报表提取语言。

估计构建时间: 0.6 SBU
需要硬盘空间: 295 MB

7.9.1. 安装 Perl

准备编译 Perl:

```
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
```

配置选项的含义:

`-des`
这是三个选项的组合：-d 对于所有配置项目使用默认值；-e 确保所有配置任务完成；-s 使得配置脚本不输出不必要的信息。

`-D vendorprefix=/usr`
这保证 **perl** 正确告知软件包安装其 Perl 模块的位置。

`-D useshrplib`
将一些 Perl 模块所需的 `libperl` 构建为共享库，而非静态库。

`-D privlib,-D archlib,-D sitelib,...`
这些选项定义 Perl 查找系统上安装的模块的位置。LFS 编辑决定将它们存放在以主版本号.次版本号 (如 5.42) 格式表示 Perl 版本的目录结构中，这样在升级 Perl 到更新的修订号 (修订号就是类似 5.42.0 这样的完整版本号中用小数点分割得到的最后一部分) 时，不需要重新安装所有模块。

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.43.2 节 “Perl 的内容”中找到。

7.10. Python-3.13.7

Python 3 软件包包含 Python 开发环境。它被用于面向对象编程，编写脚本，为大型程序建立原型，或者开发完整的应用。Python 是一种解释性的计算机语言。

估计构建时间: 0.5 SBU
需要硬盘空间: 546 MB

7.10.1. 安装 Python

注意

该软件包包含两个文件名以 “python” 为前缀的压缩包。我们应该解压的包是 `Python-3.13.7.tar.xz` (注意首字母是大写的)。

准备编译 Python:

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

配置选项的含义:

`--enable-shared`
该选项防止安装静态库。

`--without-ensurepip`
该选项禁止构建 Python 软件包安装器，它在当前阶段没有必要。

`--without-static-libpython`
该选项防止安装一个巨大且不必要的静态库。

编译该软件包:

```
make
```

注意

一些 Python 3 模块目前无法构建，这是因为它们的依赖项尚未安装。对于 `ssl` 模块，构建系统会输出一条消息 `Python requires a OpenSSL 1.1.1 or newer` (“Python 需要 OpenSSL 1.1.1 或更新版本”)。这条消息应该被忽略。只需要确认最外层的 **make** 命令执行成功即可。目前不需要这些可选的模块，它们将在第 8 章中被构建。

安装该软件包:

```
make install
```

关于该软件包的详细信息可以在第 8.51.2 节 “Python 3 的内容”中找到。

7.11. Texinfo-7.2

Texinfo 软件包包含阅读、编写和转换 info 页面的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 152 MB

7.11.1. 安装 Texinfo

准备编译 Texinfo:

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

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.72.2 节 “Texinfo 的内容”中找到。

7.12. Util-linux-2.41.1

Util-linux 软件包包含一些工具程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 192 MB

7.12.1. 安装 Util-linux

FHS 建议使用 `/var/lib/hwclock` 目录，而非一般的 `/etc` 目录作为 `adjtime` 文件的位置。首先创建该目录:

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

准备编译 Util-linux:

```
./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-2.41.1
```

配置选项的含义:

`ADJTIME_PATH=/var/lib/hwclock/adjtime`
该选项根据 FHS 的规则，设定硬件时钟信息记录文件的位置。对于临时工具，这并不是严格要求的，但是这样可以防止在其他位置创建该文件，导致这个文件在安装最终的 Util-linux 软件包时不被覆盖或移除。

`--libdir=/usr/lib`
该选项确保 `.so` 符号链接直接指向同一目录 (`/usr/lib`) 中的共享库文件。

`--disable-*`
这些选项防止产生关于一些组件的警告，这些组件需要一些 LFS 之外，或当前尚未安装的软件包。

`--without-python`
该选项禁用 Python，防止构建系统尝试构建不需要的语言绑定。

`runstatedir=/run`
该选项正确设定 **uuidd** 和 `libuuid` 使用的套接字的位置。

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.80.2 节 “Util-linux 的内容”中找到。

7.13. 清理和备份临时系统

7.13.1. 清理

首先，删除已经安装的临时工具文档文件，以防止它们进入最终构建的系统，并节省大约 35 MB：

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

其次，在现代 Linux 系统中，libtool 的 .la 文件仅用于 libltdl。LFS 中没有库通过 libltdl 加载，而且已知一些 .la 文件会导致 BLFS 软件包出现异常。现在删除这些文件：

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

当前临时系统使用约 3 GB 空间，但是我们已经不需要其中的 /tools 目录了。该目录使用约 1 GB 存储空间。现在删除它：

```
rm -rf /tools
```

7.13.2. 备份

现在，已经为系统安装了所有必要的程序和库，且 LFS 系统的当前状态良好。可以将系统备份起来，以便以后重新使用。如果在后续章节发生了无法挽回的错误，通常来说，最好的办法是删除所有东西，然后（更小心地）从头开始。不幸的是，这也会删除所有临时工具。为了避免浪费时间对已经构建成功的部分进行返工，可以准备一个备份。

注意

本节中的其余步骤都是可选的。不过，一旦您开始在第 8 章中安装软件包，临时工具就会被覆盖。因此，按照下面描述的步骤备份临时工具可能是个好主意。

以下步骤在 chroot 环境之外进行。这意味着您在进行它们之前必须离开 chroot 环境。这样做是为了访问 chroot 环境之外的文件系统位置，以写入或读取备份档案，备份档案不应存放在 $LFS 目录树中。

现在，如果您决定进行备份，离开 chroot 环境：

```
exit
```

重要

以下给出的所有步骤都在宿主系统中以 root 身份执行。请非常小心地执行命令，此处如果在命令中出现错误，则可能损坏您的宿主系统。特别注意环境变量 LFS 会自动为用户 lfs 设定，但可能没有为 root 设定。

无论何时，只要准备以 root 身份执行命令，一定要确认 LFS 变量已经正确设定。

第 2.6 节 “设置 $LFS 环境变量和 Umask”已经讨论了这个问题。

在进行备份之前，解除内核虚拟文件系统的挂载：

```
mountpoint -q $LFS/dev/shm && umount $LFS/dev/shm
umount $LFS/dev/pts
umount $LFS/{sys,proc,run,dev}
```

确认在 root 的主目录所在的文件系统中，有至少 1 GB 的可用存储空间 (源代码压缩包也会被包含在备份档案中)。

注意以下命令指定的是宿主系统中 root 用户的主目录，它通常在根文件系统中。如果不希望使用 root 的主目录，将 $HOME 替换成您选择的目录。

运行以下命令，创建备份档案：

注意

由于备份档案需要进行压缩，即使您的系统运行速度较快，该命令也会消耗较长的时间（可能超过 10 分钟)。

```
cd $LFS
tar -cJpf $HOME/lfs-temp-tools-12.4-systemd.tar.xz .
```

注意

正如下面的“重要”提示框所述，在继续进行第 8 章的操作之前，不要忘记重新进入 chroot 环境。

7.13.3. 还原

如果您犯下了一些错误，并不得不重新开始构建，您可以使用备份档案还原临时系统，节约一些工作时间。由于源代码在 `$LFS` 中，它们也包含在备份档案内，因此不需要重新下载它们。在确认 `$LFS` 设定正确后，可以运行以下命令从备份档案进行还原：

警告

下面的命令非常危险。如果您在没有切换到 $LFS 目录或 `LFS` 环境变量没有为 `root` 用户正确设定的情况下运行了 **rm -rf ./*** 命令，它会完全摧毁宿主系统。后果自负。

```
cd $LFS
rm -rf ./*
tar -xpf $HOME/lfs-temp-tools-12.4-systemd.tar.xz
```

再一次复查环境是否配置正确，即可继续构建系统。

重要

如果您在进行备份或从备份进行恢复时退出了 chroot 环境，记得检查内核虚拟文件系统是否仍然处于挂载状态（可以使用 **findmnt | grep $LFS** 进行检查)。如果它们尚未挂载，需要按照第 7.3 节 “准备虚拟内核文件系统”的描述重新挂载内核虚拟文件系统，并重新进入 chroot 环境 (参阅第 7.4 节 “进入 Chroot 环境”)，再继续进行构建。

第 IV 部分 构建 LFS 系统

第 8 章 安装基本系统软件

8.1. 概述

在本章中，我们将真正开始构造 LFS 系统。

软件的安装过程是简单直接的。尽管很多时候可以把安装说明写得更短、更通用，我们还是选择为每个包提供完整的安装流程，以尽量减小出错的可能。学习 Linux 系统工作原理的关键就是要知道每个包的作用，以及您 (或者系统) 为什么需要它。

我们不推荐在编译中使用自定义优化。自定义优化可以使程序跑得稍微快一点，但也可能在编译或运行的过程中带来问题。如果一个软件包在打开自定义优化时无法编译，试着关闭优化再编译它。即使一个软件包在使用自定义优化时可以编译，由于源代码和编译工具的复杂相互作用，仍然存在编译不正确的风险。另外请注意，除本书明确说明外，设定 `-march` 和 `-mtune` 是未经验证的。它们可能在工具链软件包 (Binutils、GCC 和 Glibc) 中引发问题。自定义编译优化带来的微小性能增益往往不值得冒上述风险。我们建议第一次构建 LFS 的读者不要使用自定义的优化选项。

另一方面，我们保持软件包默认配置启用的优化选项。另外，我们有时显式启用软件包提供但未作为默认的优化配置。软件包维护者已经测试了这些配置并认为它们是安全的，因此它们不太可能导致构建失败。通常来说，默认配置已经启用 `-O2` 或 `-O3`，因此在不使用任何自定义优化的情况下，得到的系统仍然会运行得很快，同时保持稳定。

在提供安装过程的说明之前，每个页面都提供了软件包的基本信息，包括其内容的简要描述，以及构建过程大概需要的时间和磁盘空间。在安装指令之后，有一个包含该软件包提供的所有程序和库的清单 (以及对它们的简要描述)。

注意

对于拥有可用的测试套件的软件包，第 8 章中给出的 SBU 值和需要的磁盘空间包含了运行测试套件需要的时间和磁盘空间。如无特殊说明，SBU 值根据仅使用四个 CPU 核心 (-j4) 进行操作时测得的时间计算。

8.1.1. 关于库

一般来说，LFS 作者不鼓励构建和安装静态库。在现代 Linux 系统中，多数静态库已经失去存在的意义。另外，将静态库链接到程序中可能是有害的。如果需要更新这个库以解决安全问题，所有使用该静态库的程序都要重新链接到新版本的库。程序对静态库的使用并不总是显然的，甚至可能无法查明有哪些程序需要重新链接 (以及如何重新链接)。

本章给出的安装过程删除或者禁止安装多数静态库。一般来说，传递 `--disable-static` 选项给 **configure** 即可达成目的。然而，某些情况下需要使用其他手段。在极个别情况下，特别是对于 Glibc 和 GCC，静态库在一般软件包的构建过程中仍然很关键，就不能禁用静态库。

关于库的更详细讨论，可以参阅 BLFS 手册中的 Libraries: Static or shared? 一节。

8.2. 软件包管理

经常有人请求将软件包管理加入 LFS 手册。包管理器跟踪文件的安装过程，简化移除或升级软件包的工作。一个好的包管理器还会特殊处理配置文件，以在重新安装或升级软件包时保留用户配置。在您开始想入非非前，不 —— 本节不会讨论或者推荐任何一个特定的包管理器。本节对软件包管理的流行技术及其工作原理进行综述。对您来说，完美的包管理器可能是其中的某个技术，也可能是几个技术的结合。本节还会简要介绍在升级软件包时可能遇到的问题。

LFS 或 BLFS 不介绍任何包管理器的原因包括：

· 处理软件包管理会偏离这两本手册的目标 —— 讲述如何构建 Linux 系统。

· 存在多种软件包管理的解决方案，它们各有优缺点。很难找到一种让所有读者满意的方案。

已经有人写了一些关于软件包管理这一主题的短文。您可以访问 Hints Project 并看一看是否有符合您的需求的方案。

8.2.1. 升级问题

使用包管理器可以在软件包新版本发布后容易地完成升级。一般来说，使用 LFS 或者 BLFS 手册给出的构建方法即可升级软件包。下面是您在升级时必须注意的重点，特别是升级正在运行的系统时。

· 如果需要升级 Linux 内核 (例如，从 5.10.17 升级到 5.10.18 或 5.11.1)，则不需要重新构建其他任何软件包。因为内核态与用户态的接口十分清晰，系统仍然能够继续正常工作。特别地，在升级内核时，不需要一同更新 Linux API 头文件。重新引导系统即可使用升级后的内核。

· 如果需要将 Glibc 升级到一个新版本 (例如，从 Glibc-2.36 升级到 Glibc-2.42)，则需要进行一些额外操作，以防止损坏系统。详见第 8.5 节 “Glibc-2.42”。[1]

· 如果更新了一个包含共享库的软件包，而且共享库的名称 发生改变，那么所有动态链接到这个库的软件包都需要重新编译，以链接到新版本的库。例如，考虑一个软件包 foo-1.2.3 安装了名为 `libfoo.so.1` 的共享库，如果您把该软件包升级到了新版本 foo-1.2.4，它安装了名为 `libfoo.so.2` 的共享库。那么，所有链接到 `libfoo.so.1` 的软件包都要重新编译以链接到 `libfoo.so.2`。注意，您不能删除旧版本的库，直到将所有依赖它的软件包都重新编译完成。

· 如果一个软件包 (直接或间接地) 链接到同一共享库的旧名称和新名称 (例如，同时链接到 `libfoo.so.2` 和 `libbar.so.1`，而后者又链接到 `libfoo.so.3`)，这个软件包可能无法正常工作，这是由于不同版本的共享库可能对同一符号名提供不兼容的定义。这种情况可能由于在更新提供共享库的软件包后，重新编译一部分 (而非所有) 链接到旧的共享库的软件包而出现。为了避免这种问题，用户需要在共享库名称被更新时，尽快重新构建所有链接到该共享库的软件包。

· 如果更新了一个包含共享库的软件包，且共享库的名称没有改变，但是库**文件**的版本号降低了 (例如，库的名称保持 `libfoo.so.1` 不变，但是库文件名由 `libfoo.so.1.25` 变为 `libfoo.so.1.24`)，则需要删除旧版本软件包安装的库文件 (对于上述示例，需要删除 `libfoo.so.1.25`)。否则，**ldconfig** 命令 (可能是您通过命令行执行，也可能由一些软件包的安装过程自动执行) 会将符号链接 `libfoo.so.1` 的目标重设为旧版本的库文件，因为它版本号更大，看上去更“新”。在不得不降级软件包，或者软件包作者更改库文件版本号格式时，可能出现这种问题。

· 如果更新了一个包含共享库的软件包，且共享库的名称没有改变，但是这次更新修复了一个严重问题 (特别是安全缺陷)，则要重新启动所有链接到该库的程序。在更新软件包的过程完成后，以 `root` 身份，运行以下命令，即可列出所有正在使用旧版本共享库的进程 (将 `libfoo` 替换成库名)：

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

如果正在使用 OpenSSH 访问系统，且它链接到了被更新的库，则需要重启 **sshd** 服务，登出并重新登录，然后再次运行上述命令，确认没有进程使用被删除的库文件。

如果 **systemd** 守护进程 (以 PID 1 运行) 链接到了被更新的库，可以在不重启系统的前提下重启它：以 `root` 用户身份，运行 **systemctl daemon-reexec** 命令即可。

· 如果一个可执行程序或共享库被覆盖，正在使用该程序或库中的代码或数据的进程可能崩溃。正确的，不会导致进程崩溃的更新程序或共享库的方法是：先删除旧版本，再安装新版本。Coreutils 提供的 **install** 已经实现了这一过程，多数软件包使用该命令安装二进制文件和库。这意味着在更新软件包时通常不会遇到这个问题。然而，一些软件包 (如 BLFS 中的 SpiderMonkey) 的安装过程会简单覆盖已经存在的文件并导致进程崩溃，因此在进行更新前，最好保存工作并关闭不需要的，正在运行的进程。

[1] 共享库的名称是其 ELF dynamic 节中 `DT_SONAME` 条目编码的字符串。您可以通过命令 **readelf -d <library file> | grep SONAME** 获取它。多数情况下，它以 `.so.<a version number>` 结尾，但也有包含多个版本号 (如`libbz2.so.1.0`)，版本号在 `.so` 后缀之前 (如 `libbfd-2.45`)，或者根本没有版本号 (如 `libmemusage.so`) 的情况。软件包的版本和共享库名称中的版本号没有必然联系。

8.2.2. 软件包管理技术

以下是几种常见的软件包管理方案。在决定使用某种包管理器前，请研读这些方案，特别是要了解每种方案的不足。

8.2.2.1. 这都在我的脑袋里!

没错，这是一种包管理技术。有些人不需要包管理器，因为他们十分了解软件包，知道每个软件包安装了什么文件。有的用户则计划每次有软件包发生变动时就重新构建系统，所以不需要管理软件包。

8.2.2.2. 安装到独立目录

这是一种最简单的软件包管理方式，它不需要控制软件包安装的专用程序。每个软件包都被安装在单独的目录中。例如，软件包 foo-1.1 将会被安装在 `/opt/foo-1.1`，然后创建一个符号链接 `/opt/foo` 指向 `/opt/foo-1.1`。在安装新版本 foo-1.2 的时候，把它安装到 `/opt/foo-1.2`，然后把之前的符号链接替换为指向新版本的符号链接。

`PATH`、`MANPATH`、`INFOPATH`、`PKG_CONFIG_PATH`、`CPPFLAGS`、`LDFLAGS` 等环境变量，以及配置文件 `/etc/ld.so.conf` 可能需要被扩充，以包含 `/opt/foo-x.y`。

BLFS 手册使用这种模式安装一些非常庞大的软件包，这样能更容易地更新它们。一旦用这种方式安装了比较多的软件包，这种模式就会变得难以控制。另外，一些软件包 (如 Linux API 头文件和 Glibc) 在使用这种方式安装时可能无法正常工作。**永远不要对整个系统的所有软件包使用这种模式**。

8.2.2.3. 符号链接风格的软件包管理

这是前一种软件包管理技术的变种。和前一种方式一样，将各个软件包同样安装在独立的目录中。但不是使用软件包的名称建立符号链接，而是将软件包中的每个文件符号链接到 `/usr` 目录树中对应的位置。这样就不需要修改环境变量。虽然这些符号链接可以由用户自己创建，但已经有许多包管理器能够自动化这一过程。一些流行的包管理器如 Stow、Epkg、Graft 和 Depot 使用这种管理方式。

需要欺骗安装脚本，使得软件包认为它处于 `/usr` 中，尽管它实际上被安装在 `/usr/pkg` 目录结构中。这种安装过程往往是超出常规的。例如，考虑安装软件包 libfoo-1.1。下面的方法可能不能正确安装该软件包：

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

尽管安装过程本身可以顺利进行，但依赖于它的软件包可能不会像您期望的那样链接 libfoo 库。如果要编译一个依赖于 libfoo 的软件包，您可能发现它链接到了 `/usr/pkg/libfoo/1.1/lib/libfoo.so.1` 而不是您期望的 `/usr/lib/libfoo.so.1`。正确的做法是使用 `DESTDIR` 环境变量转移安装位置。就像下面这样做：

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

多数软件包可以这样安装，但有些不能。对于那些不兼容的软件包，您要么亲自动手安装，要么更简单地把一些出问题的软件包安装在 `/opt` 中。

8.2.2.4. 基于时间戳的方案

在这种方案中，安装一个软件包之前，创建一个时间戳文件。在安装后，用一行简单的 **find** 命令，加上正确的参数，就能生成安装日志，包含在时间戳文件创建以后安装的所有文件。有一个采用这个方案的包管理器叫做 install-log。

尽管这种方式很简单，但它有两个缺点。如果在安装过程中，某些文件没有以当前时间作为时间戳安装，它们就不能被包管理器跟踪。另外，只有每次只安装一个软件包时才能使用这种技术。如果在两个控制台中同时安装两个不同的软件包，它们的安装日志就不可靠了。

8.2.2.5. 追踪安装脚本

在这种方案中，安装脚本执行的命令被记录下来。有两种技巧可以用于记录：

在安装前设置 `LD_PRELOAD` 环境变量，将其指向一个库以在安装过程中预加载它。在安装过程中，这个库附加在 **cp**、**install**、**mv** 等可执行文件上，跟踪修改文件系统的系统调用。如果要使用这种方法，所有需要跟踪的可执行文件必须是动态链接的，且没有设定 suid 和 sgid 位。预加载动态库可能在安装过程中导致不希望的副作用。因此，最好在实际使用前进行一些测试，以确保包管理器不会造成破坏，并且记录了所有应该记录的文件。

第二种技巧是使用 **strace**，它能够记录安装脚本执行过程中的所有系统调用。

8.2.2.6. 创建软件包档案

在这种方案中，软件包被伪装安装到一个独立的目录树中，就像软链接风格的软件包管理那样。在安装后，使用被安装的文件创建一个软件包档案。它可以被用来在本地机器甚至其他机器上安装该软件包。

大多数商业发行版的包管理器采用这种策略。例如 RPM (值得一提的是，它被 Linux Standard Base 规则所要求)、pkg-utils、Debian 的 apt，以及 Gentoo 的 Portage 系统等。LFS Hint 中的一篇短文描述了如何为 LFS 系统适用这种管理方式：https://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt。

创建包含依赖关系信息的软件包文件十分复杂，超出了 LFS 的范畴。

Slackware 使用一个基于 **tar** 的系统创建软件包档案。和更复杂的包管理器不同，该系统有意地没有涉及软件包依赖关系。如果想了解 Slackware 包管理器的详细信息，阅读 https://www.slackbook.org/html/package-management.html。

8.2.2.7. 基于用户的软件包管理

这种架构是 LFS 特有的，由 Matthias Benkmann 提出，可以在 Hints Project 查阅。在该架构中，每个软件包都由一个单独的用户安装到标准位置。只要检查文件所有者，就能找出属于一个软件包的所有文件。它的优缺点十分复杂，无法在本节讨论。如果想详细了解，请访问 https://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt 阅读。

8.2.3. 在多个系统上部署 LFS

LFS 系统的一项优势是，没有依赖于磁盘系统中文件位置的文件。将构建好的 LFS 系统复制到另一台具有相同硬件架构的计算机很简单，只要用 **tar** 命令把包含根目录的 LFS 分区打包 (未压缩的情况下，一个基本的 LFS 系统需要 900 MB)，然后通过网络或者 CD-ROM 复制到新的系统上，再展开即可。之后，个别配置文件需要修改。可能需要更新的配置文件有：`/etc/hosts`，`/etc/fstab`，`/etc/passwd`，`/etc/group`，`/etc/shadow`，以及 `/etc/ld.so.conf`。

由于系统硬件和内核配置的区别，可能需要为新系统重新配置并构建内核。

重要

如果计划将 LFS 系统部署到使用不同型号 CPU 的硬件上，在构建第 8.21 节“GMP-6.3.0”和第 8.50 节“Libffi-3.5.2”时必须按照为它们提供的注意事项，覆盖特定于架构的优化，以得到既能在当前硬件运行，又能在计划部署 LFS 系统的硬件运行的库。否则在运行 LFS 时，会出现 `Illegal Instruction` (非法指令) 错误。

最后，按照第 10.4 节 “使用 GRUB 设定引导过程”中的说明，为新系统配置引导加载器。

8.3. Man-pages-6.15

Man-pages 软件包包含 2,400 多个手册页。

估计构建时间: 0.1 SBU
需要硬盘空间: 52 MB

8.3.1. 安装 Man-pages

移除描述密码散列函数的两个手册页。Libxcrypt 会提供这些手册页的更好版本。

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

执行以下命令安装 Man-pages:

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

各选项的含义:

`-R`
该选项阻止 **make** 设置内建变量。内建变量会干扰 man-pages 构建系统的正常工作，但目前除了直接通过命令行传递 `-R` 选项外，没有其他禁用内建变量的方法。

`GIT=false`
该选项防止构建系统输出许多行 `git: command not found` (git: 未找到命令) 警告消息。

8.3.2. Man-pages 的内容

安装的文件: 若干手册页

简要描述

手册页	描述 C 语言函数、重要的设备文件以及主要配置文件

8.4. Iana-Etc-20250807

Iana-Etc 软件包包含网络服务和协议的数据。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.8 MB

8.4.1. 安装 Iana-Etc

对于该软件包，我们只需要将文件复制到正确的位置：

```
cp services protocols /etc
```

8.4.2. Iana-Etc 的内容

安装的文件: /etc/protocols 和 /etc/services

简要描述

/etc/protocols	描述 TCP/IP 子系统中可用的各种 DARPA Internet 协议
/etc/services	提供 Internet 服务的可读文本名称、底层的分配端口号以及 协议类型之间的对应关系

8.5. Glibc-2.42

Glibc 软件包包含主要的 C 语言库。它提供用于分配内存、检索目录、打开和关闭文件、读写文件、字符串处理、模式匹配、算术等用途的基本子程序。

估计构建时间: 12 SBU
需要硬盘空间: 3.3 GB

8.5.1. 安装 Glibc

一些 Glibc 程序使用与 FHS 不兼容的 `/var/db` 目录存放它们的运行时数据。应用一个补丁，使得这些程序在 FHS 兼容的位置存放运行时数据：

```
patch -Np1 -i ../glibc-2.42-fhs-1.patch
```

现在修复一项可能导致 BLFS 中的 Valgrind 出现故障的问题：

```
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 文档推荐在一个新建的目录中构建 Glibc：

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

确保将 **ldconfig** 和 **sln** 工具安装到 `/usr/sbin` 目录中：

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

准备编译 Glibc：

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

配置选项的含义：

`--disable-werror`
该选项禁用 GCC 的 -Werror 选项。这对于运行测试套件来说是必须的。

`--enable-kernel=5.4`
该选项告诉构建系统 Glibc 可能被与 5.4 这样老版本的内核一起使用。这样，Glibc 会生成代码，在后续版本引入的系统调用不可用时绕过它们。

`--enable-stack-protector=strong`
该选项加入额外的缓冲区溢出检查代码以防范栈溢出攻击。注意 Glibc 总是显式指定栈防护设置并覆盖 GCC 的默认值，因此尽管已经为 GCC 指定了 `--enable-default-ssp` 选项，仍然需要使用该选项。

`--disable-nscd`
不构建目前已经没有作用的命名服务缓存守护程序。

`libc_cv_slibdir=/usr/lib`
这个变量纠正库文件安装位置。我们不希望使用 lib64 目录。

编译该软件包：

```
make
```

重要

我们认为，在本节中，Glibc 的测试套件十分关键。在任何情况下都不要跳过它。

通常来说，可能会有极少数测试不能通过，下面列出的失败结果一般可以安全地忽略。执行以下命令进行测试：

```
make check
```

您可能看到一些失败结果。Glibc 的测试套件和宿主系统之间有某种依赖关系。在 6000 多项测试中，如果只有几项测试失败，一般可以忽略它们。下面列出在一些版本的 LFS 上发现的，最常见的问题：

- 已知 io/tst-lchmod 在 LFS chroot 环境中会失败。
- 已知 misc/tst-preadvwritev2 和 misc/tst-preadvwritev64v2 在宿主内核为 Linux-6.14 或更高版本时可能失败。
- 一些测试，例如 nss/tst-nss-files-hosts-multi 和 nptl/tst-thread-affinity* 会由于超时而失败 (特别是在系统较慢和/或使用多个并行 make 任务进行测试时)。可以使用下列命令识别因为该原因失败的测试：

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

  可以执行 **TIMEOUTFACTOR=<倍率> make test t=<测试名>** 重新运行单项测试并放宽其运行时间限制。例如，**TIMEOUTFACTOR=10 make test t=nss/tst-nss-files-hosts-multi** 会以十倍于默认的最长运行时间重新运行 nss/tst-nss-files-hosts-multi。
- 另外，一些测试在 CPU 型号 (如 elf/tst-cpu-features-cpuinfo) 或宿主系统内核版本 (如 stdlib/tst-arc4random-thread) 较为老旧时可能失败。

在安装 Glibc 时，它会抱怨文件 `/etc/ld.so.conf` 不存在。尽管这是一条无害的消息，执行以下命令即可防止这个警告：

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

修改 Makefile，跳过一个过时的，对于现代的 Glibc 构型会失败的完整性检查：

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

重要

如果在正在运行的 LFS 系统上升级 Glibc 到一个新的次版本号 (例如，从 Glibc-2.36 升级到 Glibc-2.42)，则需要额外注意一些事项，以防止损坏系统：

- 我们不支持在 11.0 (不含) 之前版本的 LFS 系统上升级 Glibc。如果您正在运行这样古老的 LFS 系统，但需要更新 Glibc，则需要重新构建 LFS。
- 如果在 12.0 (不含) 之前版本的 LFS 系统上进行升级，需要按照第 8.27 节 “Libxcrypt-4.4.38” 安装 Libxcrypt。除了正常安装 Libxcrypt 外，**必须按照 Libxcrypt 一节中的“注意”部分，安装 `libcrypt.so.1*` (将之前 Glibc 安装的 `libcrypt.so.1` 替换)**。
- 如果在 12.1 (不含) 之前的 LFS 系统上进行升级，删除 **nscd** 程序：

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

  如果正在升级的 LFS 系统 (12.1 之前，不含 12.1) 使用 Systemd，则同时需要禁用并停止 **nscd** 服务：

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

- 如果内核版本早于 5.4 (使用 **uname -r** 查看当前版本) 或者您希望升级内核，按照第 10.3 节 “Linux-6.16.1” 升级内核并重启。
- 如果内核 API 头文件版本早于 5.4 (使用 **cat /usr/include/linux/version.h** 查看当前版本) 或者您希望升级内核头文件，按照第 5.4 节 “Linux-6.16.1 API 头文件” (但是需要从命令中删去 `$LFS`) 进行升级。
- 使用 `DESTDIR` 进行安装，并使用一条 **install** 命令一次性升级系统上安装的的所有 Glibc 共享库：

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

必须严格执行上述步骤，除非您完全理解您在做什么。**有意或无意地偏离上述步骤都可能导致您的系统完全无法使用。后果自负。**

之后继续运行 **make install** 命令，针对 `/usr/bin/ldd` 的 **sed** 命令，以及安装 locale 的命令。完成这些命令后，立刻重启系统。

如果正在运行版本 12.0 之前 (不含) 的 LFS 系统，由于 GCC 构建时未使用 `--disable-fixincludes` 选项，在重启成功后，需要将两个 GCC 头文件移动到更好的位置，并删除遗留的“修复过的” Glibc 头文件副本：

```
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
```

安装该软件包：

```
make install
```

改正 **ldd** 脚本中硬编码的可执行文件加载器路径：

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

下面，安装一些 locale，它们可以使得系统用不同语言响应用户请求。这些 locale 都不是必须的，但是如果缺少了它们中的某些，在运行一些软件包的测试套件时，可能跳过重要的测试。

可以用 **localedef** 程序安装单独的 locale。例如，下面的第二个 **localedef** 命令将 `/usr/share/i18n/locales/cs_CZ` 中的字符集无关 locale 定义和 `/usr/share/i18n/charmaps/UTF-8.gz` 中的字符映射定义组合起来，并附加到 `/usr/lib/locale/locale-archive` 文件。以下命令将会安装能够覆盖测试所需的最小 locale 集合:

```
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
```

另外，安装适合您自己国家、语言和字符集的 locale。

或者，也可以执行这个需要很长时间的命令，直接安装 `glibc-2.42/localedata/SUPPORTED` 中列出的所有 locale (包括上面列出的所有 locale，以及其他很多):

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

注意

目前 glibc 在解析国际化域名时使用 libidn2。这形成了一个运行时依赖关系。如果需要使用解析国际化域名的功能，参阅 BLFS libidn2 页面安装 libidn2。

8.5.2. 配置 Glibc

8.5.2.1. 创建 nsswitch.conf

由于 Glibc 的默认值在网络环境下不能很好地工作，需要创建配置文件 `/etc/nsswitch.conf`。

执行以下命令创建新的 /etc/nsswitch.conf：

```
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. 添加时区数据

输入以下命令，安装并设置时区数据：

```
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 命令的含义：

zic -L /dev/null ...
: 该命令创建没有闰秒的 POSIX 时区。一般的惯例是将它们安装在 zoneinfo 和 zoneinfo/posix 两个目录中。必须将 POSIX 时区安装到 zoneinfo，否则若干测试套件会报告错误。在嵌入式系统上，如果存储空间十分紧张，而且您永远不会更新时区信息，您可以不使用 posix 目录，以节约 1.9 MB，但个别程序或测试套件可能会失败。

zic -L leapseconds ...
: 该命令创建正确的，包含闰秒的时区。在嵌入式系统上，如果存储空间十分紧张，而且您永远不会更新时区信息，也不关心系统时间是否正确，您可以跳过 right 目录，以节约 1.9 MB。

zic ... -p ...
: 该命令创建 posixrule 文件。我们使用纽约时区，因为 POSIX 要求与美国一致的夏令时规则。

一种确定本地时区的方法是运行脚本：

```
tzselect
```

在回答关于当前位置的若干问题后，脚本会输出对应时区的名字（例如 America/Edmonton）。在 /usr/share/zoneinfo 中还有一些该脚本不能识别，但可以使用的时区，如 Canada/Eastern 或者 EST5EDT。

确定时区后，执行以下命令，创建 /etc/localtime：

```
ln -sfv /usr/share/zoneinfo/<xxx> /etc/localtime
```

将 <xxx> 替换成选定时区的名称 (例如 Canada/Eastern)。

8.5.2.3. 配置动态加载器

默认情况下，动态加载器 (/lib/ld-linux.so.2) 在 /usr/lib 中搜索程序运行时需要的动态库。然而，如果在除了 /usr/lib 以外的其他目录中有动态库，为了使动态加载器能够找到它们，需要把这些目录添加到文件 /etc/ld.so.conf 中。有两个目录 /usr/local/lib 和 /opt/lib 经常包含附加的共享库，所以现在将它们添加到动态加载器的搜索目录中。

运行以下命令，创建一个新的 /etc/ld.so.conf：

```
cat > /etc/ld.so.conf << "EOF"
# Begin /etc/ld.so.conf
/usr/local/lib
/opt/lib

EOF
```

如果希望的话，动态加载器也可以搜索一个目录，并将其中的文件包含在 ld.so.conf 中。通常包含文件目录中的文件只有一行，指定一个期望的库文件目录。如果需要这项功能，执行以下命令：

```
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 的内容

安装的程序: gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, ld.so (到 ld-linux-x86-64.so.2 或 ld-linux.so.2 的符号链接), locale, localedef, makedb, mtrace, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, 以及 zic

安装的库: 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, 以及 libutil.{a,so.1}

安装的目录: /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, 以及 /var/lib/nss_db

简要描述

gencat	生成消息目录
getconf	显示文件系统指定的系统配置变量值
getent	从管理数据库取得条目
iconv	转换给定文件的字符集
iconvconfig	创建可快速加载的 **iconv** 模块配置文件

ldconfig	配置动态链接器运行时绑定
ldd	报告给定程序或共享库依赖于哪些共享库
lddlibc4	辅助 **ldd** 处理目标文件。它在 x86_64 等较新的架构上不存在
locale	给出当前 locale 的一些信息
localedef	编译 locale 规范
makedb	从文本输入创建简单的数据库
mtrace	读取并解析内存跟踪文件，以人类可读的形式输出内存跟踪信息
pcprofiledump	显示基于程序计数器的性能剖析数据
pldd	列出正在运行的进程使用的共享库
sln	静态链接的 **ln** 程序
sotruss	跟踪特定命令对共享库中子程序的调用
sprof	读取并显示共享库性能剖析数据
tzselect	询问用户系统所在的位置并报告对应的时区
xtrace	显示正在执行的函数以跟踪程序执行
zdump	输出当前时间在多个时区中的表示
zic	时区编译器
`ld-*.so`	动态链接器/加载器
`libBrokenLocale`	被 Glibc 内部用作使某些不正确的程序 (例如某些 Motif 程序) 正常运行的粗糙手段，参阅 `glibc-2.42/locale/broken_cur_max.c` 中的注释了解更多信息
`libanl`	没有功能的空库。曾经是异步命名查找库，但其功能已经并入 `libc`
`libc`	主要的 C 运行库
`libc_malloc_debug`	预加载该库时启用内存分配检查
`libdl`	没有功能的空库。曾经是动态链接接口库，但其功能已经并入 `libc`
`libg`	没有功能的空库，曾经是 **g++** 的运行库
`libm`	数学库
`libmvec`	向量数学库，在使用 `libm` 时自动按需链接。
`libmcheck`	链接到该库时启用内存分配检查
`libmemusage`	被 **memusage** 用于收集程序内存使用信息
`libnsl`	网络服务库，已经弃用
`libnss_*`	命名服务开关模块，包含用于解析域名、用户名、组名、代号、服务、协议等的函数。由 `libc` 根据 `/etc/nsswitch.conf` 的配置进行加载。
`libpcprofile`	可以预加载它，以对程序进行基于程序计数器的性能剖析
`libpthread`	没有功能的空库。曾经包含 POSIX.1c 线程扩展要求的多数接口函数和 POSIX.1b 实时扩展要求的信号量接口函数，但这些函数现已并入 `libc`
`libresolv`	包含用于创建、发送和解析因特网域名服务数据包的函数
`librt`	包含 POSIX.1b 实时扩展要求的多数接口
`libthread_db`	包含用于构建多线程程序调试器的函数
`libutil`	没有功能的空库。曾经包含一些 Unix 工具使用的“标准”函数。这些函数已经并入 `libc`

8.6. Zlib-1.3.1

Zlib 软件包包含一些程序使用的压缩和解压缩子程序。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 6.4 MB

8.6.1. 安装 Zlib

准备编译 Zlib:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

删除无用的静态库:

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

8.6.2. Zlib 的内容

安装的库: libz.so

简要描述

`libz` 包含一些程序使用的压缩和解压缩函数

8.7. Bzip2-1.0.8

Bzip2 软件包包含用于压缩和解压缩文件的程序。使用 **bzip2** 压缩文本文件可以获得比传统的 **gzip** 优秀许多的压缩比。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 7.3 MB

8.7.1. 安装 Bzip2

应用一个补丁，以安装该软件包的文档:

```
patch -Np1 -i ../bzip2-1.0.8-install_docs-1.patch
```

以下命令保证安装的符号链接是相对的:

```
sed -i 's@\(ln -s -f \)$(PREFIX)/bin/@\1@' Makefile
```

确保手册页被安装到正确位置:

```
sed -i "s@(PREFIX)/man@(PREFIX)/share/man@g" Makefile
```

执行以下命令，准备编译 Bzip2:

```
make -f Makefile-libbz2_so
make clean
```

make 命令参数的含义:

`-f Makefile-libbz2_so`
该选项使用一个不同的 `Makefile` 文件构建 Bzip2，对于我们的例子来说就是使用 `Makefile-libbz2_so` 文件。它创建一个共享库 `libbz2.so`，并将 Bzip2 工具链接到这个库。

编译并测试该软件包:

```
make
```

安装软件包中的程序:

```
make PREFIX=/usr install
```

安装共享库:

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

安装链接到共享库的 **bzip2** 二进制程序到 `/bin` 目录，并将两个和 **bzip2** 完全相同的文件替换成符号链接:

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

删除无用的静态库:

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

8.7.2. Bzip2 的内容

安装的程序: bunzip2 (链接到 bzip2), bzcat (链接到 bzip2), bzcmp (链接到 bzdiff), bzdiff, bzegrep (链接到 bzgrep), bzfgrep (链接到 bzgrep), bzgrep, bzip2, bzip2recover, bzless (链接到bzmore), 以及 bzmore
安装的库: libbz2.so
安装的目录: /usr/share/doc/bzip2-1.0.8

简要描述

bunzip2	解压缩 bzip 压缩文件
bzcat	解压到标准输出
bzcmp	对 bzip 压缩过的文件运行 **cmp**
bzdiff	对 bzip 压缩过的文件运行 **diff**
bzegrep	对 bzip 压缩过的文件运行 **egrep** 命令
bzfgrep	对 bzip 压缩过的文件运行 **fgrep** 命令
bzgrep	对 bzip 压缩过的文件运行 **grep** 命令
bzip2	使用 Burrows-Wheeler 块排序文本压缩算法和 Huffman 编码压缩文件；其压缩率优于更常见的使用 “Lempel-Ziv” 算法的压缩工具，如 **gzip**
bzip2recover	试图从损坏的 bzip2 压缩文件中恢复数据
bzless	对 bzip 压缩过的文件运行 **less** 命令
bzmore	对 bzip 压缩过的文件运行 **more** 命令
`libbz2`	这个库实现基于 Burrows-Wheeler 算法的无损块排序数据压缩

8.8. Xz-5.8.1

Xz 软件包包含文件压缩和解压缩工具，它能够处理 lzma 和新的 xz 压缩文件格式。使用 **xz** 压缩文本文件，可以得到比传统的 **gzip** 或 **bzip2** 更好的压缩比。

估计构建时间: 0.1 SBU
需要硬盘空间: 24 MB

8.8.1. 安装 Xz

准备编译 Xz:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/xz-5.8.1
```

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.8.2. Xz 的内容

安装的程序: lzcat (到 xz 的链接), lzcmp (到 xzdiff 的链接), lzdiff (到 xzdiff 的链接), lzegrep (到 xzgrep 的链接), lzfgrep (到 xzgrep 的链接), lzgrep (到 xzgrep 的链接), lzless (到 xzless 的链接), lzma (到 xz 的链接), lzmadec, lzmainfo, lzmore (到 xzmore 的链接), unlzma (到 xz 的链接), unxz (到 xz 的链接), xz, xzcat (到 xz 的链接), xzcmp (到 xzdiff 的链接), xzdec, xzdiff, xzegrep (到 xzgrep 的链接), xzfgrep (到 xzgrep 的链接), xzgrep, xzless, 以及 xzmore
安装的库: liblzma.so
安装的目录: /usr/include/lzma 和 /usr/share/doc/xz-5.8.1

简要描述

lzcat	解压到标准输出
lzcmp	在 LZMA 压缩文件上执行 **cmp**
lzdiff	在 LZMA 压缩文件上执行 **diff**
lzegrep	在 LZMA 压缩文件上执行 **egrep**
lzfgrep	在 LZMA 压缩文件上执行 **fgrep**
lzgrep	在 LZMA 压缩文件上执行 **grep**
lzless	在 LZMA 压缩文件上执行 **less**
lzma	使用 LZMA 格式压缩或解压缩文件
lzmadec	一个轻量、快速的 LZMA 压缩文件解码器
lzmainfo	显示 LZMA 压缩文件头中存储的信息
lzmore	在 LZMA 压缩文件上执行 **more**
unlzma	使用 LZMA 格式解压缩文件

unxz 使用 XZ 格式解压缩文件
xz 使用 XZ 格式压缩或解压缩文件
xzcat 解压到标准输出
xzcmp 在 XZ 压缩文件上执行 **cmp**
xzdec 一个轻量、快速的 XZ 压缩文件解码器
xzdiff 在 XZ 压缩文件上执行 **diff**
xzegrep 在 XZ 压缩文件上执行 **egrep**
xzfgrep 在 XZ 压缩文件上执行 **fgrep**
xzgrep 在 XZ 压缩文件上执行 **grep**
xzless 在 XZ 压缩文件上执行 **less**
xzmore 在 XZ 压缩文件上执行 **more**
`liblzma` 实现基于 Lempel-Zip-Markov 链的无损块排序数据压缩算法的库

8.9. Lz4-1.10.0

Lz4 是一种无损压缩算法，其压缩速率可达每 CPU 核心 500 MB/s。它的优势是解压缩非常快，速率高达每 CPU 核心若干 GB/s。Lz4 可以和 Zstandard 共同使用以达到更高的压缩速率。

估计构建时间: 0.1 SBU
需要硬盘空间: 4.2 MB

8.9.1. 安装 Lz4

编译该软件包：

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

运行命令以测试编译结果：

```
make -j1 check
```

安装该软件包：

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

8.9.2. Lz4 的内容

安装的程序: lz4, lz4c (到 lz4 的链接), lz4cat (到 zstd 的链接), 以及 unlz4 (到 lz4 的链接)
安装的库: liblz4.so

简要描述

lz4 使用 LZ4 格式压缩或解压缩文件
lz4c 使用 LZ4 格式压缩文件
lz4cat 解压缩 LZ4 压缩文件并输出解压缩后的内容
unlz4 解压缩 LZ4 压缩文件
`liblz4` 基于 LZ4 算法实现无损数据压缩的库

8.10. Zstd-1.5.7

Zstandard 是一种实时压缩算法，提供了较高的压缩比。它具有很宽的压缩比/速度权衡范围，同时支持具有非常快速的解压缩。

估计构建时间: 0.4 SBU
需要硬盘空间: 86 MB

8.10.1. 安装 Zstd

编译该软件包：

```
make prefix=/usr
```

注意

在输出的测试结果中，可能会出现 'failed'。这是正常的，只有 'FAIL' 才表示测试失败。该软件包的测试应该能够全部通过。

运行命令以测试编译结果：

```
make check
```

安装该软件包：

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

删除静态库：

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

8.10.2. Zstd 的内容

安装的程序: zstd, zstdcat (到 zstd 的链接), zstdgrep, zstdless, zstdmt (到 zstd 的链接), 以及 unzstd (到 zstd 的链接)
安装的库: libzstd.so

简要描述

zstd 使用 ZSTD 格式压缩或解压缩文件
zstdgrep 在 ZSTD 压缩文件上运行 **grep**
zstdless 在 ZSTD 压缩文件上运行 **less**
`libzstd` 基于 ZSTD 算法实现无损数据压缩的库

8.11. File-5.46

File 软件包包含用于确定给定文件类型的工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 19 MB

8.11.1. 安装 File

准备编译 File:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.11.2. File 的内容

安装的程序: file
安装的库: libmagic.so

简要描述

file 通过进行文件系统、魔数和语言等测试，尝试对每个给定的文件进行分类

`libmagic` 包含 **file** 程序使用的魔数识别子程序

8.12. Readline-8.3

Readline 软件包包含一些提供命令行编辑和历史记录功能的库。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 17 MB

8.12.1. 安装 Readline

重新安装 Readline 会导致旧版本的库被重命名为 <库名称>.old。这一般不是问题，但某些情况下会触发 **ldconfig** 的一个链接 bug。运行下面的两条 sed 命令防止这种情况:

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

下面防止在共享库中硬编码库文件搜索路径 (rpath)。该软件包在安装到标准位置时并不需要 rpath，而且 rpath 在一些情况下会产生我们不希望的副作用，甚至导致安全问题:

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

准备编译 Readline:

```
./configure --prefix=/usr    \
            --disable-static \
            --with-curses    \
            --docdir=/usr/share/doc/readline-8.3
```

新的配置选项的含义:

`--with-curses`
该选项告诉 Readline 它可以在 curses 库中查找 termcap 库函数，而不是单独的 termcap 库。这样就能生成正确的 `readline.pc` 文件。

编译该软件包:

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

make 命令选项的含义:

`SHLIB_LIBS="-lncursesw"`
该选项强制 Readline 链接到 `libncursesw` 库。详见软件包 `README` 文件中“Shared Libraries” (共享库) 一节。

该软件包不包含测试套件。

安装该软件包:

```
make install
```

如果需要，安装该软件包的文档:

```
install -v -m644 doc/*.{ps,pdf,html,dvi} /usr/share/doc/readline-8.3
```

8.12.2. Readline 的内容

安装的库: libhistory.so 和 libreadline.so
安装的目录: /usr/include/readline 和 /usr/share/doc/readline-8.3

简要描述

`libhistory`	提供一个查询之前输入行的一致用户接口
`libreadline`	提供一组在程序的交互会话中操纵输入的文本的命令

8.13. M4-1.4.20

M4 软件包包含一个宏处理器。

估计构建时间: 0.4 SBU
需要硬盘空间: 60 MB

8.13.1. 安装 M4

准备编译 M4:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.13.2. M4 的内容

安装的程序: m4

简要描述

m4 复制给定文件，并展开它们包含的宏。这些宏可能是内置或用户定义的，可以接受任意个参数。除了展开宏外，**m4** 还包含用于包含指定文件、运行 Unix 命令、进行整数运算、处理文本、递归执行等功能的内建函数。**m4** 程序可以被用作编译器前端，也可以被单独用作宏处理器

8.14. Bc-7.0.3

Bc 软件包包含一个任意精度数值处理语言。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 7.8 MB

8.14.1. 安装 Bc

准备编译 Bc:

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

配置选项的含义:

`CC='gcc -std=c99'`
该选项指定编译时使用的编译器和 C 标准。

`-G`
忽略在 bc 程序未安装时无法工作的测试。

`-O3`
该选项指定编译时使用的优化等级。

`-r`
允许 bc 使用 Readline，以增强命令行编辑功能。

编译该软件包:

```
make
```

为了测试 bc，运行:

```
make test
```

安装该软件包:

```
make install
```

8.14.2. Bc 的内容

安装的程序: bc 和 dc

简要描述

bc 一个命令行计算器
dc 一个逆波兰式命令行计算器

8.15. Flex-2.6.4

Flex 软件包包含一个工具，用于生成在文本中识别模式的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 33 MB

8.15.1. 安装 Flex

准备编译 Flex:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

个别程序还不知道 **flex**，并试图去运行它的前身 **lex**。为了支持这些程序，创建一个名为 `lex` 的符号链接，它运行 `flex` 并启动其模拟 **lex** 的模式，同时将 **lex** 的手册页也创建为符号链接:

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

8.15.2. Flex 的内容

安装的程序: flex, flex++ (到 flex 的链接), 以及 lex (到 flex 的链接)
安装的库: libfl.so
安装的目录: /usr/share/doc/flex-2.6.4

简要描述

flex 一个用于生成在文本文件中识别模式的程序的工具；它允许灵活地指定查找模式的规则，消除了开发专用程序的需要

flex++ flex 的扩展，用于生成 C++ 代码和类。它是一个指向 **flex** 的符号链接

lex 一个以 **lex** 仿真模式运行 **flex** 的符号链接

`libfl` `flex` 库

8.16. Tcl-8.6.16

Tcl 软件包包含工具命令语言，它是一个可靠的通用脚本语言。Expect 软件包是用 Tcl (读作“tickle”) 编写的。

估计构建时间: 3.0 SBU
需要硬盘空间: 91 MB

8.16.1. 安装 Tcl

为了支持 Binutils，GCC，以及其他一些软件包测试套件的运行，需要安装这个软件包和接下来的两个 (Expect 与 DejaGNU)。为了测试目的安装三个软件包看似浪费，但是只有运行了测试，才能放心地确定多数重要工具可以正常工作，即使测试不是必要的。我们必须安装这些软件包，才能执行本章中的测试套件。

准备编译 Tcl:

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

新的配置选项的含义:

--disable-rpath
该选项阻止在二进制可执行文件和共享库中硬编码库文件搜索路径 (rpath)。该软件包在安装到标准位置时并不需要 rpath，而且 rpath 在一些情况下会产生我们不希望的副作用，甚至导致安全问题。

构建该软件包:

```
make

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

sed -e "s|$SRCDIR/unix/pkgs/tdbc1.1.10|/usr/lib/tdbc1.1.10|" \
    -e "s|$SRCDIR/pkgs/tdbc1.1.10/generic|/usr/include|"     \
    -e "s|$SRCDIR/pkgs/tdbc1.1.10/library|/usr/lib/tcl8.6|"  \
    -e "s|$SRCDIR/pkgs/tdbc1.1.10|/usr/include|"             \
    -i pkgs/tdbc1.1.10/tdbcConfig.sh

sed -e "s|$SRCDIR/unix/pkgs/itcl4.3.2|/usr/lib/itcl4.3.2|" \
    -e "s|$SRCDIR/pkgs/itcl4.3.2/generic|/usr/include|"    \
    -e "s|$SRCDIR/pkgs/itcl4.3.2|/usr/include|"            \
    -i pkgs/itcl4.3.2/itclConfig.sh

unset SRCDIR
```

“make”命令之后的若干“sed”命令从配置文件中删除构建目录，并用安装目录替换它们。构建 LFS 的后续过程不对此严格要求，但如果之后构建使用 Tcl 的软件包，则可能需要这样的操作。

运行命令以测试编译结果:

```
make test
```

安装该软件包:

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

将安装好的库加上写入权限，以便将来移除调试符号：

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

安装 Tcl 的头文件。下一个软件包 Expect 需要它们才能构建。

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

创建一个必要的符号链接：

```
ln -sfv tclsh8.6 /usr/bin/tclsh
```

重命名一个与 Perl 手册页文件名冲突的手册页：

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

如果需要，可以运行以下命令安装文档：

```
cd ..
tar -xf ../tcl8.6.16-html.tar.gz --strip-components=1
mkdir -v -p /usr/share/doc/tcl-8.6.16
cp -v -r  ./html/* /usr/share/doc/tcl-8.6.16
```

8.16.2. Tcl 的内容

安装的程序: tclsh (到 tclsh8.6 的链接) 和 tclsh8.6
安装的库: libtcl8.6.so 和 libtclstub8.6.a

简要描述

tclsh8.6	Tcl 命令行 shell
tclsh	一个指向 tclsh8.6 的链接
`libtcl8.6.so`	Tcl 运行库
`libtclstub8.6.a`	Tcl 端桩库

8.17. Expect-5.45.4

Expect 软件包包含通过脚本控制的对话，自动化 **telnet**, **ftp**, **passwd**, **fsck**, **rlogin**, 以及 **tip** 等交互应用的工具。Expect 对于测试这类程序也很有用，它简化了这类通过其他方式很难完成的工作。DejaGnu 框架是使用 Expect 编写的。

估计构建时间: 0.2 SBU
需要硬盘空间: 3.9 MB

8.17.1. 安装 Expect

Expect 需要伪终端 (PTY) 才能正常工作。进行简单测试以验证 PTY 是否在 chroot 环境中正常工作:

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

该命令应该输出 `ok`。如果该命令反而输出 `OSError: out of pty devices`，说明 PTY 在当前环境无法正常工作。此时需要退出 chroot 环境，再次阅读第 7.3 节 “准备虚拟内核文件系统”，并确认 `devpts` 文件系统 (以及其他虚拟内核文件系统) 已被正确挂载。之后按照第 7.4 节 “进入 Chroot 环境”重新进入 chroot 环境。在继续构建之前，必须解决这一问题，否则需要使用 Expect 的测试套件 (例如 Bash，Binutils，GCC，GDBM 等的测试套件，当然还有 Expect 本身的测试套件) 都会出现大规模的测试失败，而且也可能产生其他隐蔽的问题。

对该软件包进行一些修改，以允许使用 gcc-15.1 或更新版本构建它:

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

准备编译 Expect:

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

配置选项的含义:

`--with-tcl=/usr/lib`
需要使用该选项告知 **configure** 配置脚本 **tclConfig.sh** 的位置。

`--with-tclinclude=/usr/include`
该选项显式指定查找 Tcl 内部头文件的位置。

构建该软件包:

```
make
```

运行命令以测试编译结果:

```
make test
```

安装该软件包:

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

8.17.2. Expect 的内容

安装的程序: expect
安装的库: libexpect5.45.4.so

简要描述

expect 根据一个脚本与其他交互程序交流

libexpect-5.45.4.so	包含一些函数，使得 Expect 可以作为 Tcl 扩展使用，也可以直接在 C 或 C++ 中使用 (不使用 Tcl)

8.18. DejaGNU-1.6.3

DejaGnu 包含使用 GNU 工具运行测试套件的框架。它是用 **expect** 编写的，后者又使用 Tcl (工具命令语言)。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 6.9 MB

8.18.1. 安装 DejaGNU

DejaGNU 开发者建议在专用的目录中进行构建:

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

准备编译 DejaGNU:

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

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
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
```

8.18.2. DejaGNU 的内容

安装的程序: dejagnu 和 runtest

简要描述

dejagnu DejaGNU 辅助命令启动器

runtest 一个寻找正确的 **expect** shell，并运行 DejaGNU 的封装脚本

8.19. Pkgconf-2.5.1

pkgconf 软件包是 pkg-config 的接替者，它包含用于在软件包安装的配置和生成阶段向构建工具传递头文件或库文件搜索路径的工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 5.0 MB

8.19.1. 安装 Pkgconf

准备编译 Pkgconf:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/pkgconf-2.5.1
```

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

为了维持与原始的 Pkg-config 软件包的兼容性，创建两个符号链接:

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

8.19.2. Pkgconf 的内容

安装的程序: pkgconf, pkg-config (到 pkgconf 的链接), and bomtool
安装的库: libpkgconf.so
安装的目录: /usr/share/doc/pkgconf-2.5.1

简要描述

pkgconf 返回给定库或软件包的元信息

bomtool 根据 pkg-config .pc 文件生成软件物料清单

`libpkgconf` 包含 pkgconf 的大多数功能，以允许 IDE 和编译器等工具使用这些功能

8.20. Binutils-2.45

Binutils 包含汇编器、链接器以及其他用于处理目标文件的工具。

估计构建时间: 1.6 SBU
需要硬盘空间: 832 MB

8.20.1. 安装 Binutils

Binutils 文档推荐创建一个新的目录，以在其中构建 Binutils:

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

准备编译 Binutils:

```
../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
```

新的配置选项的含义:

`--enable-ld=default`
构建传统的 bfd 链接器，并且将它安装为 ld (默认链接器) 和 ld.bfd。

`--enable-plugins`
启用链接器插件支持。

`--with-system-zlib`
使用安装好的 zlib 库，而不是构建附带的版本。

编译该软件包:

```
make tooldir=/usr
```

make 命令参数的含义:

`tooldir=/usr`
一般来说，工具目录 (最终存放该软件包中可执行文件的目录) 被设定为 `$(exec_prefix)/$(target_alias)`。例如，在 x86_64 机器上，它将展开为 `/usr/x86_64-pc-linux-gnu`。因为 LFS 是定制系统，不需要 `/usr` 中的特定目标工具目录。如果系统用于交叉编译 (例如，在 Intel 机器上编译软件包，生成可以在 PowerPC 机器上执行的代码)，就会使用 `$(exec_prefix)/$(target_alias)` 目录。

重要

本节中，Binutils 的测试套件被认为是十分关键的，在任何情况下都不能跳过。

测试编译结果:

```
make -k check
```

如果需要列出所有失败的测试，执行:

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

安装该软件包：

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

删除无用的静态库等文件：

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

8.20.2. Binutils 的内容

安装的程序: addr2line, ar, as, c++filt, dwp, elfedit, gprof, gprofng, ld, ld.bfd, nm, objcopy, objdump, ranlib, readelf, size, strings, 以及 strip

安装的库: libbfd.so, libctf.so, libctf-nobfd.so, libgprofng.so, libopcodes.so, 以及 libsframe.so

安装的目录: /usr/lib/ldscripts

简要描述

addr2line 将程序中的地址翻译成文件名和行号；给定一个内存地址以及可执行程序的名字，该程序使用可执行文件中的调试信息，确定与该地址相关的源代码文件和行号

ar 创建、修改、提取档案文件

as 一个能够汇编 **gcc** 输出的汇编代码并生成目标文件的汇编器

c++filt 被链接器用于 demangle C++ 和 Java 符号，防止重载函数冲突

dwp DWARF 封装工具

elfedit 更改 ELF 文件的 ELF 头

gprof 显示函数调用图性能分析数据

gprofng 收集和分析性能数据

ld 一个链接器，将一些目标文件和档案文件组合为一个单独的文件，重定位它们的数据，并绑定符号引用

ld.bfd 一个指向 **ld.bfd** 的硬链接

nm 列出给定目标文件中的符号

objcopy 将一种目标文件翻译成另一种

objdump 显示给定目标文件的信息，通过命令行选项指定要显示哪些信息；这些信息对开发编译工具的程序员很有用

ranlib 生成档案文件内容的索引，并将索引存入档案文件；索引列出档案文件中所有可重定位目标文件定义的符号

readelf 显示 ELF 格式二进制文件的信息

size 列出给定文件各个段的大小和文件总大小

strings 对于每个给定文件，输出其中长度不小于给定长度 (默认是 4) 的可打印字符序列；对于目标文件，它默认只输出可加载的已初始化数据段中的字符串，对于其他文件，它扫描整个文件

strip 移除目标文件中的符号

`libbfd` 二进制文件描述符库

`libctf` 紧凑 ANSI-C 类型格式调试支持库

`libctf-nobfd` libctf 的变体，它不需要 libbfd 的功能

`libgprofng` 包含 **gprofng** 使用的多数子程序的库

libopcodes	一个用于处理操作码 —— 处理器指令的 “可读文本”版本的库；它被 **objdump** 等构建工具所使用
libsframe	使用简单的栈展开器支持在线栈回溯的库

8.21. GMP-6.3.0

GMP 软件包包含提供任意精度算术函数的数学库。

估计构建时间: 0.3 SBU
需要硬盘空间: 54 MB

8.21.1. 安装 GMP

注意

如果您在为 32 位 x86 构建 LFS，但您的 CPU 能够运行 64 位代码，而且 您指定了 `CFLAGS` 环境变量，配置脚本会试图为 64 位 CPU 进行配置并且失败。为了避免这个问题，像下面这样执行 configure 命令:

```
ABI=32 ./configure ...
```

注意

GMP 的默认设定会生成为本机处理器优化的库。如果您希望获得适合功能较弱的 CPU 的库，可以向 **configure** 命令传递 `--host=none-linux-gnu` 选项，以生成通用库。

首先，使得该软件包能够用 gcc-15 或更新版本构建:

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

准备编译 GMP:

```
./configure --prefix=/usr    \
            --enable-cxx     \
            --disable-static \
            --docdir=/usr/share/doc/gmp-6.3.0
```

新的配置选项的含义:

`--enable-cxx`
该参数启用 C++ 支持

`--docdir=/usr/share/doc/gmp-6.3.0`
该变量指定文档的正确位置

编译该软件包，并生成 HTML 文档:

```
make
make html
```

重要

我们认为，本节中 GMP 的测试套件是关键的。无论如何都不要跳过测试过程。

测试编译结果:

```
make check 2>&1 | tee gmp-check-log
```

小心

GMP 中的代码是针对本机处理器高度优化的。在偶然情况下，检测处理器的代码会错误识别 CPU 的功能，导致测试套件或使用 GMP 的其他程序输出消息 `Illegal instruction` (非法指令)。如果发生这种情况，需要使用选项 `--build=none-pc-linux-gnu` 重新配置 GMP 并重新构建它。

务必确认测试套件中至少 199 项测试通过。运行以下命令检验结果：

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

安装该软件包及其文档：

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

8.21.2. GMP 的内容

安装的库: libgmp.so 和 libgmpxx.so
安装的目录: /usr/share/doc/gmp-6.3.0

简要描述

`libgmp` 包含任意精度数学函数

`libgmpxx` 包含 C++ 任意精度数学函数

8.22. MPFR-4.2.2

MPFR 软件包包含多精度数学函数。

估计构建时间: 0.2 SBU
需要硬盘空间: 43 MB

8.22.1. 安装 MPFR

准备编译 MPFR:

```
./configure --prefix=/usr        \
            --disable-static     \
            --enable-thread-safe \
            --docdir=/usr/share/doc/mpfr-4.2.2
```

编译该软件包，并生成 HTML 文档:

```
make
make html
```

重要

本节中 MPFR 的测试套件被认为是非常关键的，无论如何不能跳过。

测试编译结果，并确认所有 198 项测试都能通过:

```
make check
```

安装该软件包及其文档:

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

8.22.2. MPFR 的内容

安装的库: libmpfr.so
安装的目录: /usr/share/doc/mpfr-4.2.2

简要描述

`libmpfr` 包含多精度数学函数

8.23. MPC-1.3.1

MPC 软件包包含一个任意高精度，且舍入正确的复数算术库。

估计构建时间: 0.1 SBU
需要硬盘空间: 22 MB

8.23.1. 安装 MPC

准备编译 MPC:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/mpc-1.3.1
```

编译该软件包，并生成 HTML 文档:

```
make
make html
```

运行命令以测试编译结果:

```
make check
```

安装该软件包及其文档:

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

8.23.2. MPC 的内容

安装的库: libmpc.so
安装的目录: /usr/share/doc/mpc-1.3.1

简要描述

`libmpc` 包含复数数学运算函数

8.24. Attr-2.5.2

Attr 软件包包含管理文件系统对象扩展属性的工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.1 MB

8.24.1. 安装 Attr

准备编译 Attr:

```
./configure --prefix=/usr     \
            --disable-static  \
            --sysconfdir=/etc \
            --docdir=/usr/share/doc/attr-2.5.2
```

编译该软件包:

```
make
```

测试套件必须在支持扩展属性的文件系统，如 ext2、ext3 或 ext4 上运行。运行下列命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.24.2. Attr 的内容

安装的程序: attr, getfattr, 以及 setfattr
安装的库: libattr.so
安装的目录: /usr/include/attr 和 /usr/share/doc/attr-2.5.2

简要描述

attr	在文件系统对象上扩展属性
getfattr	查询文件系统对象的扩展属性
setfattr	设定文件系统对象的扩展属性
`libattr`	包含处理扩展属性的库函数

8.25. Acl-2.3.2

Acl 软件包包含管理访问控制列表的工具，访问控制列表能够细致地自由定义文件和目录的访问权限。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 6.5 MB

8.25.1. 安装 Acl

准备编译 Acl:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/acl-2.3.2
```

编译该软件包:

```
make
```

测试套件必须在支持访问控制的文件系统上运行。运行下列命令以测试编译结果:

```
make check
```

已知名为 `test/cp.test` 的一项测试会由于 Coreutils 的 Acl 支持尚未构建而失败。

安装该软件包:

```
make install
```

8.25.2. Acl 的内容

安装的程序: chacl, getfacl, 以及 setfacl
安装的库: libacl.so
安装的目录: /usr/include/acl 和 /usr/share/doc/acl-2.3.2

简要描述

chacl 修改文件或目录的访问控制列表
getfacl 获取文件访问控制列表
setfacl 设定文件访问控制列表
`libacl` 包含操作访问控制列表的库函数

8.26. Libcap-2.76

Libcap 软件包为 Linux 内核提供的 POSIX 1003.1e 权能字实现用户接口。这些权能字是 root 用户的最高特权分割成的一组不同权限。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 3.1 MB

8.26.1. 安装 Libcap

防止静态库的安装:

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

编译该软件包:

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

make 命令选项的含义:

`lib=lib`
在 x86_64 上，该参数将库文件目录设定为 `/usr/lib`，而不是 `/usr/lib64`。它在 x86 上没有作用。

运行命令以测试编译结果:

```
make test
```

安装该软件包:

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

8.26.2. Libcap 的内容

安装的程序: capsh, getcap, getpcaps, 以及 setcap
安装的库: libcap.so 和 libpsx.so

简要描述

capsh	一个用于演示和限制 Linux 权能字的 shell 封装器
getcap	检验文件权能字
getpcaps	查询进程的权能字
setcap	设定文件权能字
`libcap`	包含操作 POSIX 1003.1e 权能字的库函数
`libpsx`	包含为 pthread 库相关的系统调用提供 POSIX 语义的函数

8.27. Libxcrypt-4.4.38

Libxcrypt 软件包包含用于对密码进行单向散列操作的，现代化的库。

估计构建时间: 0.1 SBU
需要硬盘空间: 12 MB

8.27.1. 安装 Libxcrypt

准备编译 Libxcrypt:

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

新的配置选项的含义:

`--enable-hashes=strong,glibc`
构建对于安全相关的用途来说推荐使用的高强度散列算法，为了兼容性，同时构建传统的 Glibc `libcrypt` 提供的散列算法。

`--enable-obsolete-api=no`
禁用过时的 API 函数。它们对于从源代码构建的现代 Linux 系统来说没有必要。

`--disable-failure-tokens`
禁用失败标识功能。它用于满足与一些平台的传统散列算法库的兼容性，但是基于 Glibc 的 Linux 系统不需要它。

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

注意

上述指令禁用了过时的 API 函数，因为从源码编译的软件包不会在运行时链接到它们。然而，已知的需要链接到这些函数的二进制程序都需要 ABI 版本 1。如果您为了满足一些仅有二进制版本的程序，或者满足 LSB 兼容性，必须使用这些函数，执行以下命令再次构建该软件包:

```
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
```

8.27.2. Libxcrypt 的内容

安装的库: libcrypt.so

简要描述

`libcrypt` 包含密码散列函数

8.28. Shadow-4.18.0

Shadow 软件包包含安全地处理密码的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 115 MB

8.28.1. 安装 Shadow

重要

如果已经安装了 Linux-PAM，则应该按照 BLFS shadow 页面，而非本页的说明安装 shadow。

注意

如果您希望强制使用强密码，首先安装并配置 Linux-PAM。之后安装并配置带有 PAM 支持的 shadow。最后安装 libpwquality 并配置 PAM 使用它。

禁止该软件包安装 **groups** 程序和它的手册页，因为 Coreutils 会提供更好的版本。同样，避免安装第 8.3 节 “Man-pages-6.15”软件包已经提供的手册页：

```
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 / /'   {} \;
```

不使用默认的 crypt 加密方法，使用安全程度高很多的 YESCRYPT 算法加密密码，这也允许设定长度超过 8 个字符的密码。还需要把过时的用户邮箱位置 `/var/spool/mail` 改为当前普遍使用的 `/var/mail` 目录。另外，从默认的 `PATH` 中删除`/bin` 和 `/sbin`，因为它们只是指向 `/usr` 中对应目录的符号链接：

警告

将 `/bin` 和/或 `/sbin` 包含在 `PATH` 中会导致一些 BLFS 软件包构建失败，因此不要通过 `.bashrc` 等这样做。

```
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
```

准备编译 Shadow：

```
touch /usr/bin/passwd
./configure --sysconfdir=/etc   \
            --disable-static    \
            --with-{b,yes}crypt \
            --without-libbsd    \
            --with-group-name-max-length=32
```

新的配置选项的含义：

touch /usr/bin/passwd
我们需要保证 `/usr/bin/passwd` 存在，因为它的位置会被硬编码到一些程序中；且如果它不存在的话，安装脚本会在错误的位置创建它。

`--with-{b,yes}crypt`
Shell 将它展开为两个选项，`--with-bcrypt` 和 `--with-yescrypt`。它们允许 shadow 使用 Libxcrypt 实现的 Bcrypt 和 Yescrypt 算法对密码进行散列操作。这些算法相比于传统的 SHA 系列算法更安全 (特别是对基于 GPU 的攻击抵抗力强很多)。

`--with-group-name-max-length=32`
用户名最长只能有 32 个字符。设定组名称最大长度为相同值。

`--without-libbsd`
不要使用 LFS 不包含的 libbsd 库中的 readpassphrase 函数。使用软件包内置的 readpassphrase 实现代替它。

编译该软件包:

```
make
```

该软件包不包含测试套件。

安装该软件包:

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

8.28.2. 配置 Shadow

该软件包包含用于添加、修改、删除用户和组，设定和修改它们的密码，以及进行其他管理任务的工具。如果希望查阅关于 password shadowing 的详细解释，阅读解压得到源代码目录树中的 `doc/HOWTO` 文件。如果使用 Shadow 支持，请注意所有需要验证密码的程序 (如显示管理器、FTP 程序、pop3 守护进程等) 都必须和 Shadow 兼容。换句话说，它们必须能使用 Shadow 加密的密码。

如果要对用户密码启用 Shadow 加密，执行以下命令:

```
pwconv
```

如果要对组密码启用 Shadow 加密，执行:

```
grpconv
```

下面需要解释 Shadow 中 **useradd** 的默认配置。首先，**useradd** 的默认操作是创建一个用户，以及一个名字和用户名相同的组。默认情况下，用户 ID (UID) 和组 ID (GID) 会从 1000 开始。这意味着，如果您不向 **useradd** 传递参数，每个用户都会属于一个不同的组。如果您不希望这样，就要传递 `-g` 或者 `-N` 参数给 **useradd**，或者在`/etc/login.defs` 中修改 `USERGROUPS_ENAB` 的值。参阅 useradd(8) 了解更多相关信息。

其次，为了修改默认参数，必须创建 `/etc/default/useradd` 文件，并定制其内容，以满足您的特定需要。使用以下命令创建它:

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

/etc/default/useradd 参数解释

`GROUP=999`
该参数设定 /etc/group 文件中使用的第一个组编号。这里的值 999 是在上面的 `--gid` 参数中给定的。您可以将它修改为您希望的任何值。注意，**useradd** 绝不会重用 UID 或 GID。如果该参数指定的数字已经被使用了，它就会使用下一个可用的数字。另外，如果您第一次使用不带 `-g` 参数的 **useradd** 命令时没有编号 999 的组，您就会在终端看到错误消息 —— `useradd: unknown GID 999`，尽管用户账号仍会被正常创建。为了防止这种现象的出现，我们在第 7.6 节 “创建必要的文件和符号链接”中已经用编号 999 创建了 `users` 组。

`CREATE_MAIL_SPOOL=yes`
该参数使得 **useradd** 为每个新用户创建邮箱文件。**useradd** 会使得 `mail` 为邮箱文件属组，并为邮箱文件赋予 0660 权限模式。如果您不希望 **useradd** 创建邮箱文件，执行以下命令:

```
sed -i '/MAIL/s/yes/no/' /etc/default/useradd
```

8.28.3. 设定根用户密码

为用户 root 选择一个密码，并执行以下命令设定它：

```
passwd root
```

8.28.4. Shadow 的内容

安装的程序: 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 (到 newgrp 的链接), su, useradd, userdel, usermod, vigr (到 vipw 的链接), 以及 vipw
安装的目录: /etc/default 和 /usr/include/shadow
安装的库: libsubid.so

简要描述

chage 用于修改强制性密码更新的最大天数
chfn 用于修改用户全名和其他信息
chgpasswd 用于批量更新组密码
chpasswd 用于批量更新用户密码
chsh 用于改变用户的默认登录 shell
expiry 检查并强制保证当前密码过期策略
faillog 用于检查失败登录日志，设定锁定账户的最大失败次数，以及重置失败次数
getsubids 用于列出一个用户的辅助 ID 范围
gpasswd 用于增加或删除组的用户和管理员
groupadd 以指定名称创建组
groupdel 删除指定的组
groupmems 允许用户在不需要超级用户权限的情况下，管理自己的组成员列表
groupmod 用于修改给定的组名称或 GID
grpck 验证组文件 `/etc/group` 和 `/etc/gshadow` 的完整性
grpconv 根据普通组文件创建或更新加密组文件
grpunconv 根据 `/etc/gshadow` 文件更新 `/etc/group` 文件，并删除前者
login 被系统用于允许用户登录
logoutd 是一个限制登录时间和端口的守护进程
newgidmap 用于设定用户命名空间的 gid 映射
newgrp 用于在登录会话中修改当前 GID
newuidmap 用于设定用户命名空间的 uid 映射
newusers 用于批量创建或更新用户账户
nologin 显示一条账户不可用的消息；它被设计为充当已禁用账户的默认 shell
passwd 用于修改用户或组账户的密码
pwck 检验密码文件 `/etc/passwd` 和 `/etc/shadow` 的完整性
pwconv 从普通密码文件创建或更新加密密码文件

pwunconv 根据 /etc/shadow 更新 /etc/passwd 并删除前者
sg 在用户 GID 设为给定组 ID 的情况下，执行给定命令
su 用替换的用户和组 ID 运行 shell
useradd 以指定名称创建新用户，或更新新用户默认信息
userdel 删除给定用户
usermod 用于修改给定用户的登录名称、用户标识符 (UID)、shell、初始组、home 目录等信息
vigr 编辑 /etc/group 或 /etc/gshadow 文件
vipw 编辑 /etc/passwd 或 /etc/shadow 文件
libsubid 用于操作用户或组的辅助 ID 范围的库

8.29. GCC-15.2.0

GCC 软件包包含 GNU 编译器集合，其中有 C 和 C++ 编译器。

估计构建时间: 46 SBU (已计入测试时间)
需要硬盘空间: 6.6 GB

8.29.1. 安装 GCC

在 x86_64 上构建时，修改存放 64 位库的默认路径为 “lib”：

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

GCC 文档建议在一个新建的目录中构建 GCC:

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

准备编译 GCC:

```
../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 支持七种程序设计语言，但其中多数语言需要尚未安装的依赖项。阅读 BLFS 手册，以了解如何构建 GCC 支持的所有语言。

新的配置选项的含义：

`LD=ld`
该选项使得配置脚本使用之前在本章中构建的 Binutils 提供的 ld 程序，而不是交叉编译构建的版本。

`--disable-fixincludes`
默认情况下，在安装 GCC 时，一些系统头文件会被“修复”以供 GCC 使用。这对于现代的 Linux 系统来说是不必要的，而且如果一个软件包在安装 GCC 后被重新安装，还可能造成损害。这个开关阻止 GCC 尝试“修复”头文件。

`--with-system-zlib`
该选项使得 GCC 链接到系统安装的 Zlib 库，而不是它自带的 Zlib 副本。

注意

PIE (位置无关可执行文件) 是能加载到内存中任意位置的二进制程序。在不使用 PIE 时，称为 ASLR (地址空间布局随机化) 的安全特性能被用于共享库，但不能被用于可执行程序本身。启用 PIE 使得 ASLR 在作用于共享库的同时，同样作用于可执行程序，以预防一些基于可执行程序中关键代码或数据的固定地址的攻击。

SSP (栈溢出防护) 是保证程序的调用栈不被破坏的技术。在调用栈被破坏时可能导致安全问题，例如子程序的返回地址可能被修改，进而执行一些危险代码 (这些危险代码可能已经存在于程序或共享库中，或被攻击者用某种方式注入)。

编译该软件包：

```
make
```

重要

在本节中，GCC 的测试套件十分重要，但需要消耗较长的时间。我们建议首次编译 LFS 的读者运行测试套件。通过在以下命令中添加 -jx 参数，可以显著降低测试需要的时间，其中 x 表示系统 CPU 核心数。

GCC 在编译一些具有复杂模式的代码时，可能需要更多的栈空间。万一宿主系统的栈空间限制较为严格，我们需要手工将栈空间的硬上限设为无限大。在多数宿主发行版 (和最终的 LFS 系统) 中，默认的硬上限值已经是无限大，但手工调整限制无论如何都不会造成损害。我们不需要调整软上限，因为 GCC 会自动将软上限设为合适的值，只要确保该值不会超过硬限制即可。

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

现在移除若干已知会失败的测试：

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

以非特权用户身份测试编译结果，但出错时继续执行其他测试：

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

输入以下命令提取测试结果的摘要：

```
../contrib/test_summary
```

如果只想看摘要，将输出用管道送至 **grep -A7 Summ**。

可以将结果与 https://www.linuxfromscratch.org/lfs/build-logs/12.4/ 和 https://gcc.gnu.org/ml/gcc-testresults/ 的结果进行比较。

已知和 `pr90579.c` 有关的测试会失败。

少量意外的失败有时无法避免。某些情况下特定的硬件配置可能触发测试失败。我们可以放心地忽略测试失败并继续构建系统，除非测试结果和上述 URL 提供的结果截然不同。

安装该软件包：

```
make install
```

GCC 构建目录目前属于用户 `tester`，导致安装的头文件目录 (及其内容) 具有不正确的所有权。将所有者修改为 `root` 用户和组：

```
chown -v -R root:root \
    /usr/lib/gcc/$(gcc -dumpmachine)/15.2.0/include{,-fixed}
```

创建一个 FHS 因 “历史原因” 要求的符号链接。

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

许多软件包使用 **cc** 这一名称调用 C 编译器。在第二遍的 GCC 中我们已经将 **cc** 创建为符号链接，这里将其手册页也创建为符号链接：

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

创建一个兼容性符号链接，以支持在构建程序时使用链接时优化 (LTO)：

```
ln -sfv ../../libexec/gcc/$(gcc -dumpmachine)/15.2.0/liblto_plugin.so \
        /usr/lib/bfd-plugins/
```

现在最终的工具链已经就位，重要的是再次确认编译和链接像我们期望的一样正常工作。为此，进行下列完整性检查：

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

上述命令不应该出现错误，最后一行命令输出的结果应该 (不同平台的动态链接器名称可能不同) 是：

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

下面确认我们在使用正确的启动文件：

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

以上命令应该输出：

```
/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
```

以上结果可能随您的机器体系结构不同而略微不同。差异在于 `/usr/lib/gcc` 之后的目录名。我们关注的重点是，**gcc** 应该找到所有三个 `crt*.o` 文件，它们应该位于 `/usr/lib` 目录中。

确认编译器能正确查找头文件：

```
grep -B4 '^ /usr/include' dummy.log
```

该命令应当输出：

```
#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
```

同样要注意，以您的目标三元组命名的目录由于您体系结构的不同，可能和以上不同。

下一步确认新的链接器使用了正确的搜索路径：

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

那些包含 '-linux-gnu' 的路径应该忽略，除此之外，以上命令应该输出：

```
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");
```

在 32 位系统上可能使用不同的目录。例如，下面是 i686 机器上的输出：

```
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");
```

之后确认我们使用了正确的 libc：

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

以上命令应该输出：

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

确认 GCC 使用了正确的动态链接器：

```
grep found dummy.log
```

以上命令应该输出 (不同平台的动态链接器名称可能不同)：

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

如果输出和以上描述不符，或者根本没有输出，那么必然有什么地方出了严重错误。检查并重新跟踪以上步骤，找到问题的原因，并修复它。在继续构建前必须解决这里发现的所有问题。

在确认一切工作良好后，删除测试文件：

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

最后移动一个位置不正确的文件：

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

8.29.2. GCC 的内容

安装的程序: c++, cc (到 gcc 的链接), cpp, g++, gcc, gcc-ar, gcc-nm, gcc-ranlib, gcov, gcov-dump, gcov-tool, 以及 lto-dump

安装的库: 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}, 以及 libubsan.{a,so}

安装的目录: /usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc, 以及 /usr/share/gcc-15.2.0

简要描述

c++	C++ 编译器
cc	C 编译器
cpp	C 预处理器；编译器使用它展开源文件中的 #include、#define，以及类似指令
g++	C++ 编译器
gcc	C 编译器
gcc-ar	**ar** 的一个包装器，它在命令行中添加一个插件。这个程序只被用于提供链接时优化功能，对于默认的构建选项来说没有作用。
gcc-nm	**nm** 的一个包装器，它在命令行中添加一个插件。这个程序只被用于提供链接时优化功能，对于默认的构建选项来说没有作用。
gcc-ranlib	**ranlib** 的一个包装器，它在命令行中添加一个插件。这个程序只被用于提供链接时优化功能，对于默认的构建选项来说没有作用。
gcov	一个覆盖率测试工具；用于分析程序以确定在哪里优化最有效
gcov-dump	离线 gcda 和 gcno 性能剖析数据显示工具
gcov-tool	离线 gcda 性能剖析预处理工具
lto-dump	用于转储 GCC 启用 LTO 时生成的目标文件
`libasan`	地址完整性检查库

libatomic	GCC 内建原子操作运行库
libcc1	允许 GDB 使用 GCC 功能的库
libgcc	包含 **gcc** 的运行时支持
libgcov	在 GCC 被指示启动性能剖析时，这个库被链接到程序中
libgomp	OpenMP API 的 GNU 实现，用于 C/C++ 和 Fortran 的跨平台共享内存并行编程
libhwasan	硬件辅助地址完整性检查库
libitm	GNU 事务内存库
liblsan	内存泄露清理检查库
liblto_plugin	GCC 的 LTO 插件，允许 Binutils 处理 GCC 在启用 LTO 时生成的目标文件
libquadmath	GCC 四精度数学 API 库
libssp	包含 GCC 的栈溢出保护功能支持子程序。一般不使用它，因为 Glibc 也提供这些子程序。
libstdc++	C++ 标准库
libstdc++exp	实验性的 C++ Contracts 库
libstdc++fs	ISO/IEC TS 18822:2015 文件系统库
libsupc++	包含 C++ 编程语言支持子程序
libtsan	线程完整性检查库
libubsan	未定义行为清理检查库

8.30. Ncurses-6.5-20250809

Ncurses 软件包包含使用时不需考虑终端特性的字符屏幕处理函数库。

估计构建时间: 0.2 SBU
需要硬盘空间: 46 MB

8.30.1. 安装 Ncurses

准备编译 Ncurses:

```
./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
```

新的配置选项的含义:

`--with-shared`
该选项使得 Ncurses 将 C 函数库构建并安装为共享库。

`--without-normal`
该选项禁止将 C 函数库构建和安装为静态库。

`--without-debug`
该选项禁止构建和安装用于调试的库。

`--with-cxx-shared`
该选项使得 Ncurses 将 C++ 绑定构建并安装为共享库，同时防止构建和安装静态的 C++ 绑定库。

`--enable-pc-files`
该参数使得构建系统生成并安装 pkg-config 使用的 .pc 文件。

编译该软件包:

```
make
```

该软件包有测试套件，但只能在安装该软件包后才能运行。测试用例位于 `test/` 中。阅读其中的 `README` 文件了解更多细节。

安装该软件包会直接覆盖文件 `libncursesw.so.6.5`。这可能导致正在使用该库文件中的代码和数据的 shell 进程发生崩溃。因此，需要使用 `DESTDIR` 进行安装，并正确地使用 **install** 命令安装库文件:

```
make DESTDIR=$PWD/dest install
install -vm755 dest/usr/lib/libncursesw.so.6.5 /usr/lib
rm -v  dest/usr/lib/libncursesw.so.6.5
sed -e 's/^#if.*XOPEN.*$/#if 1/' \
    -i dest/usr/include/curses.h
cp -av dest/* /
```

许多程序仍然希望链接器能够找到非宽字符版本的 Ncurses 库。通过使用符号链接和链接脚本，诱导它们链接到宽字符库 (注意这里扩展名为 `so` 的链接只有在已经编辑 `curses.h` 使之总是使用宽字符 ABI 时才安全):

```
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
```

最后，确保那些在构建时寻找 -lcurses 的老式程序仍然能够构建：

```
ln -sfv libncursesw.so /usr/lib/libcurses.so
```

如果需要的话，安装 Ncurses 文档：

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

注意

上述指令没有创建非宽字符的 Ncurses 库，因为从源码编译的软件包不会在运行时链接到它。然而，已知的需要链接到非宽字符 Ncurses 库的二进制程序都需要版本 5。如果您为了满足一些仅有二进制版本的程序，或者满足 LSB 兼容性，必须安装这样的库，执行以下命令再次构建该软件包：

```
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
```

8.30.2. Ncurses 的内容

安装的程序: captoinfo (链接到 tic), clear, infocmp, infotocap (链接到 tic), ncursesw6-config, reset (链接到 tset), tabs, tic, toe, tput, 以及 tset

安装的库: libcurses.so (符号链接), libform.so (符号链接), libformw.so, libmenu.so (符号链接), libmenuw.so, libncurses.so (符号链接), libncursesw.so, libncurses++w.so, libpanel.so (符号链接), 以及 libpanelw.so,

安装的目录: /usr/share/tabset, /usr/share/terminfo, 以及 /usr/share/doc/ncurses-6.5-20250809

简要描述

captoinfo 将 termcap 描述转换成 terminfo 描述

clear 如果可能的话，清空屏幕

infocmp 比较或输出 terminfo 描述

infotocap 将 terminfo 描述转化为 termcap 描述

ncursesw6-config 提供 ncurses 的配置信息

reset 以终端默认值重新初始化终端

tabs 清除并设置终端的制表符宽度

tic Terminfo 条目描述编译器，将 terminfo 文件从源代码格式翻译为 ncurses 库子程序需要的二进制格式 [terminfo 文件包含特定终端的功能信息。]

toe 列出所有可用的终端类型，并给出每种类型的主要名称和描述

tput 使 shell 可以使用终端相关的功能；也可以重置或初始化终端，或者报告它的长名称

tset 可以被用于初始化终端

`libncursesw` 包含在终端屏幕上以多种复杂方式显示文本的函数；使用这些函数的典型例子是运行内核的 **make menuconfig** 时显示的目录

libncurses++w	包含该软件包中其他库的 C++ 语言绑定
libformw	包含实现表单的函数
libmenuw	包含实现目录的函数
libpanelw	包含实现面板的函数

8.31. Sed-4.9

Sed 软件包包含一个流编辑器。

估计构建时间: 0.3 SBU
需要硬盘空间: 30 MB

8.31.1. 安装 Sed

准备编译 Sed:

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

编译该软件包，并生成 HTML 文档:

```
make
make html
```

运行命令以测试编译结果:

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

安装该软件包及其文档:

```
make install
install -d -m755           /usr/share/doc/sed-4.9
install -m644 doc/sed.html /usr/share/doc/sed-4.9
```

8.31.2. Sed 的内容

安装的程序: sed
安装的目录: /usr/share/doc/sed-4.9

简要描述

sed 一次性过滤和转换文本文件

8.32. Psmisc-23.7

Psmisc 软件包包含显示正在运行的进程信息的程序。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 6.7 MB

8.32.1. 安装 Psmisc

准备编译 Psmisc:

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

编译该软件包:

```
make
```

如果要运行测试套件，执行命令:

```
make check
```

安装该软件包:

```
make install
```

8.32.2. Psmisc 的内容

安装的程序: fuser, killall, peekfd, prtstat, pslog, pstree, 以及 pstree.x11 (到 pstree 的链接)

简要描述

fuser	报告使用给定文件或文件系统的进程 ID (PID)
killall	根据名称杀死进程；它向所有运行给定命令的进程发送信号
peekfd	根据给定 PID，查看正在运行进程的文件描述符
prtstat	打印某个进程的信息
pslog	报告某个进程当前使用的日志路径
pstree	以树形格式列出正在运行的进程
pstree.x11	除了在退出前等待用户确认外，和 **pstree**相同

8.33. Gettext-0.26

Gettext 软件包包含国际化和本地化工具，它们允许程序在编译时加入 NLS (本地语言支持) 功能，使它们能够以用户的本地语言输出消息。

估计构建时间: 2.1 SBU
需要硬盘空间: 395 MB

8.33.1. 安装 Gettext

准备编译 Gettext:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/gettext-0.26
```

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

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

8.33.2. Gettext 的内容

安装的程序: autopoint, envsubst, gettext, gettext.sh, gettextize, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, recode-sr-latin, 以及 xgettext
安装的库: libasprintf.so, libgettextlib.so, libgettextpo.so, libgettextsrc.so, libtextstyle.so, 以及 preloadable_libintl.so
安装的目录: /usr/lib/gettext, /usr/share/doc/gettext-0.26, /usr/share/gettext, 以及 /usr/share/gettext-0.26

简要描述

autopoint 将标准 Gettext 微架构文件复制到源代码包
envsubst 替换 shell 格式化字符串中的环境变量
gettext 通过查询消息目录中的翻译，将中性语言消息翻译成用户的语言
gettext.sh 主要用于 gettext 的 shell 函数库
gettextize 将所有标准 Gettext 文件复制到软件包顶层目录中，以开始国际化该软件包
msgattrib 根据属性过滤翻译目录中的消息，或修改这些属性
msgcat 连接并合并给定的 .po 文件
msgcmp 比较两个 .po 文件，以检查它们是否包含相同的 msgid 字符串集合
msgcomm 找出给定的多个 .po 中的公共消息
msgconv 将翻译目录转换成另一种字符编码
msgen 创建英文翻译目录
msgexec 对翻译目录中的所有翻译执行命令

msgfilter	对翻译目录中的所有翻译应用过滤器
msgfmt	根据翻译目录创建二进制消息目录
msggrep	找出翻译目录中所有匹配给定模式，或属于给定源代码文件的消息
msginit	创建一个新的 .po 文件，以用户环境中的值初始化其元信息
msgmerge	将两个原始翻译文件组合成一个文件
msgunfmt	反编译二进制消息目录，生成原始翻译文本
msguniq	去除翻译目录中重复的翻译
ngettext	显示某条语法形式依赖于数字的文本消息的母语翻译
recode-sr-latin	将塞尔维亚语文本从西里尔字符转换为拉丁字符
xgettext	从给定源代码文件中提取可翻译消息，以生成最初的翻译模板
libasprintf	定义 autosprintf 类，使得 C 格式化输出子程序在 C++ 程序中能够与 <string> 字符串和 <iostream> 流一起使用
libgettextlib	包含若干 Gettext 程序的公共子程序；并未设计为供其他程序使用
libgettextpo	用于编写处理 .po 文件的专用程序；这个库在 Gettext 发行的标准程序 (如 **msgcomm**, **msgcmp**, **msgattrib**, 以及 **msgen**) 不能满足要求时使用
libgettextsrc	包含若干 Gettext 程序的公共子程序；并未设计为供其他程序使用
libtextstyle	文本样式库
preloadable_libintl	一个被设计为由 LD_PRELOAD 预加载的库，它帮助 libintl 记录未翻译的消息

8.34. Bison-3.8.2

Bison 软件包包含语法分析器生成器。

估计构建时间: 2.1 SBU
需要硬盘空间: 63 MB

8.34.1. 安装 Bison

准备编译 Bison:

```
./configure --prefix=/usr --docdir=/usr/share/doc/bison-3.8.2
```

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.34.2. Bison 的内容

安装的程序: bison 和 yacc
安装的库: liby.a
安装的目录: /usr/share/bison

简要描述

bison 根据一组规则，创建一个用于分析文本文件结构的程序；Bison 是 Yacc (Yet Another Compiler Compiler) 的替代品

yacc **bison** 的一个封装器，被那些仍然调用 **yacc** 而非 **bison** 的程序使用，它调用 **bison** 时附加 -y 选项

`liby` Yacc 库包含与 Yacc 兼容的 `yyerror` 和 `main` 函数实现；它并不是很有用，但 POSIX 需要它

8.35. Grep-3.12

Grep 软件包包含在文件内容中进行搜索的程序。

估计构建时间: 0.6 SBU
需要硬盘空间: 48 MB

8.35.1. 安装 Grep

首先，移除使用 egrep 和 fgrep 时的警告，它会导致一些软件包测试失败：

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

准备编译 Grep：

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

编译该软件包：

```
make
```

运行命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.35.2. Grep 的内容

安装的程序: egrep, fgrep, 以及 grep

简要描述

egrep 打印与固定字符串匹配的行。已弃用，使用 **grep -E** 代替它
fgrep 打印与固定字符串匹配的行。已弃用，使用 **grep -F** 代替它
grep 打印与基本正则表达式匹配的行

8.36. Bash-5.3

Bash 软件包包含 Bourne-Again Shell。

估计构建时间: 1.5 SBU
需要硬盘空间: 56 MB

8.36.1. 安装 Bash

准备编译 Bash:

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

新的配置选项的含义:

`--with-installed-readline`
该选项告诉 Bash 使用系统中已经安装的 `readline` 库，而不是它自己的 readline 版本。

编译该软件包:

```
make
```

如果不运行测试套件，直接跳到 “安装该软件包”。

为了准备进行测试，确保 `tester` 用户可以写入源代码目录:

```
chown -R tester .
```

该软件包的测试套件被设计为以非 `root` 用户身份运行，且该用户必须是标准输入所连接的终端的所有者。为了满足这一条件，使用 Expect 生成一个新的伪终端，并以 `tester` 用户身份运行测试:

```
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
```

测试套件使用 **diff** 检测测试脚本输出和预期是否存在不同。**diff** 的任何输出 (以 < 和 > 开头) 都表示测试失败，除非有消息提示可以忽略这里报告的差别。已知一项名为 `run-builtins` 的测试在一些宿主系统上可能失败，此时该测试输出的 479 和 480 行和预期存在不同。已知其他一些使用 `zh_TW.BIG5` 和 `ja_JP.SJIS` locale 的测试在未安装对应 locale 时会失败。

安装该软件包:

```
make install
```

执行新编译的 **bash** 程序 (替换当前正在执行的版本):

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

8.36.2. Bash 的内容

安装的程序: bash, bashbug, 以及 sh (到 bash 的链接)
安装的目录: /usr/include/bash, /usr/lib/bash, 和 /usr/share/doc/bash-5.3

简要描述

bash 一个广泛使用的命令解释器；它在执行命令前对命令行进行多种展开和替换操作，这些操作使得它成为强大的工具

bashbug	一个 shell 脚本，用于帮助用户按照电子邮件标准格式编写关于 **bash** 的 bug 报告
sh	一个指向 **bash** 程序的符号链接；当以 **sh** 命令运行时，**bash** 试图尽可能地模仿 **sh** 的历史版本，以符合 POSIX 标准

8.37. Libtool-2.5.4

Libtool 软件包包含 GNU 通用库支持脚本。它提供一致、可移植的接口，以简化共享库的使用。

估计构建时间: 0.6 SBU
需要硬盘空间: 44 MB

8.37.1. 安装 Libtool

准备编译 Libtool:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

删除仅在运行测试时有实际意义的静态库:

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

8.37.2. Libtool 的内容

安装的程序: libtool 和 libtoolize
安装的库: libltdl.so
安装的目录: /usr/include/libltdl 和 /usr/share/libtool

简要描述

libtool	提供通用化库文件构建支持服务
libtoolize	提供为软件包增加 **libtool** 支持的标准方法
`libltdl`	隐藏使用动态加载库时可能遇到的若干困难

8.38. GDBM-1.26

GDBM 软件包包含 GNU 数据库管理器。它是一个使用可扩展散列的数据库函数库，功能类似于标准的 UNIX dbm。该库提供用于存储键值对、通过键搜索和获取数据，以及删除键和对应数据的原语。

估计构建时间: 不到 0.2 SBU
需要硬盘空间: 13 MB

8.38.1. 安装 GDBM

准备编译 GDBM：

```
./configure --prefix=/usr    \
            --disable-static \
            --enable-libgdbm-compat
```

配置选项的含义：

`--enable-libgdbm-compat`
该选项启用 libgdbm 兼容性库的构建。LFS 之外的一些软件包需要它提供的老式 DBM 子程序。

编译该软件包：

```
make
```

运行命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.38.2. GDBM 的内容

安装的程序: gdbm_dump, gdbm_load, 以及 gdbmtool
安装的库: libgdbm.so 和 libgdbm_compat.so

简要描述

gdbm_dump	将 GDBM 数据库转储到文件
gdbm_load	从转储文件重建 GDBM 数据库
gdbmtool	测试和修改 GDBM 数据库
`libgdbm`	包含用于操作散列数据库的函数
`libgdbm_compat`	包含老式 DBM 函数的兼容性库

8.39. Gperf-3.3

Gperf 根据一组键值，生成完美散列函数。

估计构建时间: 0.2 SBU
需要硬盘空间: 12 MB

8.39.1. 安装 Gperf

准备编译 Gperf:

```
./configure --prefix=/usr --docdir=/usr/share/doc/gperf-3.3
```

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.39.2. Gperf 的内容

安装的程序: gperf
安装的目录: /usr/share/doc/gperf-3.3

简要描述

gperf 根据键值集合生成完美散列函数

8.40. Expat-2.7.1

Expat 软件包包含用于解析 XML 文件的面向流的 C 语言库。

估计构建时间: 0.1 SBU
需要硬盘空间: 14 MB

8.40.1. 安装 Expat

准备编译 Expat:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/expat-2.7.1
```

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

如果需要，安装该软件包的文档:

```
install -v -m644 doc/*.{html,css} /usr/share/doc/expat-2.7.1
```

8.40.2. Expat 的内容

安装的程序: xmlwf
安装的库: libexpat.so
安装的目录: /usr/share/doc/expat-2.7.1

简要描述

xmlwf 是一个用于检验 XML 文档是否良好的非验证性工具

`libexpat` 包含解析 XML 文档的 API 函数

8.41. Inetutils-2.6

Inetutils 软件包包含基本网络程序。

估计构建时间: 0.3 SBU
需要硬盘空间: 36 MB

8.41.1. 安装 Inetutils

首先，使得该软件包能够用 gcc-14.1 或更新版本构建:

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

准备编译 Inetutils:

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

配置选项的含义:

`--disable-logger`
该选项防止 Inetutils 安装 **logger** 程序，它被脚本文件用于向系统日志守护程序传递消息。这里不安装它，因为 Util-linux 会安装更新的版本。

`--disable-whois`
该选项防止构建过时的 **whois** 客户端，BLFS 手册中有一个更好的 **whois** 客户端。

`--disable-r*`
这些参数禁用过时的程序，由于安全问题，它们不应该被继续使用。它们提供的功能可以由 BLFS 手册中的 openssh 软件包代替。

`--disable-servers`
该选项禁用 Inetutils 软件包包含的若干网络服务程序，它们在基本的 LFS 系统中注定是不合适的。其中一些服务程序从本质上就不安全，只有在可信的网络环境中才能被认为是安全的。要注意的是，对于其中许多服务程序，都能找到更好的替代品。

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

已知名为 `libls.sh` 的一项测试有时会失败。

安装该软件包:

```
make install
```

将一个程序移动到正确的位置:

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

8.41.2. Inetutils 的内容

安装的程序: dnsdomainname, ftp, ifconfig, hostname, ping, ping6, talk, telnet, tftp, 以及 traceroute

简要描述

dnsdomainname 显示系统的 DNS 域名

ftp 文件传输程序

hostname 报告或设定主机名称

ifconfig 管理网络接口

ping 发送回显请求数据包并报告响应用时

ping6 用于 IPv6 网络的 **ping** 版本

talk 用于和其他用户聊天

telnet TELNET 协议的接口

tftp 简单文件传输程序

traceroute 追踪您的数据包从您工作的主机到网络上另一台主机的路径，显示中间通过的跳跃(网关)

8.42. Less-679

Less 软件包包含一个文本文件查看器。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 16 MB

8.42.1. 安装 Less

准备编译 Less:

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

配置选项的含义:

`--sysconfdir=/etc`
该选项告诉该软件包创建的程序在 `/etc` 查找配置文件

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.42.2. Less 的内容

安装的程序: less, lessecho, 以及 lesskey

简要描述

less 一个文件查看器或分页器；它显示给定文件的内容，使得用户可以进行滚动、查找字符串，或跳到标记

lessecho 用于展开元字符，例如 Unix 系统上文件名中的 * 和 ?

lesskey 用于指定 **less** 的按键绑定

8.43. Perl-5.42.0

Perl 软件包包含实用报表提取语言。

估计构建时间: 1.3 SBU
需要硬盘空间: 257 MB

8.43.1. 安装 Perl

该版本的 Perl 会构建 Compress::Raw::ZLib 和 Compress::Raw::BZip2 模块。默认情况下 Perl 会使用内部的源码副本构建它们。执行以下命令，使得 Perl 使用已经安装到系统上的库:

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

为了能够完全控制 Perl 的设置，您可以在以下命令中移除 “-des” 选项，并手动选择构建该软件包的方式。或者，直接使用下面给出的命令，以使用 Perl 自动检测的默认值:

```
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
```

新出现的配置选项的含义:

`-D pager="/usr/bin/less -isR"`
这保证该软件包使用 **less** 对输出进行分页，而不是使用 **more**。

`-D man1dir=/usr/share/man/man1 -D man3dir=/usr/share/man/man3`
由于 Groff 还没有安装，**Configure** 不会创建 Perl 的手册页。这些参数覆盖这个默认行为。

`-D usethreads`
构建带有线程支持的 Perl。

编译该软件包:

```
make
```

运行命令以测试编译结果:

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

安装该软件包，并清理环境变量:

```
make install
unset BUILD_ZLIB BUILD_BZIP2
```

8.43.2. Perl 的内容

安装的程序: corelist, cpan, enc2xs, encguess, h2ph, h2xs, instmodsh, json_pp, libnetcfg, perl, perl5.42.0 (指向 perl 的硬链接), perlbug, perldoc, perlivp, perlthanks (指向 perlbug 的硬链接), piconv, pl2pm, pod2html, pod2man, pod2text, pod2usage, podchecker, podselect, prove, ptar, ptardiff, ptargrep, shasum, splain, xsubpp, 以及 zipdetails
安装的库: 很多，无法在这里全部列出
安装的目录: /usr/lib/perl5

简要描述

corelist	Module::CoreList 的命令行前端
cpan	通过命令行与 Perl 综合归档网络 (CPAN) 交互
enc2xs	从 Unicode 字符映射或 Tcl 编码文件构建 Encode 模块使用的 Perl 扩展
encguess	猜测一些文件的编码格式
h2ph	将 `.h` C 头文件转化为 `.ph` Perl 头文件
h2xs	将 `.h` C 头文件转化为 Perl 扩展
instmodsh	用于检验安装好的 Perl 模块的 shell 脚本；可以从安装好的模块创建压缩包
json_pp	在特定输入输出格式之间转化数据
libnetcfg	可以被用于配置 `libnet` Perl 模块
perl	由 C 语言、**sed**、**awk** 和 **sh** 的最好特性结合成的一门瑞士军刀式语言
perl5.42.0	指向 **perl** 的硬链接
perlbug	用于创建关于 Perl 或者它附带的模块的 bug 报告，并用邮件发送它们
perldoc	显示集成在 Perl 安装目录树或某个 Perl 脚本中的一页 pod 格式文档
perlivp	Perl 安装检验程序；它可以被用于确认 Perl 和它的库都安装正确
perlthanks	用于生成发送给 Perl 开发者的感谢信
piconv	字符编码转换器 **iconv** 的 Perl 版本
pl2pm	一个用于将 Perl4 `.pl` 文件转换成 Perl5 `.pm` 模块的粗糙工具
pod2html	将 pod 格式的文件转换为 HTML 格式
pod2man	将 pod 数据转换为格式化的 *roff 输入
pod2text	将 pod 数据转化为格式化的 ASCII 文本
pod2usage	输出文件中嵌入的 pod 文档中的使用方法信息
podchecker	检查 pod 格式文档文件的语法
podselect	显示 pod 文档中的指定章节
prove	用于运行使用 Test::Harness 模块的测试
ptar	一个 Perl 编写的类似 **tar** 的程序
ptardiff	一个比较压缩档案和未压缩版本的 Perl 程序
ptargrep	一个在 tar 档案中的文件内容上进行模式匹配的 Perl 程序
shasum	打印或检查 SHA 校验和
splain	被用于 Perl 的强制性详细警告诊断
xsubpp	将 Perl XS 代码转换为 C 代码
zipdetails	显示 Zip 文件内部结构的详细信息

8.44. XML::Parser-2.47

XML::Parser 模块是 James Clark 的 XML 解析器 Expat 的 Perl 接口。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 2.4 MB

8.44.1. 安装 XML::Parser

准备编译 XML::Parser:

```
perl Makefile.PL
```

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make test
```

安装该软件包:

```
make install
```

8.44.2. XML::Parser 的内容

安装的模块: Expat.so

简要描述

`Expat` 提供 Expat 的 Perl 接口

8.45. Intltool-0.51.0

Intltool 是一个从源代码文件中提取可翻译字符串的国际化工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 1.5 MB

8.45.1. 安装 Intltool

首先修复由 perl-5.22 及更新版本导致的警告:

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

注意

上面使用的正则表达式由于大量的反斜线，看上去比较奇怪。它的功能是在序列 '\${' 的花括号之前增加一个反斜线，得到 '\$\{'。

准备编译 Intltool:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

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

8.45.2. Intltool 的内容

安装的程序: intltool-extract, intltool-merge, intltool-prepare, intltool-update, 以及 intltoolize
安装的目录: /usr/share/doc/intltool-0.51.0 和 /usr/share/intltool

简要描述

intltoolize	准备为软件包应用 intltool
intltool-extract	生成可供 **gettext** 读取的头文件
intltool-merge	将翻译好的字符串合并为多种类型的文件
intltool-prepare	更新 pot 文件并将它们和翻译文件合并
intltool-update	更新 po 模板文件并将它们和翻译文件合并

8.46. Autoconf-2.72

Autoconf 软件包包含生成能自动配置软件包的 shell 脚本的程序。

估计构建时间: 不到 0.1 SBU (计入测试时间为约 0.4 SBU)
需要硬盘空间: 25 MB

8.46.1. 安装 Autoconf

准备编译 Autoconf:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.46.2. Autoconf 的内容

安装的程序: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, 以及 ifnames
安装的目录: /usr/share/autoconf

简要描述

autoconf 产生自动配置软件源码包，使其适用于多种类 Unix 系统的 shell 脚本；它产生的脚本可以独立运行 —— 运行它们不需要 **autoconf** 程序

autoheader 一个创建 C #define 预处理指令的模板，以供配置脚本使用的程序

autom4te M4 宏处理器的封装器

autoreconf 在 **autoconf** 和 **automake** 模板文件发生变化时，按照正确顺序自动运行 **autoconf**、**autoheader**、**aclocal**、**automake**、**gettextize**，以及 **libtoolize**，以便节省时间

autoscan 帮助用户为软件包创建`configure.in` 文件；它检验目录树中的源代码文件，在其中找出一般的移植性问题，然后创建一个 `configure.scan` 文件，作为软件包的原始 `configure.in` 文件

autoupdate 修改 `configure.in` 文件，将其中过时的 **autoconf** 宏名改为新的宏名。

ifnames 帮助用户为软件包编写 `configure.in`；它打印软件包在 C 预处理器条件中使用的所有标识符 [如果一个软件包已经被设定为有一定的可移植性，该程序可以帮助确定 **configure** 需要进行哪些测试。它也会填充 **autoscan** 生成的 `configure.in` 中留下的空隙。]

8.47. Automake-1.18.1

Automake 软件包包含自动生成 Makefile，以便和 Autoconf 一同使用的程序。

估计构建时间: 不到 0.1 SBU (计入测试时间为约 1.1 SBU)
需要硬盘空间: 123 MB

8.47.1. 安装 Automake

准备编译 Automake：

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.18.1
```

编译该软件包：

```
make
```

由于测试点内部的时延，使用四个并行任务能提高测试速度，即使系统的逻辑 CPU 核心数较小。输入以下命令测试编译结果：

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

如果不希望使用所有逻辑 CPU 核心，将 `$((...))` 替换成希望使用的逻辑 CPU 核心数。

安装该软件包：

```
make install
```

8.47.2. Automake 的内容

安装的程序: aclocal, aclocal-1.18 (与 aclocal 互为硬链接), automake, 以及 automake-1.18 (与 automake 互为硬链接)
安装的目录: /usr/share/aclocal-1.18, /usr/share/automake-1.18, 以及 /usr/share/doc/automake-1.18.1

简要描述

aclocal 根据 `configure.in` 文件内容生成 `aclocal.m4`

aclocal-1.18 指向 **aclocal** 的硬链接

automake 一个根据 `Makefile.am` 文件，自动生成 `Makefile.in` 文件的工具 [为了创建软件包中的所有 `Makefile.in` 文件，在顶层目录运行该程序。它通过扫描 `configure.in` 文件，找到每个适用的 `Makefile.am` 文件，并生成对应的 `Makefile.in` 文件。]

automake-1.18 指向 **automake** 的硬链接

8.48. OpenSSL-3.5.2

OpenSSL 软件包包含密码学相关的管理工具和库。它们被用于向其他软件包提供密码学功能，例如 OpenSSH，电子邮件程序和 Web 浏览器 (以访问 HTTPS 站点)。

估计构建时间: 1.9 SBU
需要硬盘空间: 1.1 GB

8.48.1. 安装 OpenSSL

准备编译 OpenSSL:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

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

一项名为 30-test_afalg.t 的测试在宿主内核未启用 CONFIG_CRYPTO_USER_API_SKCIPHER，或未启用任何提供 AES-CBC 模式加密实现的选项 (例如，CONFIG_CRYPTO_AES 和 CONFIG_CRYPTO_CBC 的组合，或者 CONFIG_CRYPTO_AES_NI_INTEL —— 如果 CPU 支持 AES-NI)。如果该测试失败，可以安全地忽略它。

安装该软件包:

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

将版本号添加到文档目录名，以和其他软件包保持一致:

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-3.5.2
```

如果需要的话，安装一些额外的文档:

```
cp -vfr doc/* /usr/share/doc/openssl-3.5.2
```

注意

一旦新版的 OpenSSL 被发布，且它包含对安全缺陷的修复，就应该更新 OpenSSL。从 OpenSSL 3.0.0 起，OpenSSL 的版本号使用主版本号.次版本号.修订号的格式。主版本号相同的版本更新保证 API 与 ABI 的兼容性。由于 LFS 只安装共享库，没有必要重新编译链接到 libcrypto.so 或者 libssl.so 的软件包，前提是更新前后主版本号不变。

然而，需要重新启动链接到这两个库的，正在运行的程序。详见第 8.2.1 节 “升级问题”中的相关条目。

8.48.2. OpenSSL 的内容

安装的程序: c_rehash 和 openssl
安装的库: libcrypto.so 和 libssl.so
安装的目录: /etc/ssl，/usr/include/openssl，/usr/lib/engines 以及 /usr/share/doc/openssl-3.5.2

简要描述

c_rehash 一个 Perl 脚本，扫描一个目录中的所有文件，并添加它们的符号链接，符号链接名为对应文件的散列值。**c_rehash** 已弃用，应该使用 **openssl rehash** 命令替代它

openssl 一个命令行工具，用于从 shell 使用 OpenSSL的密码学库的一些密码学函数。它可以被用于 openssl(1) 描述的许多功能

`libcrypto.so` 实现不同 Internet 标准使用的许多密码学算法。该库提供的服务被 OpenSSL 的 SSL、TLS 和 S/MIME 实现使用，也被用于实现 OpenSSH、OpenPGP，以及其他密码学标准

`libssl.so` 实现传输层安全 (TLS v1) 协议。它提供了丰富的 API，这些 API 的文档可以在 ssl(7) 中查阅

8.49. Elfutils-0.193 中的 Libelf

Libelf 是一个处理 ELF (可执行和可链接格式) 文件的库。

估计构建时间: 0.3 SBU
需要硬盘空间: 156 MB

8.49.1. 安装 Libelf

Libelf 是 elfutils-0.193 软件包的一部分。请使用 elfutils-0.193.tar.bz2 作为源代码包。

准备编译 Libelf:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

已知 dwarf_srclang_check 和 run-backtrace-native-core.sh 这两项测试可能失败。

只安装 Libelf:

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

8.49.2. Libelf 的内容

安装的库: libelf.so
安装的目录: /usr/include/elfutils

简要描述

`libelf.so` 包含处理 ELF 目标文件的 API 函数

8.50. Libffi-3.5.2

Libffi 库提供一个可移植的高级编程接口，用于处理不同调用惯例。这允许程序在运行时调用任何给定了调用接口的函数。

FFI 是 Foreign Function Interface (跨语言函数接口) 的缩写。FFI 允许使用某种编程语言编写的程序调用其他语言编写的程序。特别地，Libffi 为 Perl 或 Python 等解释器提供使用 C 或 C++ 编写的共享库中子程序的能力。

估计构建时间: 1.7 SBU
需要硬盘空间: 10 MB

8.50.1. 安装 Libffi

注意

和 GMP 类似，Libffi 在构建时会使用特定于当前处理器的优化。如果是在为另一台计算机构建系统，请将 `--with-gcc-arch=` 的设定值改为当前计算机和那一台计算机的 CPU **都**完全实现的某个架构名称。否则，所有链接到 `libffi` 的程序都可能触发非法指令异常。如果无法找出对两种 CPU 都合适的值，将该选项改为 `--without-gcc-arch` 可得到通用的库。

准备编译 Libffi:

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

配置选项的含义:

`--with-gcc-arch=native`
保证 gcc 为当前系统进行优化。如果不使用该选项，构建系统会猜测系统架构，可能生成不正确的代码。如果要将生成的代码从本地系统复制到指令集功能较弱的系统中，需要使用目标系统架构作为该选项的参数值。关于不同系统架构的信息，参阅 gcc 手册中提供的的 x86 选项。

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.50.2. Libffi 的内容

安装的库: libffi.so

简要描述

`libffi` 包含跨语言函数接口 API 函数

8.51. Python-3.13.7

Python 3 软件包包含 Python 开发环境。它被用于面向对象编程，编写脚本，为大型程序建立原型，或者开发完整的应用。Python 是一种解释性的计算机语言。

估计构建时间: 2.0 SBU
需要硬盘空间: 453 MB

8.51.1. 安装 Python 3

准备编译 Python:

```
./configure --prefix=/usr          \
            --enable-shared        \
            --with-system-expat    \
            --enable-optimizations \
            --without-static-libpython
```

配置选项的含义:

`--with-system-expat`
该选项允许链接到系统已经安装的 Expat。

`--enable-optimizations`
该选项启用有力，但是消耗时间的优化步骤。它会两次构建解释器；在初次构建结果上运行测试，并使用测试结果优化最终构建版本的性能。

编译该软件包:

```
make
```

已知一些测试偶尔会陷入无限等待状态。因此如果需要测试编译结果，运行测试套件，但是将每个单项测试的最长运行时间限制为 2 分钟:

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

对于较慢的系统，您可能需要放宽运行时间限制，1 SBU (通过仅用单个 CPU 核心构建第一遍的 Binutils 测得) 一般来说是足够的。一些测试并不精确，因此测试套件会自动重新运行失败的测试。如果某项测试失败后在重新运行时能够通过，则应该将其视为正常通过。已知一项名为 test_ssl 的测试在 chroot 环境中会失败。

安装该软件包:

```
make install
```

在本书中，我们以 `root` 用户身份，使用 **pip3** 命令，为所有用户安装 Python 3 程序和模块。这和 Python 开发者的建议不同：他们认为应该将这些包安装到虚拟环境中，或者某个用户的主目录中 (即，以该用户身份执行 **pip3** 命令)。只要使用 `root` 用户身份执行 **pip3** 命令，它就会输出若干行警告信息。

这项建议主要是为了避免和系统包管理器 (如 **dpkg**) 的冲突。LFS 没有系统包管理器，所以不存在这个问题。另外，**pip3** 在运行时，会检查它的最新版本。由于 LFS chroot 环境中没有配置域名解析，**pip3** 无法检查它的最新版本，也会输出一条警告信息。

在引导 LFS 系统并配置网络连接后，则会出现一条不同的警告信息，指示用户使用 PyPI 提供的，预先构建的 wheel 档案更新 **pip3** (如果有更新的版本)。但是对于 LFS，我们认为 **pip3** 是 Python 3 的一部分，因此不应单独升级它。另外，使用预先构建的 wheel 包也会偏离我们的目标：从源代码构建 Linux 系统。因此，应该忽略关于 **pip3** 新版本的警告。如果您不想看到这些警告，可以执行以下命令，创建配置文件，以禁止这些警告:

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

重要

在 LFS 和 BLFS 中，我们通常用 **pip3** 命令构建和安装 Python 模块。请注意两份手册中的 **pip3 install** 命令都应该以 `root` 身份运行 (除非这条命令是为 Python 虚拟环境进行安装)。以非 `root` 用户身份运行 **pip3 install** 命令可能看似正常工作，但这会导致其他用户无法访问安装的模块。

pip3 install 不会自动重新安装已经安装好的模块。如果需要用 **pip3 install** 命令升级模块 (例如，从 meson-0.61.3 升级到 meson-0.62.0)，则需要在命令行中加入 `--upgrade` 选项。如果由于某种原因，确实需要降级某个模块，或重新安装某个已安装的版本，需要在命令行中加入 `--force-reinstall --no-deps` 选项。

如果需要的话，安装预先格式化的文档：

```
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
```

文档安装命令的含义：

`--no-same-owner` 和 `--no-same-permissions`
保证安装的文件拥有正确的所有者和权限模式。否则，tar 会保留上游开发者使用的用户所有者和权限模式。

8.51.2. Python 3 的内容

安装的程序: 2to3, idle3, pip3, pydoc3, python3, 以及 python3-config
安装的库: libpython3.13.so 和 libpython3.so
安装的目录: /usr/include/python3.13, /usr/lib/python3 以及 /usr/share/doc/python-3.13.7

简要描述

2to3 是一个 Python 程序，读取 Python 2.x 源代码并对它进行一系列修正，转换成合法的 Python 3.x 源代码

idle3 一个封装脚本，启动支持 Python 语法的 GUI 文本编辑器。要运行这个脚本，必须在 Python 之前安装 Tk，从而构建 Tkinter Python 模块。

pip3 Python 包安装器。您可以使用 pip 安装来自 Python 软件包目录或其他目录的包。

pydoc3 是 Python 文档工具

python3 是 Python 的解释器，Python 是一种解释性的、交互的、面向对象的程序设计语言

8.52. Flit-Core-3.12.0

Flit-core 是 Flit (一个用于简单的 Python 模块的打包工具) 中用于为发行版进行构建的组件。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 1.3 MB

8.52.1. 安装 Flit-Core

构建该软件包:

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

安装该软件包:

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

pip3 配置选项和命令的含义:

wheel
该命令构建该软件包的 wheel 档案。

`-w dist`
指示 pip 将生成的 wheel 包置入 `dist` 目录。

`--no-cache-dir`
防止 pip 将生成的 wheel 包复制到 `/root/.cache/pip` 目录。

install
该命令安装该软件包。

`--no-build-isolation`, `--no-deps`, 以及 `--no-index`
这些选项防止 pip 从在线软件包仓库 (PyPI) 获取文件。如果按正确的顺序安装软件包，则 pip 完全不会尝试获取文件；但是在用户不小心犯下错误时，这些选项可以作为保险措施。

`--find-links dist`
指示 pip 在 `dist` 目录中搜索 wheel 档案。

8.52.2. Flit-Core 的内容

安装的目录: /usr/lib/python3.13/site-packages/flit_core 和 /usr/lib/python3.13/site-packages/flit_core-3.12.0.dist-info

8.53. Packaging-25.0

Packaging 模块是一个 Python 工具库，此库提供了实现互操作性规范的实用工具，这些规范具有明确且唯一正确的行 (例如 PEP440) 或从单一共享实现中获益良多 (例如 PEP 425)。这些工具处理 Python 包的版本，指定符，标记，需求，标签，以及依赖等属性。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 3.3 MB

8.53.1. 安装 Packaging

执行以下命令编译 packaging:

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

使用以下命令安装 packaging:

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

8.53.2. Packaging 的内容

安装的目录: /usr/lib/python3.13/site-packages/packaging and /usr/lib/python3.13/site-packages/packaging-25.0.dist-info

8.54. Wheel-0.46.1

Wheel 是 Python wheel 软件包标准格式的参考实现。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 608 KB

8.54.1. 安装 Wheel

执行以下命令编译 Wheel:

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

使用以下命令安装 Wheel:

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

8.54.2. Wheel 的内容

安装的程序: wheel
安装的目录: /usr/lib/python3.13/site-packages/wheel 和 /usr/lib/python3.13/site-packages/wheel-0.46.1.dist-info

简要描述

wheel 是用于解包、包装，或者转换 wheel 档案的工具

8.55. Setuptools-80.9.0

Setuptools 是一个用于下载、构建、安装、升级，以及卸载 Python 软件包的工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 25 MB

8.55.1. 安装 Setuptools

构建该软件包:

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

安装该软件包:

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

8.55.2. Setuptools 的内容

安装的目录: /usr/lib/python3.13/site-packages/_distutils_hack, /usr/lib/python3.13/site-packages/pkg_resources, /usr/lib/python3.13/site-packages/setuptools, 以及 /usr/lib/python3.13/site-packages/setuptools-80.9.0.dist-info

8.56. Ninja-1.13.1

Ninja 是一个注重速度的小型构建系统。

估计构建时间: 0.2 SBU
需要硬盘空间: 43 MB

8.56.1. 安装 Ninja

在运行时，**ninja** 通常并行运行尽可能多的进程。默认情况下最大进程数是系统 CPU 核心数加 2 得到的值。这可能导致 CPU 过热，或者耗尽系统内存。如果使用命令行执行 ninja，可以传递 -jN 参数以限制并行进程数。但是，某些软件包内嵌了 ninja 的执行过程，且并不传递 -j 参数。

应用下面这个可选的修改，用户即可通过一个环境变量 NINJAJOBS 限制并行进程数量。**例如**设置：

```
export NINJAJOBS=4
```

会限制 **ninja** 使用 4 个并行进程。

如果希望 **ninja** 能够识别环境变量 NINJAJOBS，使用流编辑器，添加这一功能：

```
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
```

构建 Ninja：

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

构建选项的含义：

`--bootstrap`
这个参数强制 Ninja 为当前系统重新构建自身。

`--verbose`
该选项使得 **configure.py** 显示构建 Ninja 的进度。

测试套件无法在 chroot 环境中运行。它需要 cmake。不过，重新构建该软件包本身 (由 `--bootstrap` 选项要求) 已经测试了它的基本功能。

安装该软件包：

```
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
```

8.56.2. Ninja 的内容

安装的程序: ninja

简要描述

ninja 是 ninja 构建系统

8.57. Meson-1.8.3

Meson 是一个开放源代码构建系统，它的设计保证了非常快的执行速度，和尽可能高的用户友好性。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 45 MB

8.57.1. 安装 Meson

执行以下命令编译 Meson：

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

测试套件需要一些超过 LFS 覆盖范围的软件包。

安装该软件包：

```
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
```

安装选项的含义：

`-w dist`
将生成的 wheel 包置入 `dist` 目录。

`--find-links dist`
从 `dist` 目录安装 wheel 包。

8.57.2. Meson 的内容

安装的程序: meson
安装的目录: /usr/lib/python3.13/site-packages/meson-1.8.3.dist-info 和 /usr/lib/python3.13/site-packages/mesonbuild

简要描述

meson 一个高产出的构建系统

8.58. Kmod-34.2

Kmod 软件包包含用于加载内核模块的库和工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 6.7 MB

8.58.1. 安装 Kmod

准备编译 Kmod:

```
mkdir -p build
cd       build

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

配置选项的含义:

`-D manpages=false`
该选项禁止生成手册页，因为生成手册页需要不属于 LFS 的程序。

编译该软件包:

```
ninja
```

该软件包的测试套件需要内核的原始头文件 (不是之前安装的 “净化的” 内核头文件)，原始头文件超出了 LFS 的范畴。

现在安装该软件包:

```
ninja install
```

8.58.2. Kmod 的内容

安装的程序: depmod (到 kmod 的链接), insmod (到 kmod 的链接), kmod, lsmod (到 kmod 的链接), modinfo (到 kmod 的链接), modprobe (到 kmod 的链接), 以及 rmmod (到 kmod 的链接)
安装的库: libkmod.so

简要描述

depmod 根据现有模块的符号信息创建依赖关系文件；**modprobe** 使用依赖关系文件自动加载需要的模块
insmod 在正在运行的内核中安装可加载模块
kmod 加载或卸载内核模块
lsmod 列出当前加载的模块
modinfo 检验与某个内核模块相关的目标文件，打印它能够收集到的一切信息
modprobe 使用一个 **depmod** 创建的依赖关系文件，自动加载相关模块
rmmod 从正在运行的内核中卸载模块
`libkmod` 这个库被其他程序用于加载和卸载内核模块

8.59. Coreutils-9.7

Coreutils 软件包包含各种操作系统都需要提供的基本工具程序。

估计构建时间: 1.2 SBU
需要硬盘空间: 180 MB

8.59.1. 安装 Coreutils

首先，应用上游给出的补丁以修复一项安全问题：

```
patch -Np1 -i ../coreutils-9.7-upstream_fix-1.patch
```

POSIX 要求 Coreutils 中的程序即使在多字节 locale 中也能正确识别字符边界。下面应用一个补丁，以解决 Coreutils 不满足该要求的问题，并修复其他一些国际化相关的 bug：

```
patch -Np1 -i ../coreutils-9.7-i18n-1.patch
```

注意

在之前，这个补丁中找出了许多 bug。在向 Coreutils 维护者报告新 bug 前，请检查这些 bug 在不使用该补丁的情况下是否还会重现。

现在准备编译 Coreutils：

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

命令和配置选项的含义：

autoreconf -fv
国际化补丁更改了构建系统，因此需要重新生成配置文件。一般来说我们会使用 `-i` 选项以更新通用的辅助文件，但该软件包的 `configure.ac` 指定了过时的 gettext 版本，因此该选项无法正常工作。

automake -af
由于未使用 `-i` 选项，automake 使用的辅助文件未被 **autoreconf** 更新。该命令直接更新它们，以防止构建失败。

`FORCE_UNSAFE_CONFIGURE=1`
该环境变量允许以 `root` 用户身份构建该软件包。

`--enable-no-install-program=kill,uptime`
这个开关用于防止 Coreutils 安装那些被其他软件包安装的二进制程序。

编译该软件包：

```
make
```

如果不运行测试套件，直接跳到 “安装该软件包”。

现在测试套件已经可以运行了。首先运行那些设计为由 `root` 用户运行的测试：

```
make NON_ROOT_USERNAME=tester check-root
```

之后我们要以 `tester` 用户身份运行其余测试。然而，某些测试要求测试用户属于至少一个组。为了不跳过这些测试，我们添加一个临时组，并使得 `tester` 用户成为它的成员：

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

修正访问权限，使得非 root 用户可以编译和运行测试：

```
chown -R tester .
```

现在运行测试 (使用 /dev/null 作为标准输入，否则在图形终端，SSH 会话，或者 GNU Screen 构建 LFS 时，由于标准输入连接到了宿主发行版的 PTY，而该 PTY 的设备节点无法在 LFS chroot 环境中访问，两项测试无法正常工作)：

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

删除临时组：

```
groupdel dummy
```

安装该软件包：

```
make install
```

将程序移动到 FHS 要求的位置：

```
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
```

8.59.2. Coreutils 的内容

安装的程序: [, 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, sha1sum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami, 以及 yes
安装的库: libstdbuf.so (在 /usr/libexec/coreutils 中)
安装的目录: /usr/libexec/coreutils

简要描述

[	这是一个真实存在的命令，/usr/bin/[；它和 **test** 命令的功能相同
base32	根据 base32 标准 (RFC 4648) 编码和解码数据
base64	根据 base64 标准 (RFC 4648) 编码和解码数据
b2sum	打印或检查 BLAKE2 (512 位) 校验和
basename	从文件名移除所有路径和一个给定后缀
basenc	使用一些算法编码或解码数据
cat	将文件合并到标准输出
chcon	修改文件和目录的 SELinux 安全上下文
chgrp	修改文件和目录所属的组
chmod	修改给定文件的访问权限为指定模式；模式可以是所需修改的符号表示，或新权限的八进制码

chown 修改拥有文件的用户或组
chroot 将给定目录作为 / 目录，运行命令
cksum 输出每个给定文件的循环冗余检查 (CRC) 校验和及字节数
comm 比较两个排好序的文件，将两个文件特有的部分和它们共有的部分显示为三列
cp 复制文件
csplit 将给定文件分割为若干新文件，根据给定模式或行号进行分割，并输出每个新文件的字节数
cut 根据给定的域或位置，打印输入的分节和选定部分
date 以给定格式显示当前日期和时间，或设定系统日期和时间
dd 以给定块大小和个数复制文件，同时可以进行转换
df 报告每个已挂载文件系统 (或包含给定文件的文件系统) 的总大小和可用空间
dir 列出给定目录的内容 (和 **ls** 命令相同)
dircolors 输出用于设定 LS_COLOR 环境变量的命令，以修改 **ls** 的配色方案
dirname 提取给定文件名的目录部分
du 报告当前目录使用的磁盘空间，给出当前目录下所有子目录和文件占用的空间
echo 显示给定字符串
env 在修改的环境中运行命令
expand 将制表符转换成空格
expr 计算表达式
factor 打印给定整数的质因子
false 什么也不做；总是以失败状态码退出
fmt 重新格式化给定文件的段落
fold 折叠给定文件中的行
groups 报告用户所属的组
head 打印文件的前 10 (或给定行数) 行
hostid 以十六进制格式打印主机数字标识符
id 报告当前用户或给定用户的有效用户 ID、组 ID 和所属的组
install 复制文件并设定它们的访问权限，以及 (如果可能) 它们的所有者和属组
join 将两个文件中拥有相同域的行合并
link (以给定文件名) 创建硬链接
ln 在文件之间创建硬链接或软 (符号) 链接
logname 报告当前用户登录名
ls 列出给定目录内容
md5sum 报告或检查消息摘要 5 (MD5) 校验和
mkdir 以给定名称创建目录
mkfifo 以给定名称创建先进先出 (FIFO)，在 UNIX 惯用语中又称为 “命名管道”
mknod 以给定名称创建设备节点；设备节点可能是字符特殊文件、块特殊文件或 FIFO
mktemp 安全地创建临时文件，常用在脚本中
mv 移动或重命名文件或目录

nice 以修改的调度优先级运行程序
nl 标出给定文件的行
nohup 执行命令并使其忽略挂机信号，同时将输出重定向到日志文件
nproc 打印进程可用的处理单元数目
numfmt 在数字和人类可读字符串之间互相转换
od 以八进制或其他格式转储文件
paste 合并给定文件，将它们的对应行连接起来，以制表符分割
pathchk 检查文件名的有效性和可移植性
pinky 是轻量级 finger 客户端，报告给定用户的一些信息
pr 对文件进行分页和分栏以便打印
printenv 打印环境变量
printf 以给定格式打印给定参数，很像 C printf 函数
ptx 用文中的每个关键字，根据给定文件内容生成重排索引
pwd 报告当前工作目录名
readlink 报告给定符号链接的值
realpath 打印解析过的目录
rm 删除文件或目录
rmdir 如果目录是空的，删除它们
runcon 以给定 SELinux 安全上下文运行命令
seq 以给定的范围和增量打印等差数列
sha1sum 打印或检查 160 位安全散列算法 1 (SHA1) 校验和
sha224sum 打印或检查 224 位安全散列算法校验和
sha256sum 打印或检查 256 位安全散列算法校验和
sha384sum 打印或检查 384 位安全散列算法校验和
sha512sum 打印或检查 512 位安全散列算法校验和
shred 将给定文件多次用复杂模式覆盖，增加恢复数据的难度
shuf 打乱文件中的行
sleep 等待给定时间
sort 对给定文件的行进行排序
split 根据大小或行数，将指定文件分割成若干部分
stat 显示文件或文件系统状态
stdbuf 以修改的标准流缓冲操作运行命令
stty 设置或报告终端行设定
sum 打印每个指定文件的校验和及块个数
sync 刷新文件系统缓冲；它将修改过的块强制写入磁盘，并更新超级块
tac 逆序连接给定文件
tail 输出给定文件的最后 10 (或指定行数) 行
tee 读取标准输入，并将内容同时写入标准输出和给定文件

test	比较两个值，或检查文件类型
timeout	在限定时间内运行命令
touch	修改文件时间戳，将每个给定文件的访问和修改时间设为当前时间； 以零长度创建当前不存在的文件
tr	从标准输入变换、压缩或删除给定字符
true	什么也不做；总是以成功状态码退出
truncate	将文件截断或扩展到指定大小
tsort	进行拓扑排序；根据给定文件的部分顺序信息输出完整的排序列表
tty	报告标准输入的终端文件名
uname	报告系统信息
unexpand	将空格转换成制表符
uniq	在连续的相同行中只保留一行，删除其他所有行
unlink	删除给定文件
users	报告当前登录系统的用户名
vdir	和 **ls -l** 相同
wc	报告给定文件的行数、单词数和字节数，如果给定了多个文件，还会输出这些统计值的总和
who	报告当前登录的用户
whoami	报告与当前有效用户 ID 相关的用户名
yes	不停输出 `y`，或给定的字符串，直到被杀死
`libstdbuf`	**stdbuf** 使用的库

8.60. Diffutils-3.12

Diffutils 软件包包含显示文件或目录之间差异的程序。

估计构建时间: 0.5 SBU
需要硬盘空间: 51 MB

8.60.1. 安装 Diffutils

准备编译 Diffutils:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.60.2. Diffutils 的内容

安装的程序: cmp, diff, diff3, 以及 sdiff

简要描述

cmp 逐字节比较两个文件并报告它们的不同之处
diff 比较两个文件或目录，并报告文件中哪些行不同
diff3 逐行比较三个文件
sdiff 合并两个文件，并交互地输出结果

8.61. Gawk-5.3.2

Gawk 软件包包含操作文本文件的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 45 MB

8.61.1. 安装 Gawk

首先，确保不安装某些不需要的文件：

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

准备编译 Gawk：

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

编译该软件包：

```
make
```

运行命令以测试编译结果：

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

安装该软件包：

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

命令的含义：

rm -f /usr/bin/gawk-5.3.2
如果硬链接 `gawk-5.3.2` 已经存在，则构建系统不会更新它。移除该链接以保证之前在第 6.9 节“Gawk-5.3.2”中安装的硬链接被正确更新。

安装过程已经将 **awk** 创建为符号链接，将其手册页也创建为符号链接：

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

如果需要，安装该软件包的文档：

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

8.61.2. Gawk 的内容

安装的程序: awk (到 gawk 的链接), gawk, 以及 gawk-5.3.2
安装的库: filefuncs.so, fnmatch.so, fork.so, inplace.so, intdiv.so, ordchr.so, readdir.so, readfile.so, revoutput.so, revtwoway.so, rwarray.so, 以及 time.so (均位于 /usr/lib/gawk 中)
安装的目录: /usr/lib/gawk, /usr/libexec/awk, /usr/share/awk, 以及 /usr/share/doc/gawk-5.3.2

简要描述

awk	到 **gawk** 的链接
gawk	一个操作文本文件的程序；是 **awk** 的 GNU 实现
gawk-5.3.2	与 **gawk** 互为硬链接

8.62. Findutils-4.10.0

Findutils 软件包包含用于查找文件的程序。这些程序能直接搜索目录树中的所有文件，也可以创建、维护和搜索文件数据库 (一般比递归搜索快，但在数据库最近没有更新时不可靠)。Findutils 还提供了 **xargs** 程序，它能够对一次搜索列出的所有文件执行给定的命令。

估计构建时间: 0.7 SBU
需要硬盘空间: 61 MB

8.62.1. 安装 Findutils

准备编译 Findutils:

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

配置选项的含义:

`--localstatedir`
该选项将 **locate** 数据库移动到 `/var/lib/locate`，以与 FHS 兼容。

编译该软件包:

```
make
```

运行命令以测试编译结果:

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

安装该软件包:

```
make install
```

8.62.2. Findutils 的内容

安装的程序: find, locate, updatedb, 以及 xargs
安装的目录: /var/lib/locate

简要描述

find	在给定目录树中搜索满足给定条件的文件
locate	在文件名数据库中进行搜索，报告包含特定字符串或匹配特定模式的文件名
updatedb	更新 **locate** 数据库；它扫描整个文件系统 (包括当前挂载的其他文件系统，除非被告知不这样做)，并把找到的所有文件名加入数据库
xargs	可以将给定命令作用于一个列表中的所有文件

8.63. Groff-1.23.0

Groff 软件包包含处理和格式化文本和图像的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 108 MB

8.63.1. 安装 Groff

Groff 期望环境变量 PAGE 包含默认纸张大小。对于美国用户来说，PAGE=letter 是正确的。对于其他地方的用户，PAGE=A4 可能更好。尽管在编译时配置了默认纸张大小，可以通过写入 “A4” 或 “letter” 到 /etc/papersize 文件，覆盖默认值。

准备编译 Groff:

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

构建该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.63.2. Groff 的内容

安装的程序: 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, 以及 troff
安装的目录: /usr/lib/groff, /usr/share/doc/groff-1.23.0, 以及 /usr/share/groff

简要描述

addftinfo 读取 troff 字体文件并为其添加 **groff** 系统使用的一些额外字体规格信息
afmtodit 创建供 **groff** 和 **grops** 使用的字体文件
chem 产生化学结构式的 groff 预处理器
eqn 将 troff 输入文件中嵌入的公式描述编译成 **troff** 理解的命令
eqn2graph 将 troff EQN (公式) 转换成裁减好的图像
gdiffmk 标出 groff/nroff/troff 文件的区别
glilypond 将 lilypond 语言写成的乐谱转换为 groff 语言
gperl groff 预处理器，允许在 groff 文件中插入 perl 代码
gpinyin groff 的预处理器，允许在 groff 文件中增加汉语拼音
grap2graph 将 grap 程序文件转换成裁切好的位图 (grap 是一门古老的，用于创建示意图的 Unix 编程语言)。
grn 用于 gremlin 文件的**groff** 预处理器
grodvi **groff** 的驱动程序，生成 TeX dvi 输出文件

groff	groff 文档格式化系统的前端；一般来说，它运行 **troff** 程序和一个适用于选定设备的后处理器
groffer	在 X 和 tty 终端显示 groff 文件和手册页
grog	读取文件，并猜测 **groff** 选项 `-e`, `-man`, `-me`, `-mm`, `-ms`, `-p`, `-s`, 以及 `-t` 中哪一个在打印文件时是必要的，并报告包含这些选项的 **groff** 命令
grolbp	是一个用于 Canon CAPSL 打印机 (LBP-4 和 LBP-8 系列激光打印机) 的 **groff** 驱动程序
grolj4	是一个生成用于 HP LaserJet 4 打印机的 PCL5 格式的 **groff** 驱动程序
gropdf	将 **troff** 输出转换成 PDF
grops	将 **troff** 输出转换成 PostScript
grotty	将 **troff** 输出转换成用于打字机类设备的形式
hpftodit	根据 HP 标签的字体规格文件，创建用于 **groff -Tlj4**的字体文件
indxbib	创建用于给定文件文献数据库的反向索引，以供 **refer**、**lookbib** 以及 **lkbib** 使用
lkbib	在文献数据库中搜索包含指定关键字的引用，并报告找到的所有引用
lookbib	在标准错误输出上显示命令提示符 (除非标准输入不是终端)，读取包含一组关键字的行，在给定文件的文献数据库中搜索包含这些关键字的引用，将它们打印到标准输出，重复这一过程直到输入结束
mmroff	**groff** 的简单预处理器
neqn	将公式格式化为美国标准信息交换代码 (ASCII) 输出
nroff	一个使用 **groff** 仿真 **nroff** 命令的脚本
pdfmom	一个 groff 包装器，提供从 mom 宏包编码的文件转换为 PDF 文档的功能
pdfroff	用 groff 创建 PDF 文档
pfbtops	将 `.pfb` 格式的 PostScript 字体转换为 ASCII
pic	将 troff 或 TeX 输入文件中嵌入的图片描述编译成 TeX 或 **troff** 理解的命令
pic2graph	将 PIC 图示转换成裁切好的图像
post-grohtml	将 GNU **troff** 的输出翻译成 HTML
preconv	将输入文件的编码转换成 GNU **troff** 理解的格式
pre-grohtml	将 GNU **troff** 的输出翻译成 HTML
refer	将文件内容复制到标准输出，除了在 .[和 .] 之间的行被解释为文献引用，.R1 和 .R2 之间的行被解释为处理文献引用的方式
roff2dvi	将 roff 文件转换成 DVI 格式
roff2html	将 roff 文件转换成 HTML 格式
roff2pdf	将 roff 文件转换成 PDF
roff2ps	将 roff 文件转换成 ps 文件
roff2text	将 roff 文件转换成文本文件
roff2x	将 roff 文件转换成其他格式
soelim	读取文件，将 .so 文件 形式的行替换为该文件的内容
tbl	将 troff 输入中嵌入的表格描述编译成 **troff** 理解的命令
tfmtodit	创建用于 **groff -Tdvi** 的字体文件
troff	和 UNIX **troff** 高度兼容；它应该由 **groff** 命令调用，后者也会以正确的顺序和选项运行预处理器和后处理器

8.64. GRUB-2.12

GRUB 软件包包含 “大统一” (GRand Unified) 启动引导器。

估计构建时间: 0.3 SBU
需要硬盘空间: 166 MB

8.64.1. 安装 GRUB

注意

如果您的系统支持 UEFI，且您希望通过 UEFI 引导 LFS，您需要按照 BLFS 页面中的说明，安装支持 UEFI 的 GRUB (及其依赖项)。您可以跳过该软件包，或同时安装该软件包和 BLFS 中为 UEFI 提供的 GRUB 包，并使它们互不干扰 (BLFS 页面提供了这两种方案分别所需的操作)。

警告

移除所有可能影响构建的环境变量:

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

不要尝试使用自定义的编译选项“优化”该软件包。该软件包是一个引导加载器，其源代码中的低级操作可能被一些激进的优化所破坏。

补充源码包发布时缺失的一个文件:

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

准备编译 GRUB:

```
./configure --prefix=/usr     \
            --sysconfdir=/etc \
            --disable-efiemu  \
            --disable-werror
```

新的配置选项的含义:

`--disable-werror`
该选项允许在有较新的 Flex 版本导致的警告时完成构建。

`--disable-efiemu`
该选项通过禁用 LFS 不需要的特性和测试程序，最小化需要构建的内容。

编译该软件包:

```
make
```

不推荐运行该软件包的测试套件。许多测试依赖于在 LFS 的有限环境中不存在的软件包。如果一定要进行测试，运行 **make check**。

安装该软件包，并将 Bash 自动补全支持文件移动到 Bash 自动补全维护者建议的位置:

```
make install
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

使用 GRUB 引导您的 LFS 系统的方法将在第 10.4 节 “使用 GRUB 设定引导过程”中讨论。

8.64.2. GRUB 的内容

安装的程序: 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, 以及 grub-syslinux2cfg

安装的目录: /usr/lib/grub, /etc/grub.d, /usr/share/grub, 以及 /boot/grub (在初次运行 grub-install 时安装)

简要描述

grub-bios-setup	是 grub-install 使用的辅助程序
grub-editenv	是一个用于编辑环境块的工具
grub-file	检验文件是否是给定类型
grub-fstest	是一个调试文件系统驱动程序的工具
grub-glue-efi	将 32 位和 64 位二进制文件结合成一个文件 (以便在 Apple 生产的机器使用)。
grub-install	在您的驱动器上安装 GRUB
grub-kbdcomp	是一个脚本，用于将 xkb 布局转化为 GRUB 能够识别的格式
grub-macbless	是 Mac 风格的，用于 HFS 和 HFS+ 文件系统的 bless 程序 (**bless** 仅用于 Apple 生产的机器；它能将设备变得可引导)
grub-menulst2cfg	将经典的 GRUB `menu.lst` 转化为 `grub.cfg` 以供 GRUB 2 使用
grub-mkconfig	生成一个 `grub.cfg` 文件
grub-mkimage	创建 GRUB 可引导镜像
grub-mklayout	生成 GRUB 键盘布局文件
grub-mknetdir	准备 GRUB 网络启动目录
grub-mkpasswd-pbkdf2	生成用于引导菜单的加密 PBKDF2 密码
grub-mkrelpath	生成相对于根目录的系统路径名称
grub-mkrescue	为软盘，CDROM/DVD，或 USB 设备创建 GRUB 可引导镜像
grub-mkstandalone	生成独立 (包含所有模块) 的镜像
grub-ofpathname	打印 GRUB 设备路径的帮助程序
grub-probe	探测给定路径或设备的信息
grub-reboot	仅为下次启动设置 GRUB 默认引导项
grub-render-label	为 Apple Mac 设置 Apple .disk_label
grub-script-check	在 GRUB 配置脚本中检查语法错误
grub-set-default	设置 GRUB 默认引导项
grub-sparc64-setup	grub-setup 使用的帮助程序
grub-syslinux2cfg	将 syslinux 配置文件转换为 grub.cfg 格式

8.65. Gzip-1.14

Gzip 软件包包含压缩和解压缩文件的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 21 MB

8.65.1. 安装 Gzip

准备编译 Gzip:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.65.2. Gzip 的内容

安装的程序: gunzip, gzexe, gzip, uncompress (与 gunzip 互为硬链接), zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, 以及 znew

简要描述

gunzip	解压缩 gzip 压缩的文件
gzexe	创建自解压可执行文件
gzip	使用 Lempel-Ziv (LZ77) 编码压缩文件
uncompress	解压压缩文件
zcat	将给定 gzip 压缩文件解压到标准输出
zcmp	在 gzip 压缩文件上运行 **cmp**
zdiff	在 gzip 压缩文件上运行 **diff**
zegrep	在 gzip 压缩文件上运行 **egrep**
zfgrep	在 gzip 压缩文件上运行 **fgrep**
zforce	为给定所有文件中的 gzip 压缩文件确保 `.gz` 扩展名，这样 **gzip** 就不会重复压缩它们；在文件传输过程中文件名被截断时，这个命令很有用
zgrep	在 gzip 压缩文件上运行 **grep**
zless	在 gzip 压缩文件上运行 **less**
zmore	在 gzip 压缩文件上运行 **more**
znew	将 **compress** 格式压缩文件重新压缩为 **gzip** 格式 —— 转换 `.z` 文件为 `.gz` 文件

8.66. IPRoute2-6.16.0

IPRoute2 软件包包含基于 IPv4 的基本和高级网络程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 17 MB

8.66.1. 安装 IPRoute2

该软件包中的 **arpd** 程序依赖于 LFS 不安装的 Berkeley DB，因此不会被构建。然而，用于 **arpd** 的一个目录和它的手册页仍会被安装。运行以下命令以防止它们的安装。

```
sed -i /ARPD/d Makefile
rm -fv man/man8/arpd.8
```

编译该软件包:

```
make NETNS_RUN_DIR=/run/netns
```

该软件包没有能够工作的测试套件。

安装该软件包:

```
make SBINDIR=/usr/sbin install
```

如果需要，安装该软件包的文档:

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

8.66.2. IPRoute2 的内容

安装的程序: bridge, ctstat (到 lnstat 的链接), genl, ifstat, ip, lnstat, nstat, routel, rtacct, rtmon, rtpr, rtstat (到 lnstat 的链接), ss, 以及 tc
安装的目录: /etc/iproute2, /usr/lib/tc, 以及 /usr/share/doc/iproute2-6.16.0

简要描述

bridge 配置网桥

ctstat 连接状态工具

genl 通用网络连接工具前端

ifstat 显示网络接口统计信息，包含每个接口上发送和接收的数据包数量

ip 该软件包的主程序。它包含许多功能，例如:
ip link `<device>` 允许用户查看和修改设备状态
ip addr 允许用户查看网络地址及其属性，添加新地址或删除旧地址
ip neigh 允许用户查看 ARP 近邻绑定及其属性，增加新近邻项，或删除旧项
ip rule 允许用户查看或修改路由策略
ip route 允许用户查看路由表或修改路由表规则
ip tunnel 允许用户查看 IP 隧道及其属性，或修改它们
ip maddr 允许用户查看多播地址及其属性，或修改它们
ip mroute 允许用户设定、修改或删除多播路由
ip monitor 允许用户连续地监视设备、地址和路由的状态

lnstat 提供 Linux 网络统计；它是旧的**rtstat** 的更通用、功能更完备的替代品

nstat 显示网络统计

routel **ip route** 的一个组件，用于显示路由表

rtacct 显示 `/proc/net/rt_acct` 的内容

rtmon 路由监视工具

rtpr 将 **ip -o** 的输出转换为可读形式

rtstat 路由状态工具

ss 与 **netstat** 命令相似；显示活动连接

tc 实现服务质量 (QoS) 和服务类型 (CoS) 协议的流量控制程序
tc qdisc 允许用户设定排队规则
tc class 允许用户设定基于排队规则调度的调度类
tc filter 允许用户设定 QoS/CoS 数据包过滤
tc monitor 可用于监视内核中流量控制策略的变化。

8.67. Kbd-2.8.0

Kbd 软件包包含按键表文件、控制台字体和键盘工具。

估计构建时间: 0.1 SBU
需要硬盘空间: 43 MB

8.67.1. 安装 Kbd

退格和删除键的行为在 Kbd 软件包的不同按键映射中不一致。以下补丁修复 i386 按键映射中的这个问题:

```
patch -Np1 -i ../kbd-2.8.0-backspace-1.patch
```

在应用补丁后，退格键生成编码为 127 的字符，删除键生成广为人知的 escape 序列。

删除多余的 **resizecons** 程序 (它需要已经不存在的 svgalib 提供视频模式文件 —— 一般使用 **setfont** 即可调整控制台大小) 及其手册页。

```
sed -i '/RESIZECONS_PROGS=/s/yes/no/' configure
sed -i 's/resizecons.8 //' docs/man/man8/Makefile.in
```

准备编译 Kbd:

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

配置选项的含义:

`--disable-vlock`
该选项防止构建 vlock 工具，因为它需要 chroot 环境中不可用的 PAM 库。

编译该软件包:

```
make
```

该软件包的测试在 chroot 环境中会失败，这是因为它们需要 valgrind。另外，即使在安装了 valgrind 的完整系统上，一些测试仍然在图形环境中失败。它们在非图形环境中能够通过。

安装该软件包:

```
make install
```

注意

对于一些语言 (如白俄罗斯文)，Kbd 软件包没有提供有用的的键盘映射。它提供的白俄罗斯文“by”键盘映射假设使用 ISO-8859-5 编码，但通常应该使用的是 CP1251 编码的键盘映射。使用白俄罗斯文等文字的用户需要单独下载可工作的键盘映射。

如果需要，安装该软件包的文档:

```
cp -R -v docs/doc -T /usr/share/doc/kbd-2.8.0
```

8.67.2. Kbd 的内容

安装的程序: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbdinfo, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (到 psfxtable 的链接), psfgettable (到 psfxtable 的链接), psfstriptable (到 psfxtable 的链接), psfxtable, setfont, setkeycodes, setleds, setmetamode, setvtrgb, showconsolefont, showkey, unicode_start, 以及 unicode_stop

安装的目录: /usr/share/consolefonts, /usr/share/consoletrans, /usr/share/keymaps, /usr/share/doc/kbd-2.8.0, 以及 /usr/share/unimaps

简要描述

chvt	修改当前虚拟终端
deallocvt	取消未使用的虚拟终端分配
dumpkeys	转储键盘转换表
fgconsole	打印活动虚拟终端的个数
getkeycodes	打印内核扫描码到键码的映射表
kbdinfo	获取终端状态信息
kbd_mode	报告或设置键盘模式
kbdrate	设置键盘重复和延迟率
loadkeys	加载键盘翻译表
loadunimap	加载内核 unicode 到字体的映射表
mapscrn	一个过时程序，曾用于将用户定义输出字符映射表加载到终端驱动程序；现在该任务由 **setfont** 完成
openvt	在新的虚拟终端 (VT) 启动程序
psfaddtable	向控制台字体增加 Unicode 字符表
psfgettable	提取控制台字体中嵌入的 Unicode 字符表
psfstriptable	删除控制台字体中嵌入的 Unicode 字符表
psfxtable	处理控制台字体的 Unicode 字符表
setfont	修改控制台上的增强图形适配器 (EGA) 和视频图像阵列 (VGA) 字体
setkeycodes	加载内核扫描码到键码的映射表项；在键盘上有特殊按键时很有用
setleds	设置键盘标志位和发光二极管 (LED)
setmetamode	定义键盘转换键 (meta-key) 处理
setvtrgb	设定所有虚拟终端的控制台颜色映射
showconsolefont	显示当前 EGA/VGA 控制台屏幕字体
showkey	报告键盘按键的扫描码、键码和 ASCII 编码
unicode_start	将键盘和控制台设定为 UNICODE 模式 [不要使用该程序，除非您的键盘映射文件是 ISO-8859-1 编码的。对于其他编码，该工具产生错误结果。]
unicode_stop	使键盘和控制台退出 UNICODE 模式

8.68. Libpipeline-1.5.8

Libpipeline 软件包包含用于灵活、方便地处理子进程流水线的库。

估计构建时间: 0.1 SBU
需要硬盘空间: 10 MB

8.68.1. 安装 Libpipeline

准备编译 Libpipeline:

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

编译该软件包:

```
make
```

测试用例需要我们已经从 LFS 删除的 Check 库。

安装该软件包:

```
make install
```

8.68.2. Libpipeline 的内容

安装的库: libpipeline.so

简要描述

`libpipeline` 用于安全地在子进程之间构建流水线

8.69. Make-4.4.1

Make 软件包包含一个程序，用于控制从软件包源代码生成可执行文件和其他非源代码文件的过程。

估计构建时间: 0.7 SBU
需要硬盘空间: 13 MB

8.69.1. 安装 Make

准备编译 Make:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

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

安装该软件包:

```
make install
```

8.69.2. Make 的内容

安装的程序: make

简要描述

make 自动确定软件包中需要 (重新) 构建的部分，并执行对应命令

8.70. Patch-2.8

Patch 软件包包含通过应用 “补丁” 文件，修改或创建文件的程序，补丁文件通常是 **diff** 程序创建的。

估计构建时间: 0.2 SBU
需要硬盘空间: 13 MB

8.70.1. 安装 Patch

准备编译 Patch：

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

编译该软件包：

```
make
```

运行命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.70.2. Patch 的内容

安装的程序: patch

简要描述

patch 根据补丁文件修改文件 (补丁文件一般是使用 **diff** 程序创建的差异清单。通过将这些差异应用到原始文件，**patch** 即可创建应用补丁的文件版本。)

8.71. Tar-1.35

Tar 软件包提供创建 tar 归档文件，以及对归档文件进行其他操作的功能。Tar 可以对已经创建的归档文件进行提取文件，存储新文件，更新文件，或者列出文件等操作。

估计构建时间: 0.6 SBU
需要硬盘空间: 43 MB

8.71.1. 安装 Tar

准备编译 Tar:

```
FORCE_UNSAFE_CONFIGURE=1  \
./configure --prefix=/usr
```

配置选项的含义:

`FORCE_UNSAFE_CONFIGURE=1`
该选项强制以 `root` 用户身份运行 `mknod` 测试。一般认为以 `root` 用户身份运行该测试是危险的，不过由于是在仅仅部分构建好的系统上运行测试，可以覆盖掉这个安全措施。

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

一项名为 capabilities: binary store/restore 的测试在运行时会失败，这是由于 LFS 没有 selinux 功能，但如果宿主系统的内核在构建 LFS 使用的文件系统上不支持扩展属性或安全标记，该测试会被跳过。

安装该软件包:

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.35
```

8.71.2. Tar 的内容

安装的程序: tar
安装的目录: /usr/share/doc/tar-1.35

简要描述

tar 创建称为 tarball 的档案文件，从档案文件中提取文件，或列出档案文件内容

8.72. Texinfo-7.2

Texinfo 软件包包含阅读、编写和转换 info 页面的程序。

估计构建时间: 0.4 SBU
需要硬盘空间: 160 MB

8.72.1. 安装 Texinfo

修复导致 Perl-5.42 或更新版本显示警告的代码模式:

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

准备编译 Texinfo:

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

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

可选地，安装属于 TeX 环境的组件:

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

make 命令参数的含义:

TEXMF=/usr/share/texmf
TEXMF Makefile 变量包含之后可能安装的 TeX 软件包的 TeX 目录树根位置。

Info 文档系统使用一个纯文本文件保存目录项的列表。该文件位于 /usr/share/info/dir。不幸的是，由于一些软件包 Makefile 中偶然出现的问题，它有时会与系统实际安装的 info 页面不同步。如果需要重新创建 /usr/share/info/dir 文件，可以运行以下命令完成这一工作:

```
pushd /usr/share/info
  rm -v dir
  for f in *
    do install-info $f dir 2>/dev/null
  done
popd
```

8.72.2. Texinfo 的内容

安装的程序: info, install-info, makeinfo (到 texi2any 的链接), pdftexi2dvi, pod2texi, texi2any, texi2dvi, texi2pdf, 以及 texindex
安装的库: MiscXS.so, Parsetexi.so, 以及 XSParagraph.so (都在 /usr/lib/texinfo 中)
安装的目录: /usr/share/texinfo 和 /usr/lib/texinfo

简要描述

info 用于阅读和手册页类似的 info 页面，手册页一般只解释可用的命令行选项，而 info 页面更为深入 [例如，可以对比 **man bison** 和 **info bison**。]

install-info 用于安装 info 页面；该命令更新 **info** 索引文件

makeinfo 将给定 Texinfo 源代码文档转换成 info 页面、纯文本或 HTML

pdftexi2dvi	将给定 Texinfo 文档格式化为可移植文档格式 (PDF) 文件
pod2texi	将 Pod 转换成 Texinfo 格式
texi2any	将 Texinfo 文档转换成其他几种格式
texi2dvi	将给定 Texinfo 文档格式化为可打印的设备无关文件
texi2pdf	将给定 Texinfo 文档格式化为可移植文档格式 (PDF) 文件
texindex	用于排序 Texinfo 索引文件

8.73. Vim-9.1.1629

Vim 软件包包含强大的文本编辑器。

估计构建时间: 3.7 SBU
需要硬盘空间: 259 MB

Vim 的替代品

如果您喜爱其他编辑器 —— 例如 Emacs、Joe、或者 Nano —— 参考 https://www.linuxfromscratch.org/blfs/view/12.4-systemd/postlfs/editors.html 中建议的安装说明。

8.73.1. 安装 Vim

首先，修改 `vimrc` 配置文件的默认位置为 `/etc`：

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

准备编译 Vim：

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

编译该软件包：

```
make
```

为了准备运行测试套件，需要使得 `tester` 用户拥有写入源代码目录树的权限，并排除一个含有需要 **curl** 或 **wget** 的测试的文件：

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

现在，以 `tester` 用户身份运行测试：

```
su tester -c "TERM=xterm-256color LANG=en_US.UTF-8 make -j1 test" \
   &> vim-test.log
```

测试套件会将大量二进制数据输出到屏幕。这可能扰乱当前终端设置 (特别是像我们所做的一样，覆盖 `TERM` 环境变量以满足测试套件的一些假设时)。为了避免这个问题，以上命令将输出重定向到日志文件。测试成功完成后，日志文件末尾会包含 `ALL DONE`。

安装该软件包：

```
make install
```

许多用户条件反射地输入 **vi**，而不是 **vim**。为了在用户习惯性地输入 **vi** 时能够执行 **vim**，为二进制程序和各种语言的手册页创建符号链接：

```
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
```

默认情况下，Vim 的文档安装在 `/usr/share/vim` 中。下面创建符号链接，使得可以通过 `/usr/share/doc/vim-9.1.1629` 访问文档，这个路径与其他软件包的文档位置格式一致：

```
ln -sv ../vim/vim91/doc /usr/share/doc/vim-9.1.1629
```

如果在安装 LFS 系统后安装了 X 窗口系统，可能需要在安装 X 后重新编译 Vim。Vim 提供的 GUI 版本编辑器需要 X 和一些额外的软件包才能安装。关于这一安装过程的更多信息，参考 Vim 文档和 BLFS 手册中位于 https://www.linuxfromscratch.org/blfs/view/12.4-systemd/postlfs/vim.html 的 Vim 安装页面。

8.73.2. 配置 Vim

默认情况下，**vim** 在不兼容 vi 的模式下运行。这对于过去使用其他编辑器的用户来说可能显得陌生。以下配置包含的 “nocompatible” 设定是为了强调编辑器使用了新的行为这一事实。它也提醒那些想要使用 “compatible” 模式的用户，必须在配置文件的一开始改变模式。这是因为它会修改其他设置，对这些设置的覆盖必须在设定模式后进行。执行以下命令创建默认 **vim** 配置文件：

```
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` 设定使得 **vim** 以一种更有用的方式 (也是默认方式) 行动，而不是兼容于 vi 的旧模式。删除其中的 “no” 可以保持旧的 **vi** 行为。`set backspace=2` 设定允许退格越过换行，自动缩进，以及插入模式的起始位置。参数 `syntax on` 启用 vim 符号高亮功能。参数 `set mouse=` 允许在 chroot 中或通过远程连接工作时使用鼠标正确地粘贴文本。最后，if 语句为 `set background=dark` 纠正 **vim** 对于某些终端模拟器背景色的猜测。这能够提供更适合这些程序黑色背景的配色方案。

关于其他可用选项的文档可以通过执行以下命令获得：

```
vim -c ':options'
```

注意

默认情况下 Vim 只安装英语拼写检查文件。如果希望安装您使用的语言的拼写检查文件，需要将适用于您的语言和字符编码的 `.spl` 文件和可选的 `.sug` 文件从 `runtime/spell` 复制到`/usr/share/vim/vim91/spell/`。

为了使用这些拼写检查文件，需要在 `/etc/vimrc` 中进行配置，例如：

```
set spelllang=en,ru
set spell
```

参阅 `runtime/spell/README.txt` 获得更多信息。

8.73.3. Vim 的内容

安装的程序: ex (到 vim 的链接), rview (到 vim 的链接), rvim (到 vim 的链接), vi (到 vim 的链接), view (到 vim 的链接), vim, vimdiff (到 vim 的链接), vimtutor, 以及 xxd

安装的目录: /usr/share/vim

简要描述

ex 以 ex 模式启动 **vim**

rview 是 **view** 的受限模式；不能启动 shell 命令，且不能挂起 **view**

rvim 是 **vim** 的受限模式；不能启动 shell 命令，且不能挂起 **vim**

vi	到 **vim** 的链接
view	以只读模式启动 **vim**
vim	文本编辑器
vimdiff	用 **vim** 编辑两个或三个文件版本，并显示差异
vimtutor	教会用户使用 **vim** 的基本快捷键和命令
xxd	创建文件的十六进制转储；也可以进行逆操作，因此可用于修补二进制文件

8.74. MarkupSafe-3.0.2

MarkupSafe 是一个为 XML/HTML/XHTML 标记语言实现字符串安全处理的 Python 模块。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 500 KB

8.74.1. 安装 MarkupSafe

输入以下命令，编译 MarkupSafe:

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

该软件包不包含测试套件。

安装该软件包:

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

8.74.2. MarkupSafe 的内容

安装的目录: /usr/lib/python3.13/site-packages/MarkupSafe-3.0.2.dist-info

8.75. Jinja2-3.1.6

Jinja2 是一个实现了简单的，Python 风格的模板语言的 Python 模块。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 2.6 MB

8.75.1. 安装 Jinja2

构建该软件包:

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

安装该软件包:

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

8.75.2. Jinja2 的内容

安装的目录: /usr/lib/python3.13/site-packages/Jinja2-3.1.6.dist-info

8.76. Systemd-257.8

Systemd 软件包包含控制系统引导、运行和关闭的程序。

估计构建时间: 1.4 SBU
需要硬盘空间: 310 MB

8.76.1. 安装 systemd

从默认的 udev 规则中删除不必要的组 `render` 和 `sgx`:

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

准备安装 systemd:

```
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 userdb=false         \
      -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-257.8
```

meson 选项的含义:

`--buildtype=release`
这个开关覆盖默认的构建模式 (“debug”)，因为该模式会生成未优化的二进制代码。

`-D default-dnssec=no`
这个开关禁用实验性的 DNSSEC 支持。

`-D firstboot=false`
这个开关防止 systemd 安装用于设定系统初始值的服务。在 LFS 中所有工作都会手工完成，因此不需要它们。

`-D install-tests=false`
这个开关防止 systemd 安装编译好的测试文件。

`-D ldconfig=false`
这个开关防止一个 systemd 单元的安装，它在引导时运行 **ldconfig**；这对于 LFS 等源代码发行版来说没有意义，还会增加引导时间。如果需要在引导时运行 **ldconfig**，可以删除这个开关。

`-D sysusers=false`
这个开关防止 systemd 安装负责设定 `/etc/group` 和 `/etc/passwd` 文件的服务。我们在上一章已经创建了这两个文件。这个守护进程在 LFS 系统中没有意义，因为我们已经手动创建了用户账户。

`-D rpmmacrosdir=no`
该选项禁止安装用于 systemd 的 RPM 宏，因为 LFS 并不支持 RPM。

`-D homed=disabled` 和 `-D userdb=false`
移除两个守护程序，它们的依赖项超出了 LFS 的范围。

`-D man=disabled`
防止手册页的生成，以避免额外的依赖项。我们将会从一个压缩包安装预先生成的手册页。

`-D mode=release`
禁用一些上游开发者认为尚处于实验阶段的功能。

`-D pamconfdir=no`
防止安装在 LFS 系统无法正常工作的 PAM 配置文件。

`-D dev-kvm-mode=0660`
默认的 udev 规则会允许所有用户访问 `/dev/kvm`。本书编辑团队认为这比较危险。该选项覆盖这一默认值。

`-D nobody-group=nogroup`
告诉该软件包组编号 65534 被分配给 `nogroup` 组。

`-D sysupdate=disabled`
不安装 **systemd-sysupdate** 工具。它被设计为用于更新二进制发行版，因此它对于从源码构建的，基本的 Linux 系统来说没有作用，而且如果它被启用但未正确配置，会在系统启动时产生错误消息。

`-D ukify=disabled`
不安装 **systemd-ukify** 脚本。它在运行时需要名为 pefile 的 Python 模块，而 LFS 和 BLFS 都不提供该模块。

编译该软件包：

```
ninja
```

一些测试需要包含基本信息的 `/etc/os-release` 文件。如果需要测试结果，运行：

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

已知一项名为 `systemd:core / test-namespace` 的测试在 LFS chroot 环境中会失败。已知名为 `systemd:test / test-copy` 的测试在使用较多并行任务测试时可能由于 I/O 拥堵而失败，但如果使用 **meson test test-copy** 命令单独运行它，它应该能正常通过。一些其他测试可能由于依赖一些内核配置选项而失败。

安装该软件包：

```
ninja install
```

安装手册页：

```
tar -xf ../../systemd-man-pages-257.8.tar.xz \
    --no-same-owner --strip-components=1     \
    -C /usr/share/man
```

创建 `/etc/machine-id` 文件，**systemd-journald** 需要它：

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

设定启动目标单元的基本结构：

```
systemctl preset-all
```

8.76.2. systemd 的内容

安装的程序: busctl, coredumpctl, halt (到 systemctl 的符号链接), hostnamectl, init, journalctl, kernel-install, localectl, loginctl, machinectl, mount.ddi (到 systemd-dissect 的符号链接), networkctl, oomctl, portablectl, poweroff (到 systemctl 的符号链接), reboot (到 systemctl 的符号链接), resolvconf (到 resolvectl 的符号链接), resolvectl, runlevel (到 systemctl 的符号链接), shutdown (到 systemctl 的符号链接), systemctl, systemd-ac-power, systemd-analyze, systemd-ask-password, systemd-cat, systemd-cgls, systemd-cgtop, systemd-confext (到 systemd-sysext 的符号链接), 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-repart, systemd-resolve (到 resolvectl 的符号链接), systemd-run, systemd-socket-activate, systemd-stdio-bridge, systemd-sysext, systemd-tmpfiles, systemd-tty-ask-password-agent, systemd-umount (到 systemd-mount 的符号链接), telinit (到 systemctl 的符号链接), timedatectl, 以及 udevadm

安装的库: libnss_myhostname.so.2, libnss_mymachines.so.2, libnss_resolve.so.2, libnss_systemd.so.2, libsystemd.so, libsystemd-shared-257.8.so (在 /usr/lib/systemd 中), 以及 libudev.so

安装的目录: /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/lib/systemd, /usr/lib/udev, /usr/include/systemd, /usr/lib/binfmt.d, /usr/lib/environment.d, /usr/lib/kernel, /usr/lib/modules-load.d, /usr/lib/sysctl.d, /usr/lib/systemd, /usr/lib/tmpfiles.d, /usr/share/doc/systemd-257.8, /usr/share/factory, /usr/share/systemd, /var/lib/systemd, 以及 /var/log/journal

简要描述

busctl	用于探查和监视 D-bus 总线
coredumpctl	用于从 systemd 日志获取核心转储
halt	通常调用 **shutdown** 并传递 `-h` 参数，除非已经处于运行级别 0，则它会通知内核停止系统运行；在系统被关闭时，它在 `/var/log/wtmp` 文件中进行记录
hostnamectl	用于查询和修改系统机器名和相关设置
init	在内核初始化硬件后第一个启动的进程；**init** 接管引导过程，并根据它的配置文件启动所有进程。在本例中，它启动 systemd
journalctl	用于查询 systemd 日志的内容
kernel-install	用于在 /boot 中添加或删除内核和 initramfs 映像文件；在 LFS 中，这项工作是手工完成的
localectl	用于查询和修改系统 locale 和键盘布局设置
loginctl	用于探查和控制 systemd 登录管理器的状态
machinectl	用于探查和控制 systemd 虚拟机和容器注册管理器的状态
networkctl	用于探查和配置 systemd-networkd 管理的网络连接状态
oomctl	控制 systemd 处理内存不足问题的守护进程

portablectl	用于在本地系统附加或移除可移植服务
poweroff	告诉内核停止系统运行并关闭计算机 (见 **halt**)
reboot	告诉内核重启系统 (见 **halt**)
resolvconf	为 **systemd-resolved** 注册 DNS 服务器和域设置
resolvectl	向网络名称解析管理器发送控制命令，或解析域名，IPv4 和 IPv6 地址，DNS 记录，以及服务
runlevel	报告当前系统运行级别和上一个运行级别，上一个运行级别被记录在 `/run/utmp` 中
shutdown	安全地关闭系统，向所有进程发送信号，并通知所有登录用户
systemctl	用于探查和控制 systemd 系统和服务管理器的状态
systemd-ac-power	报告系统是否连接到外部电源。
systemd-analyze	用于分析系统引导性能，也能找出导致引导性能问题的 systemd 单元
systemd-ask-password	用于以命令行指定的消息向用户询问系统密码
systemd-cat	用于将进程的标准输出和错误输出重定向到系统日志
systemd-cgls	用树的形式递归地显示指定 Linux 控制组层次结构的内容
systemd-cgtop	显示本地 Linux 控制组层次结构中占用资源最多的，可以按 CPU、内存和磁盘 I/O 负载排序
systemd-creds	显示和处理凭证
systemd-delta	用于识别和比较那些覆盖了 `/usr` 中默认值的 `/etc` 中的配置文件
systemd-detect-virt	确定系统是否在虚拟化环境中运行，并据此调节 udev
systemd-dissect	用于检查 OS 磁盘映像
systemd-escape	用于转义字符串，以便将其包含在 systemd 单元名中
systemd-hwdb	用于管理硬件数据库 (hwdb)
systemd-id128	生成和打印 id128 (UUID) 串
systemd-inhibit	用于在关机、休眠或待机抑制锁被锁定的情况下运行程序，在进程结束前防止关闭系统等动作
systemd-machine-id-setup	被系统安装工具用于在安装时以随机生成的 ID 初始化 `/etc/machine-id` 中的机器 ID
systemd-mount	一个用于临时挂载或自动挂载驱动器的工具
systemd-notify	被守护脚本用于将状态变化告知 init 系统
systemd-nspawn	用于在轻量级命名空间容器中运行命令，或完整的操作系统
systemd-path	用于查询系统和用户路径
systemd-repart	在 systemd 和 OS 映像 (例如容器) 共同使用时，可用于在分区表中添加分区或增长分区大小
systemd-resolve	用于解析域名，IPv4 和 IPv6 地址，DNS 资源记录，以及服务
systemd-run	用于创建一个临时的 .service 或 .scope 单元，并在其中运行指定命令；这对于验证 systemd 单元很有用

systemd-socket-activate	用于监听 socket 服务，并在 socket 成功连接时启动进程
systemd-sysext	启用系统扩展镜像
systemd-tmpfiles	根据 `tmpfiles.d` 目录中的配置文件给定的文件格式和位置，创建、修改，以及清理易失性、临时性文件和目录
systemd-umount	卸载挂载点
systemd-tty-ask-password-agent	列出或处理等待中的 systemd 密码请求
telinit	告诉 **init** 切换到某个运行级别
timedatectl	用于查询和修改系统时钟及其设置
udevadm	通用 udev 管理工具，它控制 udevd 守护进程，提供 udev 数据库的信息，监视 uevent 事件，等待 uevent 事件结束，测试 udev 配置，或对于给定设备触发 uevent 事件
`libsystemd`	主要的 systemd 工具库
`libudev`	用于访问 udev 设备信息的库

8.77. D-Bus-1.16.2

D-bus 是一个消息总线系统，即应用程序之间互相通信的一种简单方式。D-Bus 提供一个系统守护进程 (负责 “添加了新硬件” 或 “打印队列发生改变” 等事件)，并对每个用户登录会话提供一个守护进程 (负责一般用户程序的进程间通信)。另外，消息总线被构建在一个通用的一对一消息传递网络上，它可以被任意两个程序用于直接通信 (不需通过消息总线守护进程)。

估计构建时间: 0.1 SBU
需要硬盘空间: 17 MB

8.77.1. 安装 D-Bus

准备编译 D-Bus:

```
mkdir build
cd    build

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

meson 选项的含义:

`--wrap-mode=nofallback`
该选项防止 meson 为了构建一些测试而尝试下载一份 Glib 软件包。

编译该软件包:

```
ninja
```

运行命令以测试编译结果:

```
ninja test
```

许多测试被禁用，因为它们需要 LFS 不包含的软件包。阅读 BLFS 手册以查阅如何运行完整的测试套件。

安装该软件包:

```
ninja install
```

创建符号链接，使 D-Bus 和 systemd 使用同一个 `machine-id` 文件:

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

8.77.2. D-Bus 的内容

安装的程序: dbus-cleanup-sockets, dbus-daemon, dbus-launch, dbus-monitor, dbus-run-session, dbus-send, dbus-test-tool, dbus-update-activation-environment, 以及 dbus-uuidgen
安装的库: libdbus-1.so
安装的目录: /etc/dbus-1, /usr/include/dbus-1.0, /usr/lib/dbus-1.0, /usr/share/dbus-1, /usr/share/doc/dbus-1.16.2, 以及 /var/lib/dbus

简要描述

dbus-cleanup-sockets	用于清理目录中遗留的套接字
dbus-daemon	是 D-Bus 消息总线守护程序
dbus-launch	从 shell 脚本启动 **dbus-daemon**
dbus-monitor	监视通过一个 D-Bus 消息总线的消息

dbus-run-session	从 shell 脚本启动一个 **dbus-daemon** 的会话总线实例，并在该会话中启动给定程序
dbus-send	向 D-Bus 消息总线发送消息
dbus-test-tool	是一个帮助软件包测试 D-Bus 的工具
dbus-update-activation-environment	更新将会为 D-Bus 会话服务设置的环境变量
dbus-uuidgen	产生通用唯一识别码
`libdbus-1`	包含用于和 D-Bus 消息总线通信的 API 函数

8.78. Man-DB-2.13.1

Man-DB 软件包包含查找和阅读手册页的程序。

估计构建时间: 0.3 SBU
需要硬盘空间: 44 MB

8.78.1. 安装 Man-DB

准备编译 Man-DB:

```
./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
```

配置选项的含义:

`--disable-setuid`
该选项防止将 **man** 程序 setuid 到用户 `man`。

`--enable-cache-owner=bin`
该选项将系统范围的缓存文件所有者改为用户 `bin`。

`--with-...`
这三个选项设定一些默认程序。**lynx** 是基于文本的 web 浏览器 (安装过程可在 BLFS 中查阅)，**vgrind** 将程序源代码转换成 Groff 输入，**grap** 用于在 Groff 文档中画图。**vgrind** 和 **grap** 在阅读手册页时一般用不到。它们不是 LFS 或 BLFS 的一部分，但如果需要的话，您应该可以在完成 LFS 的构建后自行安装它们。

编译该软件包:

```
make
```

运行命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.78.2. LFS 中的非英文手册页

下表展示了 Man-DB 假定的安装在 `/usr/share/man/<ll>` 中的手册页的编码字符集。另外，Man-DB 还能正确地判断出这些页面是否为 UTF-8 编码。

表 8.1. 传统 8 位手册页的预期字符编码

语言 (代码)	编码	语言 (代码)	编码
丹麦语 (da)	ISO-8859-1	克罗地亚语 (hr)	ISO-8859-2
德语 (de)	ISO-8859-1	匈牙利语 (hu)	ISO-8859-2
英语 (en)	ISO-8859-1	日语 (ja)	EUC-JP
西班牙语 (es)	ISO-8859-1	朝鲜语 (ko)	EUC-KR

语言 (代码)	编码	语言 (代码)	编码
爱沙尼亚语 (et)	ISO-8859-1	立陶宛语 (lt)	ISO-8859-13
芬兰语 (fi)	ISO-8859-1	拉脱维亚语 (lv)	ISO-8859-13
法语 (fr)	ISO-8859-1	马其顿语 (mk)	ISO-8859-5
爱尔兰语 (ga)	ISO-8859-1	波兰语 (pl)	ISO-8859-2
加利西亚语 (gl)	ISO-8859-1	罗马尼亚语 (ro)	ISO-8859-2
印度尼西亚语 (id)	ISO-8859-1	希腊文 (el)	ISO-8859-7
冰岛语 (is)	ISO-8859-1	斯洛伐克语 (sk)	ISO-8859-2
意大利语 (it)	ISO-8859-1	斯洛文尼亚语 (sl)	ISO-8859-2
挪威巴克摩语 (nb)	ISO-8859-1	拉丁文书写的塞尔维亚语 (sr@latin)	ISO-8859-2
荷兰语 (nl)	ISO-8859-1	塞尔维亚语 (sr)	ISO-8859-5
挪威尼诺斯克语 (nn)	ISO-8859-1	土耳其语 (tr)	ISO-8859-9
挪威语 (no)	ISO-8859-1	乌克兰语 (uk)	KOI8-U
葡萄牙语 (pt)	ISO-8859-1	越南语 (vi)	TCVN5712-1
瑞典语 (sv)	ISO-8859-1	简体中文 (zh_CN)	GBK
白俄罗斯语 (be)	CP1251	简体中文，新加坡 (zh_SG)	GBK
保加利亚语 (bg)	CP1251	繁体中文，香港特别行政区 (zh_HK)	BIG5HKSCS
捷克语 (cs)	ISO-8859-2	繁体中文 (zh_TW)	BIG5

注意

用该表之外的语言编写的手册页不被支持。

8.78.3. Man-DB 的内容

安装的程序: accessdb, apropos (到 whatis 的链接), catman, lexgrog, man, man-recode, mandb, manpath, 以及 whatis
安装的库: libman.so 和 libmandb.so (都在 /usr/lib/man-db 中)
安装的目录: /usr/lib/man-db, /usr/libexec/man-db, and /usr/share/doc/man-db-2.13.1

简要描述

accessdb 将 **whatis** 数据库内容转储为人类可读格式
apropos 搜索 **whatis** 数据库，显示包含给定字符串的系统命令的简要描述
catman 创建或更新预先格式化的手册页
lexgrog 显示给定手册页的单行摘要信息
man 格式化并显示请求的手册页
man-recode 转换手册页的编码
mandb 创建或更新 **whatis** 数据库

manpath	显示 $MANPATH 的内容，或者 (如果 $MANPATH 未设定) 根据 man.conf 和用户环境确定的合适搜索路径
whatis	搜索 **whatis** 数据库，显示包含给定关键词的系统命令的简要描述
`libman`	包含 **man** 运行时支持
`libmandb`	包含 **man** 运行时支持

8.79. Procps-ng-4.0.5

Procps-ng 软件包包含监视进程的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 28 MB

8.79.1. 安装 Procps-ng

准备编译 Procps-ng:

```
./configure --prefix=/usr                            \
            --docdir=/usr/share/doc/procps-ng-4.0.5 \
            --disable-static                         \
            --disable-kill                           \
            --enable-watch8bit                       \
            --with-systemd
```

配置选项的含义:

`--disable-kill`
该选项使得 **kill** 命令不被构建；它将由 Util-linux 软件包提供。

`--enable-watch8bit`
该选项为 **watch** 命令启用 ncursesw 支持，这样它就能处理 8 位字符。

编译该软件包:

```
make
```

如果要运行测试套件，执行命令:

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

已知名为 `ps with output flag bsdtime,cputime,etime,etimes` 的一项测试在宿主系统内核未启用 `CONFIG_BSD_PROCESS_ACCT` 时会失败。另外，一项 pgrep 测试在 chroot 环境中可能失败。

安装该软件包:

```
make install
```

8.79.2. Procps-ng 的内容

安装的程序: free, pgrep, pidof, pkill, pmap, ps, pwdx, slabtop, sysctl, tload, top, uptime, vmstat, w, 以及 watch
安装的库: libproc-2.so
安装的目录: /usr/include/procps 和 /usr/share/doc/procps-ng-4.0.5

简要描述

free	报告系统中可用和已用内存 (包括物理内存和交换空间) 的容量
pgrep	根据名称和其他属性查找进程
pidof	报告给定程序的 PID
pkill	根据名称和其他属性向进程发送信号
pmap	报告给定进程的内存映射
ps	列出正在运行的进程
pwdx	报告一个进程的当前工作目录

slabtop 实时显示内核 slab 缓存详细信息
sysctl 在运行时修改内核参数
tload 打印当前系统平均负载示意图
top 列出 CPU 占用最大的进程列表；它实时地提供处理器活动的连续概况
uptime 报告系统运行时间、登录用户数目和系统平均负载
vmstat 报告虚拟内存统计，给出进程、内存、分页、块输入输出 (IO)、陷阱和 CPU 活动信息
w 显示当前登录用户和它们的登录地点、时间
watch 重复执行给定命令，显示其输出的第一页；这使得用户可以观察输出随时间的变化
`libproc-2` 包含该软件包大多数程序使用的函数

8.80. Util-linux-2.41.1

Util-linux 软件包包含若干工具程序。这些程序中有处理文件系统、终端、分区和消息的工具。

估计构建时间: 0.5 SBU
需要硬盘空间: 346 MB

8.80.1. 安装 Util-linux

准备编译 Util-linux:

```
./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-2.41.1
```

--disable 和 --without 选项防止一些警告，它们与那些需要 LFS 中不存在的依赖项，或者和其他软件包安装的程序不兼容的组件的构建过程相关。

编译该软件包:

```
make
```

如果希望进行测试，创建 `/etc/fstab` 文件以满足两项测试的要求，并以非 `root` 用户身份运行测试套件:

警告

以 `root` 用户身份运行测试套件可能对系统造成损害。为了运行它，内核配置选项 CONFIG_SCSI_DEBUG 必须在当前运行的系统中可用，且必须被构建为内核模块。直接将其构建为内核的一部分会导致系统无法引导。为了测试的完整覆盖，必须安装其他 BLFS 软件包。如果希望的话，可以在引导构建完成的 LFS 系统后，执行以下命令运行测试:

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

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

如果宿主系统内核没有启用配置选项 `CONFIG_CRYPTO_USER_API_HASH`，或者没有启用任何提供 SHA256 实现的选项 (例如，`CONFIG_CRYPTO_SHA256`，或者 `CONFIG_CRYPTO_SHA256_SSSE3` —— 如果 CPU 支持扩展 SSE3)，则 hardlink 测试会失败。另外，lsfd: inotify 这项测试在内核配置选项 `CONFIG_NETLINK_DIAG` 未启用时会失败。

已知一项测试，`kill: decode functions`，在使用 bash-5.3-rc1 或更新版本时会失败。

安装该软件包:

```
make install
```

8.80.2. Util-linux 的内容

安装的程序: 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, hexdump, hwclock, i386 (到 setarch 的链接), ionice, ipcmk, ipcrm, ipcs, irqtop, isosize, kill, last, lastb (到 last 的链接), ldattach, linux32 (到 setarch 的链接), linux64 (到 setarch 的链接), logger, look, losetup, lsblk, lscpu, lsipc, lsirq, lslocks, lslogins, lsmem, lsns, mcookie, mesg, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, mountpoint, namei, nsenter, partx, pivot_root, prlimit, readprofile, rename, renice, resizepart, rev, rfkill, rtcwake, script, scriptlive, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swaplabel, swapoff, swapon, switch_root, taskset, uclampset, ul, umount, uname26 (到 setarch 的链接), unshare, utmpdump, uuidd, uuidgen, uuidparse, wall, wdctl, whereis, wipefs, x86_64 (到 setarch 的链接), 以及 zramctl

安装的库: libblkid.so, libfdisk.so, libmount.so, libsmartcols.so, 以及 libuuid.so

安装的目录: /usr/include/blkid, /usr/include/libfdisk, /usr/include/libmount, /usr/include/libsmartcols, /usr/include/uuid, /usr/share/doc/util-linux-2.41.1, 以及 /var/lib/hwclock

简要描述

addpart 告知 Linux 内核有新的分区

agetty 打开 tty 端口，提示输入登录名，再启动 **login** 程序

blkdiscard 丢弃设备上的扇区

blkid 一个命令行工具，用于定位和打印块设备属性

blkzone 用于管理分区存储的块存储设备

blockdev 允许用户从命令行调用块设备 ioctl

cal 显示简单的日历

cfdisk 操作给定设备的分区表

chcpu 修改 CPU 状态

chmem 配置内存

choom 显示或调整 OOM-killer 分数，这一数值用于确定 Linux 在内存不足时首先杀死哪个进程

chrt 操纵进程实时属性

col 过滤掉反向换行符

colcrt 为缺失加粗、半行等功能的终端过滤 **nroff** 输出

colrm 过滤掉给定列

column 将给定文件格式化为多栏

ctrlaltdel 将 Ctrl+Alt+Del 键组合的功能设定为硬复位或软复位

delpart 要求 Linux 内核删除分区

dmesg 转储内核引导消息

eject 弹出可移动媒体

fallocate 为文件预先分配空间
fdisk 操作给定设备的分区表
fincore 统计给定文件在内存中占用的页面数
findfs 根据标签或统一标识符 (UUID) 查找文件系统
findmnt 是 libmount 库的命令行接口，可以处理 mountinfo、fstab 和 mtab 文件
flock 获取文件锁，并在持有锁的情况下运行命令
fsck 用于检查或修复文件系统
fsck.cramfs 用于对给定设备上的 Cramfs 文件系统进行一致性检查
fsck.minix 用于对给定设备上的 Minix 文件系统进行一致性检查
fsfreeze 是内核驱动 ioctl 操作 FIFREEZE/FITHAW 的简单包装
fstrim 丢弃已挂载文件系统上未使用的块
getopt 解析给定命令行的选项
hardlink 通过创建硬链接，归并重复的文件
hexdump 以十六进制、十进制、八进制或 ascii 格式转储文件
hwclock 读取或设置系统硬件时钟，它又被称为实时时钟 (RTC) 或基本输入输出系统 (BIOS) 时钟
i386 到 setarch 的符号链接
ionice 设定程序的 IO 调度类和优先级
ipcmk 创建多种 IPC 资源
ipcrm 删除给定的进程间通信 (IPC) 资源
ipcs 提供 IPC 状态信息
irqtop 以 top(1) 风格显示内核中断计数信息
isosize 报告 ISO 9660 文件系统的大小
kill 向进程发送信号
last 显示哪些用户最后登录 (或登出)，在 `/var/log/wtmp` 文件中反向搜索；它也会显示系统引导、关闭和运行级别变化记录
lastb 显示 `/var/log/btmp` 记录的失败登录企图
ldattach 为串口线附加行规则
linux32 到 setarch 的符号链接
linux64 到 setarch 的符号链接
logger 将给定消息记入系统日志
look 显示以给定字符串开始的行
losetup 设定和控制回环设备
lsblk 以树状格式列出所有或给定块设备的信息
lscpu 打印 CPU 体系结构信息
lsfd 显示当前已打开的文件信息；可以代替 **lsof** 命令
lsipc 打印系统当前部署的 IPC 设施的信息
lsirq 显示内核中断计数信息
lslocks 列出本地系统锁

lslogins 列出用户、组和系统账户的信息
lsmem 列出可用内存的范围和它们的在线状态
lsns 列出命名空间
mcookie 为 **xauth** 创建魔术 cookie (128位随机十六进制数)
mesg 控制其他用户能否向当前用户终端发送消息
mkfs 在设备 (一般是硬盘分区) 上创建文件系统
mkfs.bfs 创建 Santa Cruz Operations (SCO) bfs 文件系统
mkfs.cramfs 创建 cramfs 文件系统
mkfs.minix 创建 Minix 文件系统
mkswap 将给定文件或设备初始化为交换空间
more 在屏幕上分页文本的过滤器
mount 将给定设备上的文件系统挂载到文件系统树结构中的给定目录
mountpoint 检查目录是否为挂载点
namei 显示给定路径中的符号链接
nsenter 在其他程序的命名空间中运行程序
partx 告知内核磁盘分区的存在性和编号
pivot_root 将当前进程的根文件系统设为给定文件系统
prlimit 获取和设定进程资源限制
readprofile 读取内核性能分析信息
rename 重命名给定文件，将给定字符串替换为另一个字符串
renice 修改正在运行的进程的优先级
resizepart 要求 Linux 内核改变分区大小
rev 反转给定文件的每一行
rfkill 用于启用或禁用无线设备的工具
rtcwake 进入睡眠状态，直到给定的唤醒时间
script 记录终端会话打字机文档
scriptlive 根据计时信息重新运行会话打字机文档
scriptreplay 根据计时信息重放终端会话打字机文档
setarch 在新程序环境中修改系统报告的体系结构，并设置进程执行域信息
setsid 在新会话中运行给定程序
setterm 设定终端属性
sfdisk 一个分区表修改器
sulogin 允许 `root` 登录；一般在系统进入单用户模式时由 **init** 执行
swaplabel 允许修改交换空间 UUID 和标签
swapoff 禁止在文件或设备上进行分页交换
swapon 启用文件或设备上的分页交换，或列出当前用于交换的设备和文件
switch_root 将另一个文件系统切换为挂载树的根
taskset 获取或设置进程 CPU 亲和性

uclampset 调节进程资源占用限位属性
ul 将下划线转换为在当前终端中表示下划线的 escape 序列的过滤器
umount 断开文件系统与系统文件目录树的连接
uname26 到 setarch 的符号链接
unshare 在某些命名空间与父进程脱离的情况下运行程序
utmpdump 以用户友好的格式显示给定登录文件内容
uuidd UUID 库使用的守护进程，用于安全、确保唯一性地生成 UUID
uuidgen 创建新的 UUID。每个新创建的 UUID 都是一个随机数，它大概率和其他 UUID[128]——包括在本地系统或其他系统，在过去或未来创建的 UUID —— 都不同 (可能存在 2 个不同的 UUID)
uuidparse 用于解析统一标识符的工具
wall 显示文件或标准输入 (默认值) 的内容到所有登录用户的终端
wdctl 显示硬件看门狗电路状态
whereis 报告给定命令二进制，源代码，以及手册页文件的位置
wipefs 从设备上擦除文件系统签名
x86_64 到 setarch 的符号链接
zramctl 设定和控制 zram (压缩内存盘) 设备的程序
`libblkid` 包含设备识别和标识提取子程序
`libfdisk` 包含操作分区表的子程序
`libmount` 包含挂载和解挂块设备的子程序
`libsmartcols` 包含以表格形式在屏幕上输出的辅助子程序
`libuuid` 包含为对象生成唯一标识符，使它在本地系统以外也可以访问的子程序

8.81. E2fsprogs-1.47.3

E2fsprogs 软件包包含处理 ext2 文件系统的工具。此外它也支持 ext3 和 ext4 日志文件系统。

估计构建时间: 机械硬盘上 2.4 SBU，固态硬盘上 0.5 SBU
需要硬盘空间: 100 MB

8.81.1. 安装 E2fsprogs

E2fsprogs 文档推荐在源代码目录树中的一个子目录内构建该软件包:

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

准备编译 E2fsprogs:

```
../configure --prefix=/usr       \
             --sysconfdir=/etc   \
             --enable-elf-shlibs \
             --disable-libblkid  \
             --disable-libuuid   \
             --disable-uuidd     \
             --disable-fsck
```

配置选项的含义:

--enable-elf-shlibs
该选项表示创建该软件包中一些程序使用的共享库。

--disable-*
这些选项防止构建和安装 libuuid 和 libblkid 库，uuidd 守护程序，以及 **fsck** 包装器；因为 Util-linux 会安装更新的版本。

编译该软件包:

```
make
```

执行以下命令，以运行测试:

```
make check
```

已知一项名为 m_assume_storage_prezeroed 的测试会失败。已知另一项名为 m_rootdir_acl 的测试在 LFS 使用的文件系统不是 ext4 时可能失败。

安装该软件包:

```
make install
```

删除无用的静态库:

```
rm -fv /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

该软件包安装了一个 gzip 压缩的 .info 文件，却没有更新系统的 dir 文件。执行以下命令解压该文件，并更新系统 dir 文件:

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir /usr/share/info/libext2fs.info
```

如果需要，执行以下命令创建并安装一些额外的文档:

```
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
```

8.81.2. 配置 E2fsprogs

`/etc/mke2fs.conf` 包含 **mke2fs** 一些命令行选项的默认值。您可以修改这些默认值，使之符合您的需求。例如，一些工具 (未被 LFS 或 BLFS 包含) 无法识别启用了 `metadata_csum_seed` 特性的 `ext4` 文件系统。**如果**需要使用这样的工具，可以执行命令，从默认的 `ext4` 特性列表中移除该特性：

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

更多信息详见手册页 mke2fs.conf(5)。

8.81.3. E2fsprogs 的内容

安装的程序: 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, 以及 tune2fs

安装的库: libcom_err.so, libe2p.so, libext2fs.so, 以及 libss.so

安装的目录: /usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/lib/e2fsprogs, /usr/share/et, 以及 /usr/share/ss

简要描述

badblocks 在一个设备 (一般是磁盘分区) 上搜索坏块

chattr 改变 `ext{234}` 文件系统上文件的标志属性

compile_et 一个错误表编译器；它将包含错误编号名称和消息的表转化成 C 源代码，以和 `com_err` 库一起使用

debugfs 一个文件系统调试器；可以检验并修改 `ext{234}` 文件系统的状态

dumpe2fs 打印给定设备上文件系统的超级块和块组信息

e2freefrag 报告可用空间碎片信息

e2fsck 用于检查或修复 `ext{234}` 文件系统

e2image 用于将 `ext{234}` 文件系统关键数据保存到文件

e2label 显示或修改给定设备上的 `ext{234}` 文件系统标签

e2mmpstatus 检查 `ext4` 文件系统的MMP (多重挂载保护) 状态

e2scrub 检查已经挂载的 `ext{234}` 文件系统的内容

e2scrub_all 检查所有已挂载的 `ext{234}` 文件系统是否存在错误

e2undo 重放设备上找到的 `ext{234}` 文件系统撤销日志 [可以用于撤销 E2fsprogs 程序的失败操作。]

e4crypt `Ext4` 文件系统加密工具

e4defrag 用于 `ext4` 文件系统的在线碎片整理工具

filefrag 报告特定文件碎片化程度

fsck.ext2 默认情况下检查 `ext2` 文件系统，是 **e2fsck** 的硬链接

fsck.ext3 默认情况下检查 `ext3` 文件系统，是 **e2fsck** 的硬链接

fsck.ext4 默认情况下检查 `ext4` 文件系统，是 **e2fsck** 的硬链接

logsave 将命令输出保存到日志文件

lsattr 列出 ext2 文件系统上的文件属性

mk_cmds 将包含命令名称和帮助信息的表格转换成 C 源代码文件，以便和 `libss` 子系统库一起使用

mke2fs 在给定设备上创建 `ext{234}` 文件系统

mkfs.ext2 默认情况下创建 `ext2` 文件系统，是 **mke2fs** 的硬链接

mkfs.ext3 默认情况下创建 `ext3` 文件系统，是 **mke2fs** 的硬链接

mkfs.ext4 默认情况下创建 `ext4` 文件系统，是 **mke2fs** 的硬链接

mklost+found 创建 `lost+found` 目录；它在 `ext{234}` 文件系统上为该目录预先分配磁盘块，以减轻 **e2fsck** 的负担

resize2fs 可以用于扩大或压缩 `ext{234}` 文件系统

tune2fs 调整 `ext{234}` 文件系统的可调参数

`libcom_err` 公用错误显示子程序

`libe2p` 被 **dumpe2fs**、**chattr**, 和 **lsattr** 使用

`libext2fs` 包含允许用户级程序操纵 `ext{234}` 文件系统的子程序

`libss` 被 **debugfs** 使用

8.82. 关于调试符号

许多程序和库在默认情况下被编译为带有调试符号的二进制文件 (通过使用 **gcc** 的 `-g` 选项)。这意味着在调试这些带有调试信息的程序和库时，调试器不仅能给出内存地址，还能给出子程序和变量的名称。

然而，包含这些调试符号会显著增大程序或库的体积。下面是两个表现调试符号占用空间的例子：

· 一个有调试符号的 **bash** 二进制程序：1200 KB

· 一个没有调试符号的 **bash** 二进制程序：480 KB (小 60%)

· 带有调试符号的 Glibc 和 GCC 文件 (`/lib` 和 `/usr/lib` 目录中)：87 MB

· 没有调试符号的 Glibc 和 GCC 文件：16 MB (小 82%)

具体的文件大小与使用的编译器和 C 库相关，但是移除调试符号的程序通常比移除调试符号前小 50% 到 80%。由于大多数用户永远不会用调试器调试系统软件，可以通过移除它们的调试符号，回收大量磁盘空间。下一节展示如何从系统程序和库中移除所有调试符号。

8.83. 移除调试符号

本节是可选的。如果系统不是为程序员设计的，也没有调试系统软件的计划，可以通过从二进制程序和库移除调试符号和不必要的符号表项，将系统的体积减小约 2 GB。对于一般的 Linux 用户，这不会造成任何不便。

大多数使用以下命令的用户不会遇到什么困难。但是，如果打错了命令，很容易导致新系统无法使用。因此，在运行 **strip** 命令前，最好备份 LFS 系统的当前状态。

strip 命令的 `--strip-unneeded`选项从程序或库中移除所有调试符号。它也会移除所有链接器 (对于静态库) 或动态链接器 (对于动态链接的程序和共享库) 不需要的符号表项。`--strip-debug` 选项则不会移除符号表项。一些程序可能需要这些符号表项，而且 `unneeded` 和 `debug` 相比，进一步节省存储空间的效果并不明显。例如，原始的 libc.a 有 22.4 MB。使用 `--strip-debug` 移除调试符号后，其大小为 5.9 MB。而使用 `--strip-unneeded` 只会将其进一步缩减到 5.8 MB。

下面将一些库的调试符号使用 Zstd 压缩并保存在单独的文件中。后续在 BLFS 中，一些软件包的测试套件使用了 valgrind 或 gdb 运行退化测试，它们需要这些调试信息。

需要注意的是，**strip** 命令会覆盖它正在处理的二进制程序或库文件。这可能导致正在使用该文件中代码或数据的进程崩溃。如果运行 **strip** 的进程受到影响，则可能导致正在被处理的程序或库完全损坏；这可能导致系统完全不可用。为了避免这种情况，将一些库和程序复制到 `/tmp` 中，在那里移除调试符号，再使用 **install** 命令重新安装它们。(第 8.2.1 节 “升级问题” 中的相关条目介绍了使用 **install** 命令的原因。)

注意

ELF 加载器的文件名在 64 位系统是 ld-linux-x86-64.so.2，在 32 位系统是 ld-linux.so.2。下面的命令会为当前架构选择正确的文件名，并排除文件名以 `g` 结尾的文件。

重要

在构建过程中，如果任何一个软件包的版本和本书指定的版本不同（无论是依照安全公告的要求还是为了满足个人需求），则可能需要更新 `save_usrlib` 或者 `online_usrlib` 中的库文件名。**否则可能导致系统完全无法使用**。

```
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
```

这里会产生关于很多文件的错误信息，因为无法识别这些文件的格式。这些警告可以安全地忽略。它们表明那些文件是脚本文件，而不是二进制文件。

8.84. 清理系统

最后，清理在执行测试的过程中遗留的一些文件：

```
rm -rf /tmp/{*,.*}
```

在 /usr/lib 和 /usr/libexec 目录中还有一些扩展名为 .la 的文件。它们是 "libtool 档案" 文件。正如我们已经讨论过的，它们在现代 Linux 系统中仅用于 libltdl。LFS 中没有库被设计为通过 libltdl 加载，而且已知一些 .la 文件能导致 BLFS 软件包构建失败。现在删除这些文件：

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

如果希望了解更多关于 libtool 档案文件的信息，参阅 BLFS 章节 "About Libtool Archive (.la) files"。

在第 6 章和第 7 章中构建的编译器仍然有一部分安装在系统上，它现在已经没有存在的意义了。执行命令删除它：

```
find /usr -depth -name $(uname -m)-lfs-linux-gnu\* | xargs rm -rf
```

最后，移除上一章开始时创建的临时 'tester' 用户账户。

```
userdel -r tester
```

第 9 章 系统配置

9.1. 概述

本章讨论配置文件和 systemd 服务。第一，展示设定网络通常需要的配置文件。

· 第 9.2 节 “一般网络配置”。
· 第 9.2.3 节 “配置系统主机名”。
· 第 9.2.4 节 “自定义 /etc/hosts 文件”。

第二，讨论影响设备正确配置的问题。

· 第 9.3 节 “设备和模块管理概述”。
· 第 9.4 节 “管理设备”。

第三，展示如何配置系统时钟和键盘布局。

· 第 9.5 节 “配置系统时钟”。
· 第 9.6 节 “配置 Linux 控制台”。

第四，简要介绍用户登录系统时使用的脚本和配置文件。

· 第 9.7 节 “配置系统 Locale”。
· 第 9.8 节 “创建 /etc/inputrc 文件”。

最后，讨论如何配置 systemd 行为。

· 第 9.10 节 “Systemd 使用和配置”。

9.2. 一般网络配置

本节只适用于需要配置网卡的情况。

9.2.1. 网络接口配置文件

从 209 版本开始，systemd 提供一个名为 **systemd-networkd** 的网络配置守护进程，它能够用于基础网络配置。另外，自 213 版本起，可以用 **systemd-resolved** 代替静态 `/etc/resolv.conf` 文件处理域名解析。这两个服务在默认情况下都是启用的。

注意

如果您不会使用 **systemd-networkd** 进行网络配置 (例如，系统根本没有连接网络，或者您希望使用另外的，NetworkManager 之类的工具进行网络配置)，禁用一项服务，以防止系统引导时出现错误消息：

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

systemd-networkd (以及 **systemd-resolved**) 的配置文件可以放置在 `/usr/lib/systemd/network` 或 `/etc/systemd/network` 中。`/etc/systemd/network` 中的配置文件优先级高于 `/usr/lib/systemd/network` 中的配置文件。有三种类型的配置文件：`.link`、`.netdev` 和 `.network` 文件。如果需要它们的详细描述和内容示例，参阅 systemd.link(5)，systemd.netdev(5)，以及 systemd.network(5) 手册页。

9.2.1.1. 网络设备命名

Udev 一般根据系统物理特征为网卡分配接口名，例如 enp2s1。如果您不确定接口名是什么，可以在引导您的系统后，运行 **ip link** 命令。

注意

接口名依赖于系统正在运行的 udev 守护进程的实现和配置。LFS 的 udev 守护进程 (**systemd-udevd**，在 第 8.76 节 “Systemd-257.8” 中安装) 直到 LFS 系统引导时才会运行。因此，无法在宿主系统中使用上述命令可靠地确定 LFS 系统将会使用的接口名，即使在 chroot 环境中仍然如此。

对于多数系统，每种连接类型只有一个网络接口。例如，有线连接的经典接口名是 eth0，而无线连接的接口名一般是 wifi0 或 wlan0。

如果您偏爱经典或自定义网络接口名，可以使用三种不同方式：

· 覆盖 udev 提供默认策略的 `.link` 文件：

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

· 手动创建命名架构，例如将网络接口命名为 `internet0`，`dmz0`，或者 `lan0`。为此，在 /etc/systemd/network 中创建 `.link` 文件，为您的一个，一些，或者全部网络接口直接选择名称，或选择更好的命名架构。例如：

```
cat > /etc/systemd/network/10-ether0.link << "EOF"
[Match]
# 将 MAC 地址替换为适用于您的网络设备的值
MACAddress=12:34:45:78:90:AB

[Link]
Name=ether0
EOF
```

参阅手册页 systemd.link(5) 获得更多信息。

· 在 /boot/grub/grub.cfg 的内核命令行中传递选项 `net.ifnames=0`。

9.2.1.2. 静态 IP 配置

以下命令为静态 IP 设置创建一个基本的配置文件 (使用 systemd-networkd 和 systemd-resolved)。

```
cat > /etc/systemd/network/10-eth-static.network << "EOF"
[Match]
Name=<网络设备名>

[Network]
Address=192.168.0.2/24
Gateway=192.168.0.1
DNS=192.168.0.1
Domains=<您的域名>
EOF
```

如果您有多个 DNS 服务器，可以在配置文件中创建多个 DNS 项。如果您希望使用静态 `/etc/resolv.conf` 文件，则不要在配置文件中包含 DNS 和 Domains 项。

9.2.1.3. DHCP 配置

以下命令为 IPv4 DHCP 配置创建基本配置文件：

```
cat > /etc/systemd/network/10-eth-dhcp.network << "EOF"
[Match]
Name=<网络设备名>

[Network]
DHCP=ipv4

[DHCPv4]
UseDomains=true
EOF
```

9.2.2. 创建 /etc/resolv.conf 文件

如果要将系统连接到 Internet，它需要某种域名服务 (DNS) 名称解析方式，以将 Internet 域名解析为 IP 地址，或将 IP 地址解析为域名。最好的方法是将 ISP 或网络管理员提供的 DNS 服务器 IP 地址写入 /etc/resolv.conf。

9.2.2.1. systemd-resolved 配置

注意

如果使用与 systemd-resolved 不兼容的方式配置网络接口 (例如 ppp 等)，或使用了某种本地解析器 (如 bind，dnsmasq，或者 unbound 等)，或其他任何生成 /etc/resolv.conf 的软件 (如并非由 systemd 本身提供的 **resolvconf** 程序)，则不应使用 **systemd-resolved** 服务。

如果需要禁用 systemd-resolved，执行命令：

```
systemctl disable systemd-resolved
```

在使用 **systemd-resolved** 进行 DNS 配置时，它创建文件 /run/systemd/resolve/stub-resolv.conf。另外，如果 /etc/resolv.conf 不存在，**systemd-resolved** 会将其创建为指向 /run/systemd/resolve/stub-resolv.conf 的符号链接。因此，不需要手动创建 /etc/resolv.conf。

注意

如果希望为 LFS 系统使用 **systemd-resolved**，但需要在 chroot 环境中访问互联网 (例如，为了构建一个构建过程需要互联网连接的 BLFS 软件包)，需要参照下文给出的静态配置创建 /etc/resolv.conf，才能使得名称解析在 chroot 环境正常工作。在退出 chroot 环境时，可以删除它，这样 **systemd-resolved** 在 LFS 启动时就会将其创建为符号链接。

9.2.2.2. 静态 resolv.conf 配置

如果希望使用静态的 /etc/resolv.conf 执行以下命令创建它：

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

domain <您的域名>
nameserver <您的主要域名服务器 IP 地址>
nameserver <您的次要域名服务器 IP 地址>

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

可以省略 domain 语句，或使用一条 search 语句代替它。阅读 resolv.conf 的手册页了解更多细节。

将 <域名服务器的 IP 地址> 替换为您的网络环境下最合适的 DNS 服务器 IP 地址。这里往往会写入不止一个 DNS 服务器 (需要次要服务器作为后备)。如果您只需要或只希望使用一个 DNS 服务器，可以删除文件中的第二个 nameserver 行。可以写入本地网络路由器的 IP 地址。也可以使用 Google 公用 DNS 服务器作为域名服务器，它们的 IP 地址在下面给出。

注意

Google 公用 DNS 服务器的 IPv4 地址是 8.8.8.8 和 8.8.4.4，IPv6 地址是 2001:4860:4860::8888 和 2001:4860:4860::8844。

9.2.3. 配置系统主机名

在引导过程中，/etc/hostname 被用于设定系统主机名。

执行以下命令，创建 /etc/hostname 文件，并输入一个主机名：

```
echo "<lfs>" > /etc/hostname
```

<lfs> 需要被替换为赋予该计算机的名称。不要在这里输入全限定域名 (FQDN)，它应该被写入 /etc/hosts 文件。

9.2.4. 自定义 /etc/hosts 文件

选择一个全限定域名 (FQDN)，和可能的别名，以供 /etc/hosts 文件使用。如果使用静态 IP 地址，您还需要确定要使用的 IP 地址。hosts 文件条目的语法是：

```
IP_地址 主机名.域名 别名
```

除非该计算机可以从 Internet 访问 (即拥有一个注册域名，并分配了一个有效的 IP 地址段 —— 多数用户没有分配公网 IP)，则使用的 IP 地址必须属于私网 IP 范围。有效的范围是：

```
私网地址范围                              公共前缀长度
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 可以是 16-31 之间的任何数字。y 可以是 0-255 之间的任何数字。

例如，192.168.1.1 是有效的私网 IP 地址。

如果计算机要被配置为能够从 Internet 访问，则可以使用注册域名本身作为有效的 FQDN，或者在其之前用“.”符号连接一个前缀 (前缀通常是主机名) 得到 FQDN。另外，您需要联系域名提供商，以将 FQDN 解析到您的公网 IP 地址。

即使计算机无法从 Internet 访问，一些程序，如 MTA，仍然需要一个 FQDN 才能正常工作。此时可以使用一个特殊的 FQDN localhost.localdomain。

执行以下命令，创建 /etc/hosts 文件：

```
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
```

其中 <192.168.0.2> 和 <FQDN> 需要为特定使用环境和需求进行修改 (如果系统或网络管理员分配了 IP 地址，且本机将被连接到现有的网络中)。可以略去可选的别名 (alias)，如果使用支持 DHCP 或者 IPv6 自动配置的连接，或者使用 localhost.localdomain 作为 FQDN，则包含 <192.168.0.2> 的一行也可以省略。

/etc/hostname 文件不包含 localhost，localhost.localdomain，或者(不包含域名的) 主机名，因为 myhostname NSS 模块会处理它们，参阅手册页 nss-myhostname(8) for details.

::1 是 127.0.0.1 在 IPv6 中的对应，即 IPv6 回环接口。

9.3. 设备和模块管理概述

在第 8 章中，我们在构建 systemd 时安装了 udev 守护程序。在我们详细讨论它的工作原理之前，首先按时间顺序简要介绍历史上曾经使用过的设备管理方式。

传统的 Linux 系统通常静态地创建设备，即在 /dev 下创建大量设备节点 (有时有数千个节点)，无论对应的硬件设备是否真的存在。一般通过 **MAKEDEV** 脚本完成这一工作，它包含以相关的主设备号和次设备号，为世界上可能存在的每个设备建立节点的大量 **mknod** 命令。

使用 udev 时，则只有那些被内核检测到的设备才会获得为它们创建的设备节点。这些设备节点在每次引导系统时都会重新创建；它们被储存在 devtmpfs 文件系统中 (一个虚拟文件系统，完全驻留在系统内存)。设备节点不需要太多空间，它们使用的系统内存可以忽略不计。

9.3.1. 历史

在 2000 年 2 月，一个称为 devfs 的新文件系统被合并到 2.3.46 版内核中，并在 2.4 系列稳定内核中可用。尽管它本身曾经存在于内核源代码中，但这种设备节点动态创建方法从未得到内核核心开发者的大力支持。

devfs 实现机制的主要问题是它处理设备的检测、创建和命名的方式。其中最致命的或许是最后一项，即设备节点命名方式。通常认为，如果设备名称是可配置的，那么设备命名策略应该由系统管理员，而不是某个 (某些) 开发者决定。devfs 还受到其设计中固有的竞争条件的严重影响；在不对内核进行大量修改的前提下无法修复这一问题。由于缺乏维护，它早已被标记为弃用特性，最终在 2006 年 6 月被从内核中移除。

在不稳定的 2.5 系列内核开发过程中，加入了一个新的虚拟文件系统，称为 sysfs，并在 2.6 系列稳定内核中发布。sysfs 的工作是将系统硬件配置信息提供给用户空间进程，有了这个用户空间可见的配置描述，就可能开发一种 devfs 的用户空间替代品。

9.3.2. Udev 实现

9.3.2.1. Sysfs

前面已经简要提到了 sysfs 文件系统。有些读者可能好奇，sysfs 是如何知道系统中存在哪些设备，以及应该为它们使用什么设备号的。答案是，那些编译到内核中的驱动程序在对应设备被内核检测到时，直接将它们注册到 sysfs (内部的 devtmpfs)。对于那些被编译为模块的驱动程序，注册过程在模块加载时进行。只要 sysfs 文件系统被挂载好 (位于 /sys)，用户空间程序即可使用驱动程序注册在 sysfs 中的数据，udev 就能够使用这些数据对设备进行处理 (包括修改设备节点)。

9.3.2.2. 设备节点的创建

内核通过 devtmpfs 直接创建设备文件，任何希望注册设备节点的驱动程序都要通过 devtmpfs (经过驱动程序核心) 进行注册。当一个 devtmpfs 实例被挂载到 /dev 时，设备节点将被以固定的名称、访问权限和所有者首次创建。

很快，内核会向 **udevd** 发送一个 uevent。根据 /etc/udev/rules.d，/usr/lib/udev/rules.d，以及 /run/udev/rules.d 目录中文件指定的规则，**udevd** 将为设备节点创建额外的符号链接，修改其访问权限，所有者，或属组，或者修改该对象的 **udevd** 数据库条目 (名称)。

以上三个目录中的规则都被编号，且这三个目录的内容将合并处理。如果 **udevd** 找不到它正在创建的设备对应的规则，它将会沿用 devtmpfs 最早使用的配置。

9.3.2.3. 模块加载

编译为内核模块的设备驱动程序可能有内建的别名。别名可以通过 **modinfo** 程序查询，它通常和该模块支持的设备的总线相关标识符有关。例如，snd-fm801 驱动程序支持厂商 ID 为 0x1319，设备 ID 为 0x0801 的 PCI 设备，其别名为 pci:v00001319d00000801sv*sd*bc04sc01l*。对于多数设备，总线驱动程序会通过 sysfs 导出应该处理该设备的驱动程序别名，例如 /sys/bus/pci/devices/0000:00:0d.0/modalias 文件应该包含字符串 pci:v00001319d00000801sv00001319sd00001319bc04sc01i00。Udev 附带的默认规则会导致 **udevd** 调用 **/sbin/modprobe** 时传递 MODALIAS uevent 环境变量 (它的值应该和 sysfs 中 modalias 文件的内容相同)，从而加载那些在通配符扩展后别名与这个字符串匹配的模块。

在本例中，这意味着除了 snd-fm801 外，过时 (且不希望) 的 forte 如果可用，也会被加载。之后将介绍防止加载不希望的驱动程序的方法。

内核本身也能够在需要时为网络协议，文件系统，以及 NLS 支持加载模块。

9.3.2.4. 处理热插拔/动态设备

当您插入一个设备，例如通用串行总线 (USB) MP3 播放器时，内核能够发现该设备现在已经被连接到系统，并生成一个 uevent 事件。之后 **udevd** 像前面描述的一样，处理该 uevent 事件。

9.3.3. 加载模块和创建设备时的问题

在自动创建设备节点时，可能出现一些问题。

9.3.3.1. 内核模块没有自动加载

Udev 只加载拥有总线特定别名，且总线驱动程序正确地向 `sysfs` 导出了必要别名的模块。如果情况不是这样，您应该考虑用其他方法加载模块。在 Linux-6.16.1 中，已知 Udev 可以加载编写正确的 INPUT，IDE，PCI，USB，SCSI，SERIO，以及 FireWire 驱动程序。

为了确定您需要的设备驱动程序是否包含 Udev 支持，以模块名为参数运行 **modinfo** 命令。然后试着在 `/sys/bus` 中找到设备对应的目录，并检查其中是否有 `modalias` 文件。

如果 `modalias` 文件存在于 `sysfs` 中，说明驱动程序支持该设备，并能够直接和设备交互，但却没有正确的别名。这是驱动程序的 bug，您需要不通过 Udev 直接加载驱动，并等待这个问题日后被解决。

如果 `modalias` 文件不存在于 `/sys/bus` 下的对应目录中，说明内核开发者尚未对该总线类型增加 modalias 支持。在 Linux-6.16.1 中，ISA 总线不受支持。只能等待这个问题在日后被解决。

Udev 根本不会尝试加载“包装器” 驱动程序，比如 snd-pcm-oss 等，或 loop 等非硬件驱动程序。

9.3.3.2. 内核模块没有自动加载，且 Udev 不尝试加载它

如果“包装器”仅仅用于增强其他模块的功能 (例如，snd-pcm-oss 增强 snd-pcm 的功能，使 OSS 应用程序能够使用声卡)，需要配置 **modprobe**，使其在 Udev 加载被包装的模块时，自动加载包装器。为此，需要将“softdep”行添加到对应的 `/etc/modprobe.d/<filename>.conf` 中。例如：

```
softdep snd-pcm post: snd-pcm-oss
```

注意“softdep”命令也允许 `pre:` 依赖项，或混合使用 `pre:` 和 `post:` 依赖项。参阅 modprobe.d(5) 手册页，了解更多关于“softdep”语法和功能的信息。

9.3.3.3. Udev 加载了不希望的模块

不要构建该模块，或者在 `/etc/modprobe.d/blacklist.conf` 文件中禁用它。以 forte 为例，下面一行禁用了该模块：

```
blacklist forte
```

被禁用的模块仍然可以通过直接执行 **modprobe** 手动加载。

9.3.3.4. Udev 创建了错误的设备或错误的符号链接

这一般是由于一条规则意外地匹配了某个设备。例如，一个写得不好的规则可能同时匹配到 SCSI 磁盘 (正确的) 和对应厂商的 SCSI 通用设备 (不正确的)。找到引起问题的规则，并通过 **udevadm info** 的帮助，将它进一步细化。

9.3.3.5. Udev 规则工作不可靠

这可能是前一个问题的另一个表现形式。如果不是，而且您的规则使用了 `sysfs` 属性，这个问题可能由内核计时问题引发，这类问题需要在新的内核版本中修复。目前，您可以创建一条规则以等待被使用的 `sysfs` 属性，并将它附加到 `/etc/udev/rules.d/10-wait-for-sysfs.rules` 文件中 (如果不存在就创建一个文件)，绕过这个问题。如果您通过这种方法解决了问题，请通知 LFS 开发邮件列表。

9.3.3.6. Udev 没有创建设备

首先，确认驱动程序已经被编译到内核中或作为模块被加载，而且udev 没有创建命名错误的设备。

如果驱动程序没有将它的信息导出到 sysfs，udev 就无法获得创建设备节点必需的信息。这种问题往往出现在内核源代码树以外的第三方驱动程序中。这时，需要在 /usr/lib/udev/devices 中使用正确的主设备号和次设备号，创建一个静态设备节点 (参考内核文档中的 devices.txt 或第三方驱动厂商提供的文档)，该静态设备节点将被复制到 /dev，**udev** 会自动完成复制。

9.3.3.7. 重启后设备命名顺序随机变化

这是由于 Udev 从设计上就是并行加载模块的，因此无法预测加载顺序。这个问题永远也不会被 “修复”。您不应该指望内核提供稳定的设备命名，而是应该创建您自己的规则，以根据设备的一些稳定属性，例如设备序列号或 Udev 安装的一些 *_id 工具的输出，来创建具有稳定名称的符号链接。可以参考第 9.4 节 “管理设备”和第 9.2 节 “一般网络配置”中的例子。

9.3.4. 扩展阅读

以下链接包含了一些额外的帮助文档:

- A Userspace Implementation of devfs http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- The sysfs Filesystem https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

9.4. 管理设备

9.4.1. 处理重复设备

正如第 9.3 节 “设备和模块管理概述”中所述，那些功能相同的设备在 /dev 中的顺序是随机的。例如，如果您有一个 USB 摄像头和一个电视棒，有时 /dev/video0 会指向摄像头，/dev/video1 指向电视棒，而有时在重启后这个顺序正好颠倒过来。对于所有除了声卡和网卡以外的设备，该问题都可以通过建立 udev 规则以创建持久化符号链接来解决。对于网卡的解决方案在第 9.2 节 “一般网络配置”中单独描述，而声卡配置可以在 BLFS 中找到。

对于您的每个可能有这类问题的设备 (即使在您当前使用的 Linux 发行版上并没有问题)，找到 /sys/class 或 /sys/block 中的对应目录。对于视频设备，目录可能是 /sys/class/video4linux/videoX。找出能够唯一确认该设备的属性 (通常是厂商和产品 ID，或者序列号):

```
udevadm info -a -p /sys/class/video4linux/video0
```

然后编写创建符号链接的规则，例如:

```
cat > /etc/udev/rules.d/83-duplicate_devs.rules << "EOF"

# 摄像头和电视棒的持久化符号链接
KERNEL=="video*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f",  ATTRS{vendor}=="0x109e", SYMLINK+="tvtuner"

EOF
```

结果是，/dev/video0 和 /dev/video1 仍然会随机指向电视棒和摄像头 (因此不应直接使用它们)，但符号链接 /dev/tvtuner 和 /dev/webcam 总会指向正确设备。

9.5. 配置系统时钟

本节讨论如何配置 **systemd-timedated** 系统服务，它的作用是配置系统时钟和时区。

如果您不确定您的硬件时钟是否设置为 UTC，运行 **hwclock --localtime --show** 命令，它会显示硬件时钟给出的当前时间。如果这个时间和您的手表显示的一致，则说明硬件时钟被设定为本地时间。相反，如果 **hwclock** 输出的时间不是本地时间，则硬件时钟很可能被设定为 UTC 时间。根据您的时区，在 **hwclock** 显示的时间上加减对应的小时数，进行进一步的验证。例如，如果您现在处于莫斯科时区，即 GMT -0700，在本地时间上加 7 小时，再进行比较。

systemd-timedated 读取 `/etc/adjtime`，并根据其内容将硬件时钟设定为 UTC 或本地时间。

如果您的硬件时钟设置为本地时间，以下列内容创建 `/etc/adjtime` 文件：

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

如果 `/etc/adjtime` 在初次引导时不存在，**systemd-timedated** 会假设硬件时钟使用 UTC，并据此调整该文件。

您也可以使用 **timedatectl** 工具告诉 **systemd-timedated** 您的硬件时钟是 UTC 还是本地时间：

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

timedatectl 也能修改系统时间和时区。

如果要修改系统时间，执行以下命令：

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

硬件时钟也会同时被更新。

要修改当前时区，执行以下命令：

```
timedatectl set-timezone TIMEZONE
```

您可以通过运行以下命令查看可用的时区列表：

```
timedatectl list-timezones
```

注意

请注意 **timedatectl** 命令在 chroot 环境无法工作。只有在使用 systemd 引导 LFS 系统后才能使用它。

9.5.1. 网络时钟同步

从版本 213 开始，systemd 附带了一个名为 **systemd-timesyncd** 的守护程序，可以用于将系统时间与远程 NTP 服务器同步。

该守护程序没有被设计为替代现有成熟的 NTP 守护程序，而是一个仅仅实现了 SNTP 协议的客户端，可以用于一些不太复杂的任务，或是资源紧张的系统。

从 systemd 版本 216 开始，**systemd-timesyncd** 守护进程被默认启用。如果希望禁用它，执行以下命令：

```
systemctl disable systemd-timesyncd
```

可以在 `/etc/systemd/timesyncd.conf` 中修改 **systemd-timesyncd** 使用的服务器。

注意，当系统时钟设定为本地时间时，**systemd-timesyncd** 不会更新硬件时钟。

9.6. 配置 Linux 控制台

本节讨论如何配置 **systemd-vconsole-setup** 系统服务，它负责配置虚拟控制台字体和控制台键盘映射。

systemd-vconsole-setup 服务从 `/etc/vconsole.conf` 文件中读取配置信息。它根据配置确定使用的键映射和控制台字体。一些与特定语言相关的 HOWTO 文档可以帮助您进行配置，参阅 https://tldp.org/HOWTO/HOWTO-INDEX/other-lang.html。浏览 **localectl list-keymaps** 输出的可用控制台键映射列表。在 `/usr/share/consolefonts` 目录中寻找可用的控制台字体。

`/etc/vconsole.conf` 文件的每一行都应该符合格式：`变量名=值`，有效的变量名包括：

KEYMAP
: 该变量指定键盘的键映射表。如果没有设定，默认为 `us`。

KEYMAP_TOGGLE
: 该变量可以用于配置第二切换键盘映射，没有默认设定值。

FONT
: 该变量指定虚拟控制台使用的字体。

FONT_MAP
: 该变量指定控制台字体映射。

FONT_UNIMAP
: 该变量指定 Unicode 字体映射。

在第 9.7 节 “配置系统 Locale”中，我们会指定 `C.UTF-8` 为 Linux 控制台中交互会话的 locale。Kbd 软件包提供的，包含 `C.UTF-8` locale 下程序输出消息中全部字符字形的控制台字体有 `/usr/share/consolefonts` 中的 `LatArCyrHeb*.psfu.gz`，`LatGrkCyr*.psfu.gz`，`Lat2-Terminus16.psfu.gz`，以及 `pancyrillic.f16.psfu.gz` (其他提供的字体缺失一些字符的字形，如 Unicode 左右引号和 Unicode 英文破折号)。因此将它们中的某个，例如 `Lat2-Terminus16.psfu.gz` 设为默认控制台字体：

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

下面的例子可以用于德文键盘和控制台：

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

在系统运行时，可以使用 **localectl** 工具修改 KEYMAP 变量值：

```
localectl set-keymap MAP
```

注意

请注意 **localectl** 命令在 chroot 环境无法工作。只有在使用 systemd 引导 LFS 系统后才能使用它。

也可以通过指定 **localectl** 工具的参数，修改 X11 键盘布局，模型，变体和选项设置：

```
localectl set-x11-keymap 布局 [模型] [变体] [选项]
```

如果需要列出可用的 **localectl set-x11-keymap** 参数值，可以使用下列参数运行 **localectl** 命令：

list-x11-keymap-models
: 列出已知的 X11 键盘映射模型。

list-x11-keymap-layouts
: 列出已知的 X11 键盘映射布局。

list-x11-keymap-variants
列出已知的 X11 键盘映射变体。

list-x11-keymap-options
列出已知的 X11 键盘映射选项。

注意

上面给出的参数都需要 BLFS 中的 XKeyboard-Config 软件包。

9.7. 配置系统 Locale

本地语言支持需要一些环境变量。正确设定它们可以带来以下好处:

· 程序输出被翻译成本地语言

· 字符被正确分类为字母、数字和其他类别，这对于使 **bash** 正确接受命令行中的非 ASCII 本地非英文字符来说是必要的

· 根据所在地区惯例排序字母

· 适用于所在地区的默认纸张尺寸

· 正确格式化货币、时间和日期值

将下面的 `<ll>` 替换为所需语言的双字符代号 (例如“en”)，`<CC>` 替换为国家或地区的双字符代号 (例如 `GB`)，`<charmap>` 替换为您选定的 locale 的标准字符映射。另外，还可以加入 `@euro` 等可选修饰符。

Glibc 支持的所有 locale 可以用以下命令列出:

```
locale -a
```

字符映射可能有多个别名，例如 `ISO-8859-1` 也可以称为 `iso8859-1` 或者 `iso88591`。某些程序不能正确处理一些别名 (例如，`UTF-8` 必须写作 `UTF-8` 才能识别，而不能识别 `utf8`)，因此在多数情况下，为了保险起见，最好使用 locale 的规范名称。为了确定规范名称，执行以下命令，将 `<locale 名>` 替换成 **locale -a** 对于您希望的 locale 的输出 (以 `en_GB.iso88591` 为例)。

```
LC_ALL=<locale 名> locale charmap
```

对于 `en_GB.iso88591` locale，以上命令输出:

```
ISO-8859-1
```

这样就最终确定 locale 应设置为 `en_GB.ISO-8859-1`。在将以上启发方法获得的 locale 添加到 Bash 启动文件之前，一定要进行下列测试:

```
LC_ALL=<locale 名> locale language
LC_ALL=<locale 名> locale charmap
LC_ALL=<locale 名> locale int_curr_symbol
LC_ALL=<locale 名> locale int_prefix
```

以上命令应该输出语言名称，选定 locale 使用的字符编码，本地货币符号，以及所在国家或地区的国际电话区号。如果以上某个命令失败并输出类似下面这样的消息，意味着您的 locale 在第 8 章中没有安装，或者不被 Glibc 的默认安装支持。

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

如果出现了这种消息，您应该用 **localedef** 命令安装所需的 locale，或重新选择一个不同的 locale。后文假设 Glibc 没有输出类似错误消息。

其他软件包在 locale 名不符合它们的期望时可能工作不正常(但未必输出错误消息)。在这种情况下，探索一下其他 Linux 发行版是如何支持您的 locale 的，可以得到一些有用的信息。

在确定了正确的 locale 设置后，创建 /etc/locale.conf 文件：

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

Shell 程序 **/bin/bash** (下文称为 “shell”) 使用一组初始化文件，以帮助创建其运行环境。这些文件有不同的使用场合，可能以不同方式影响登录和交互环境。/etc 中的文件提供全局设定。如果对应的文件同时在用户主目录中存在，它们可能覆盖全局设定。

/bin/login 在用户成功登录后，会读取 /etc/passwd 中指定的 shell，并启动一个交互式登录 shell 进程。而交互式非登录 shell 会在使用命令行直接运行 shell 时 (例如 [prompt]$/**bin/bash**) 启动。非交互式 shell 在运行 shell 脚本时启动。在处理脚本时，shell 不会在执行两条命令的间隔中等待用户输入，因此是非交互的。

登录 shell 往往不会使用 /etc/locale.conf 中的设定。创建 /etc/profile 以读取 /etc/locale.conf 中的 locale 设定值并导出它们以设定您期望的 locale，但是在 Linux 控制台中运行时则使用 C.UTF-8 locale (以防程序输出 Linux console 无法显示的字符)：

```
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
```

修改 /etc/locale.conf 的另一种方法是使用 systemd 的 **localectl** 工具。例如，要使用 **localectl** 完成上面给出的 locale 设置，运行命令：

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

您也可以指定其他语言相关的环境变量，例如 LANG，LC_CTYPE，LC_NUMERIC，或 **locale** 输出的其他环境变量，用空格将它们分割即可。例如，将 LANG 设置为 en_US.UTF-8，LC_CTYPE 设置为 en_US：

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

注意

请注意 **localectl** 命令在 chroot 环境无法工作。只有在使用 systemd 引导 LFS 系统后才能使用它。

C (默认 locale) 和 en_US (推荐美式英语用户使用的 locale) 是不同的。C locale 使用 US-ASCII 7 位字符集，并且将最高位为 1 的字节视为无效字符。因此，**ls** 等命令会将它们替换为问号。另外，如果试图用 Mutt 或 Pine 发送包含这些字符的邮件，会发出不符合 RFC 标准的消息 (发出邮件的字符集会被标为 unknown 8-bit，即“不明 8 位字符集”)。因此，您只能在确信自己永远不会使用 8 位字符时才能使用 C locale。

9.8. 创建 /etc/inputrc 文件

`inputrc` 文件是 Readline 库的配置文件，该库在用户从终端输入命令行时提供编辑功能。它的工作原理是将键盘输入翻译为特定动作。Readline 被 Bash 和大多数其他 shell，以及许多其他程序使用。

多数人不需要 Readline 的用户配置功能，因此以下命令创建全局的 `/etc/inputrc` 文件，供所有登录用户使用。如果您之后决定对于某个用户覆盖掉默认值，您可以在该用户的主目录下创建 `.inputrc` 文件，包含需要修改的映射。

关于更多如何编写 `inputrc` 文件的信息，参考 **info bash** 中 Readline Init File 一节。**info readline** 也是一个很好的信息源。

下面是一个通用的全局 `inputrc` 文件，包含解释一些选项含义的注释。注意注释不能和命令写在同一行。执行以下命令创建该文件：

```
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. 创建 /etc/shells 文件

`shells` 文件包含系统登录 shell 的列表，应用程序使用该文件判断 shell 是否合法。该文件中每行指定一个 shell，包含该 shell 相对于目录树根 (/) 的路径。

例如 **chsh** 使用该文件判断一个非特权用户是否可以修改自己的登录 shell。如果命令没有在 /etc/shell 中找到，就会拒绝修改操作。

这个文件对某些程序是必要的。例如 GDM 在找不到 `/etc/shells` 时不会填充登录界面，FTP 守护进程通常禁止那些使用未在此文件列出的终端的用户登录。

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

/bin/sh
/bin/bash

# End /etc/shells
EOF
```

9.10. Systemd 使用和配置

9.10.1. 基础设置

`/etc/systemd/system.conf` 文件包含一组控制 systemd 基本功能的选项。默认文件中所有条目都被注释掉，并标明了默认值。可以在这里修改日志级别，以及其他一些基本日志设定。参阅 systemd-system.conf(5) 手册页了解每个选项的详细信息。

9.10.2. 禁用引导时自动清屏

Systemd 的默认行为是在引导过程结束时清除屏幕。如果希望的话，您可以运行以下命令，修改这一行为：

```
mkdir -pv /etc/systemd/system/getty@tty1.service.d

cat > /etc/systemd/system/getty@tty1.service.d/noclear.conf << EOF
[Service]
TTYVTDisallocate=no
EOF
```

您总是可以用 `root` 身份运行 **`journalctl -b`** 命令，查阅引导消息。

9.10.3. 禁止将 tmpfs 挂载到 /tmp

默认情况下，`/tmp` 将被挂载 tmpfs 文件系统。如果不希望这样，可以执行以下命令覆盖这一行为：

```
ln -sfv /dev/null /etc/systemd/system/tmp.mount
```

或者，如果希望使用一个单独的 `/tmp` 分区，在 `/etc/fstab` 中为其添加一个条目。

警告

如果使用了单独的 `/tmp` 分区，不要创建上面的符号链接。这会导致根文件系统 (/) 无法重新挂载为可读写，使得系统在引导后不可用。

9.10.4. 配置文件自动创建和删除

有一些创建或删除文件、目录的服务：

- systemd-tmpfiles-clean.service
- systemd-tmpfiles-setup-dev.service
- systemd-tmpfiles-setup.service

它们的系统配置文件位于 /usr/lib/tmpfiles.d/*.conf。本地配置文件位于 /etc/tmpfiles.d。/etc/tmpfiles.d 中的文件覆盖 /usr/lib/tmpfiles.d 中的同名文件。参阅 tmpfiles.d(5) 手册页，了解配置文件格式的细节。

注意 /usr/lib/tmpfiles.d/*.conf 文件的语法较难理解。例如，删除 /tmp 目录下文件的默认规则是文件 /usr/lib/tmpfiles.d/tmp.conf 的一行：

```
q /tmp 1777 root root 10d
```

类别字段 q 表示创建一个带有配额的子卷，它实际上只适用于 btrfs 文件系统。它引用类别 v，类别 v 又引用类别 d (目录)。对于类别 d，会在目录不存在时自动创建它，并根据配置文件调整其权限和所有者。如果 age 参数被指定，该目录中较老的文件会被自动清理。

如果默认参数不符合您的期望，您可以将文件复制到 /etc/tmpfiles.d 目录，再编辑复制得到的副本。例如：

```
mkdir -p /etc/tmpfiles.d
cp /usr/lib/tmpfiles.d/tmp.conf /etc/tmpfiles.d
```

9.10.5. 覆盖系统服务默认行为

Systemd 单元的参数可以通过在 /etc/systemd/system 中创建一个包含配置文件的目录而覆盖。例如：

```
mkdir -pv /etc/systemd/system/foobar.service.d

cat > /etc/systemd/system/foobar.service.d/foobar.conf << EOF
[Service]
Restart=always
RestartSec=30
EOF
```

参阅 systemd.unit(5) 手册页获取更多信息。在创建配置文件后，执行 **systemctl daemon-reload** 和 **systemctl restart foobar**，激活对服务进行的修改。

9.10.6. 调试引导过程

与 SysVinit 或 BSD 风格 init 系统不同，systemd 使用统一格式处理不同种类的引导文件 (或称为单元)。命令 **systemctl** 能够启用、禁用单元文件，或控制、查询单元文件的状态。以下是一些常用的命令：

- **systemctl list-units -t <service> [--all]**: 列出已加载的服务 (service) 类型单元文件。
- **systemctl list-units -t <target> [--all]**: 列出已加载的引导目标 (target) 类型单元文件。
- **systemctl show -p Wants <multi-user.target>**: 显示所有依赖于 multi-user 引导目标的单元，引导目标 (target)是一种和 SysVinit 中运行级别 (runlevel) 地位相同的特殊单元文件。
- **systemctl status <servicename.service>**: 显示名为 servicename 的服务的状态。如果没有同名的其他类型单元文件，可以省略 .service 后缀。其他类型的单元文件有 .socket 文件 (它创建一个监听套接字，提供和 inetd/xinetd 类似的功能)。

9.10.7. 使用 systemd 日志

(默认情况下) 在使用 systemd 引导的系统上，systemd-journald 服务负责处理日志，它取代了传统的 Unix syslog 守护进程。如果您希望的话，也可以添加一个普通 syslog 守护进程，它和 systemd-journald 可以一起工作。systemd-journald 程序将日志项储存为二进制格式，而不是纯文本日志文件。为了解析日志文件，需要使用 systemd 提供的 **journalctl** 命令。下面是该命令的常见用法：

- **journalctl -r**：按时间顺序，倒序显示所有日志内容。
- **journalctl -u UNIT**: 显示与给定单元文件 UNIT 关联的日志。
- **journalctl -b[=ID] -r**: 按时间倒序，显示自上次引导以来 (或编号为 ID 的引导中) 的所有日志。

· **journalctl -f**: 提供类似 tail -f 的功能 (不断将新日志项输出到屏幕)。

9.10.8. 处理核心转储

核心转储在调试崩溃的程序时非常有用，特别是对于守护进程崩溃的情况。在 systemd 引导的系统上，核心转储由 **systemd-coredump** 处理。它会在日志中记录核心转储，并且将核心转储文件本身存储到 `/var/lib/systemd/coredump` 中。如果要获取和处理核心转储文件，可以使用 **coredumpctl** 工具。下面给出它的常用命令的示例：

· **coredumpctl -r**：按时间顺序，倒序显示所有核心转储记录。

· **coredumpctl -1 info**：显示最近一次核心转储的信息。

· **coredumpctl -1 debug**：将最后一次核心转储加载到 GDB 中。

核心转储可能使用大量磁盘空间。为了限制核心转储使用的最大磁盘空间，可以在 `/etc/systemd/coredump.conf.d` 中创建一个配置文件。例如：

```
mkdir -pv /etc/systemd/coredump.conf.d

cat > /etc/systemd/coredump.conf.d/maxuse.conf << EOF
[Coredump]
MaxUse=5G
EOF
```

参阅手册页 systemd-coredump(8)，coredumpctl(1)，以及 coredump.conf.d(5) 了解更多信息。

9.10.9. 持续运行进程

从 systemd 的 230 版本开始，在用户会话结束时，所有用户进程都被杀死，即使使用了 nohup 或 `daemon()` 、`setsid()` 等函数也不例外。这是开发者有意做出的修改，将传统的宽松环境改为更加严格的环境。如果您需要让持续运行的程序 (例如 **screen** 或 **tmux**) 在用户会话结束后保持运行，这项新的行为会导致问题。有三种方法使得这类驻留进程在用户会话结束后继续运行：

· 仅为选定的用户启用进程驻留：普通用户有执行命令 **loginctl enable-linger** 启用进程驻留的权限，管理员可以使用带 `user` 参数的该命令，为特定用户启用进程驻留。在启用进程驻留后，可以使用 **systemd-run** 命令启动持续运行的进程。例如，**systemd-run --scope --user /usr/bin/screen**。如果您为您的用户启用了进程驻留，则 user@.service 将持续运行，甚至在所有登录会话关闭后仍然运行，而且会在系统引导时自动启动。这种方法的好处是可以显式地允许或禁止进程在用户会话结束后继续运行，但却破坏了和 **nohup** 等工具，和使用 `daemon()` 函数的工具的兼容性。

· 为整个系统启用进程驻留：您可以在将 `KillUserProcesses=no` 设置行加入 `/etc/systemd/logind.conf`，为所有用户全局地启用进程驻留。它的好处是允许所有用户继续使用旧方法，但无法进行明确控制。

· 在编译时禁用该功能：您可以在构建 systemd 时传递参数 `-D default-kill-user-process=no` 给 **meson**，使得 systemd 默认启用进程驻留。这完全禁用了 systemd 在会话结束时杀死用户进程的功能。

第 10 章 使 LFS 系统可引导

10.1. 概述

现在应该配置 LFS 系统，使其可以引导了。本章讨论创建 /etc/fstab 文件，为新的 LFS 系统构建内核，以及安装 GRUB 引导加载器，使得系统引导时可以选择进入 LFS 系统。

10.2. 创建 /etc/fstab 文件

一些程序使用 /etc/fstab 文件，以确定哪些文件系统是默认挂载的，和它们应该按什么顺序挂载，以及哪些文件系统在挂载前必须被检查 (确定是否有完整性错误)。参考以下命令，创建一个新的文件系统表:

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

# 文件系统      挂载点         类型      选项                    转储   检查
#                                                                      顺序

/dev/<xxx>     /              <fff>    defaults              1      1
/dev/<yyy>     swap           swap     pri=1                 0      0

# End /etc/fstab
EOF
```

将 <xxx>、<yyy> 和 <fff> 替换为适用于您的系统的值，例如 sda2、sda5 和 ext4。参阅 fstab(5) 了解该文件中 6 个域的详细信息。

在挂载来源于 MS-DOS 或 Windows 的文件系统 (如 vfat、ntfs、smbfs、cifs、iso9660、udf) 时，需要一个特殊的挂载选项 —— utf8，才能正常解析文件名中的非 ASCII 字符。对于非 UTF-8 locale，选项 iocharset 的值应该和您的 locale 字符集设定一致，但改写成内核可以识别的写法。该选项能够正常工作的前提是，将相关的字符集定义 (在内核配置选项的 File Systems -> Native Language Support 子菜单中) 编译到内核中，或构建为内核模块。然而，如果使用了 UTF-8 locale，对应的 iocharset=utf8 会导致文件系统变得大小写敏感。为了避免这个问题，在使用 UTF-8 locale 时，需要用特殊选项 utf8 代替 iocharset=utf8。另外，vfat 和 smbfs 文件系统还需要“codepage”选项，它应该被设定为您的语言在 MS-DOS 下的代码页编号。例如，为了挂载一个 USB 闪存盘，一个 ru_RU.KOI8-R 用户应该在 /etc/fstab 中对应于闪存盘的行添加下列挂载选项:

```
noauto,user,quiet,showexec,codepage=866,iocharset=koi8r
```

相应的，ru_RU.UTF-8 用户应该使用下列选项:

```
noauto,user,quiet,showexec,codepage=866,utf8
```

注意此时使用的 iocharset 默认为 iso8859-1 (这保证文件系统是大小写不敏感的)，而 utf8 选项告诉内核使用 UTF-8 编码转换文件名，这样它们就能在 UTF-8 locale 中被正确解析。

也可以在内核配置中，为一些文件系统指定默认 codepage 和 iocharset 选项值。相关的配置参数名为“Default NLS Option” (CONFIG_NLS_DEFAULT)，“Default Remote NLS Option” (CONFIG_SMB_NLS_DEFAULT)，“Default codepage for FAT” (CONFIG_FAT_DEFAULT_CODEPAGE)，以及“Default iocharset for FAT” (CONFIG_FAT_DEFAULT_IOCHARSET)。无法在编译内核时为 ntfs 文件系统指定这些默认值。

10.3. Linux-6.16.1

Linux 软件包包含 Linux 内核。

估计构建时间: 0.4 - 32 SBU (一般约 2.5 SBU)
需要硬盘空间: 1.7 - 14 GB (一般约 2.3 GB)

10.3.1. 安装内核

构建内核需要三步 —— 配置、编译、安装。阅读内核源代码树中的 README 文件，了解不同于本手册的内核配置方法。

重要

初次构建 Linux 内核是 LFS 构建过程中最具挑战性的环节之一。内核配置依赖于系统硬件和您的个人需求。内核配置包含大约 12,000 个选项，尽管只有约三分之一对于大多数计算机系统是必要的。LFS 编辑建议不熟悉内核编译的用户严格遵循以下步骤。这些步骤的目的是使您能够在第 11.3 节 “重启系统”中重启进入 LFS 系统，并通过命令行登录。这里给出的步骤并不试图优化或定制内核配置。

https://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt 介绍了内核配置的常识。https://anduin.linuxfromscratch.org/LFS/kernel-nutshell/ 提供了更多关于内核配置的信息。这两份文档都有些过时，但仍然较好地概括了配置过程。

如果所有其他尝试都无法解决问题，可以在 lfs-support 邮件列表提问。注意为了防止垃圾邮件，必须先订阅列表才能向列表发送邮件。

运行以下命令，准备编译内核:

```
make mrproper
```

该命令确保内核源代码树绝对干净，内核开发组建议在每次编译内核前运行该命令。尽管内核源代码树在解压后应该是干净的，但这并不完全可靠。

有多种配置内核选项的方法。例如，通常我们通过目录驱动的界面完成这一工作:

```
make menuconfig
```

以上命令中可选的 make 环境变量及含义:

LANG=<host_LANG_value> LC_ALL=
它们根据宿主使用的 locale 建立 locale 设定。在 UTF-8 Linux 文本终端下，有时必须这样做才能正确绘制基于 ncurses 的配置菜单接口。
在这种情况下，一定要将 <host_LANG_value> 替换成宿主环境中的 $LANG 变量值。您也可以使用宿主环境中 $LC_ALL 或 $LC_CTYPE 的值代替。

make menuconfig
这会启动 ncurses 目录驱动的界面。如果希望了解其他 (图形) 界面，可以输入 **make help**。

注意

一个较好的初始内核配置可以通过运行 **make defconfig** 获得。它会考虑您的当前系统体系结构，将基本内核配置设定到较好的状态。

一定要按照以下列表启用/禁用/设定其中列出的内核特性，否则系统可能不能正常工作，甚至根本无法引导:

```
General setup --->
  [ ] Compile the kernel with warnings as errors                        [WERROR]
  CPU/Task time and stats accounting --->
    [*] Pressure stall information tracking                               [PSI]
    [ ]   Require boot parameter to enable pressure stall information tracking
                                                                ...  [PSI_DEFAULT_DISABLED]
  < > Enable kernel headers through /sys/kernel/kheaders.tar.xz      [IKHEADERS]
  [*] Control Group support --->                                       [CGROUPS]
    [*]   Memory controller                                              [MEMCG]
    [ /*] CPU controller --->                                       [CGROUP_SCHED]
      # This may cause some systemd features malfunction:
      [ ] Group scheduling for SCHED_RR/FIFO                      [RT_GROUP_SCHED]
  [ ] Configure standard kernel features (expert users) --->             [EXPERT]

Processor type and features --->
  [*] Build a relocatable kernel                                    [RELOCATABLE]
  [*]   Randomize the address of the kernel image (KASLR)       [RANDOMIZE_BASE]

General architecture-dependent options --->
  [*] Stack Protector buffer overflow detection                 [STACKPROTECTOR]
  [*]   Strong Stack Protector                             [STACKPROTECTOR_STRONG]

[*] Networking support --->                                             [NET]
  Networking options --->
    [*] TCP/IP networking                                               [INET]
    <*>   The IPv6 protocol --->                                        [IPV6]

Device Drivers --->
  Generic Driver Options --->
    [ ] Support for uevent helper                                 [UEVENT_HELPER]
    [*] Maintain a devtmpfs filesystem to mount at /dev                [DEVTMPFS]
    [*]   Automount devtmpfs at /dev, after the kernel mounted the rootfs
                                                                ...  [DEVTMPFS_MOUNT]
    Firmware loader --->
      < /*> Firmware loading facility                                 [FW_LOADER]
      [ ]     Enable the firmware sysfs fallback mechanism
                                                            ...  [FW_LOADER_USER_HELPER]
  Firmware Drivers --->
    [*] Export DMI identification via sysfs to userspace                 [DMIID]
    [*] Mark VGA/VBE/EFI FB as generic system framebuffer       [SYSFB_SIMPLEFB]
  Graphics support --->
    <*>    Direct Rendering Manager (XFree86 4.1.0 and higher DRI support) --->
                                                                        ...  [DRM]
    [*]    Display a user-friendly message when a kernel panic occurs
                                                                    ...  [DRM_PANIC]
    (kmsg)   Panic screen formatter                              [DRM_PANIC_SCREEN]
    Supported DRM clients --->
      [*] Enable legacy fbdev support for your modesetting driver
                                                            ...  [DRM_FBDEV_EMULATION]
    Drivers for system framebuffers --->
      <*> Simple framebuffer driver                                  [DRM_SIMPLEDRM]
    Console display driver support --->
      [*] Framebuffer Console support                          [FRAMEBUFFER_CONSOLE]

File systems --->
  [*] Inotify support for userspace                                 [INOTIFY_USER]
  Pseudo filesystems --->
    [*] Tmpfs virtual memory file system support (former shm fs)           [TMPFS]
    [*]   Tmpfs POSIX Access Control Lists                       [TMPFS_POSIX_ACL]
```

如果在构建 64 位系统，还需要启用一些特性。如果使用 menuconfig 进行配置，需要首先启用 CONFIG_PCI_MSI，然后启用 CONFIG_IRQ_REMAP，最后启用 CONFIG_X86_X2APIC，这是因为只有选定了一个选项的所有依赖项后，该选项才会出现。

```
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]
```

如果 LFS 系统分区在 NVME SSD 上 (即，分区的设备节点是 /dev/nvme*，而非 /dev/sd*)，启用 NVME 支持，否则 LFS 系统无法引导:

```
Device Drivers --->
  NVME Support --->
    <*> NVM Express block device                                    [BLK_DEV_NVME]
```

注意

尽管 “The IPv6 Protocol” (IPv6 协议支持) 并不是严格要求的，但是 systemd 开发者强烈推荐启用它。

根据系统的需求，可能需要一些其他配置选项。BLFS 软件包需要的内核配置选项列表可以在 BLFS 内核配置索引查阅。

注意

如果您的硬件支持 UEFI，且您希望通过 UEFI 引导 LFS 系统，则您需要按照 BLFS 页面的说明，调整一些内核配置选项，**即使您准备使用宿主发行版提供的 UEFI 加载器引导 LFS 系统，也需要进行调整**。

上述配置选项的含义：

`Randomize the address of the kernel image (KASLR)`
为内核映像启用 ASLR，以预防一些基于内核中关键数据或代码的固定地址的攻击。

`Compile the kernel with warnings as errors`
如果使用了和内核开发者不同的编译器和/或配置，启用该选项可能导致构建失败。

`Enable kernel headers through /sys/kernel/kheaders.tar.xz`
启用该选项将会导致构建内核需要 **cpio**。LFS 没有安装 **cpio**。

`Configure standard kernel features (expert users)`
该选项会导致配置界面出现一些新选项，但改变这些选项的设定值可能导致危险后果。不要使用该选项，除非您知道您在做什么。

`Strong Stack Protector`
为内核启用 SSP。我们通过在配置 GCC 时使用 `--enable-default-ssp`，已经为所有用户态代码启用了它，但是内核并不使用 GCC 默认的 SSP 设定。因此我们在这里显式地启用它。

`Support for uevent helper`
如果启用了该选项，它可能干扰 Udev 的设备管理。

`Maintain a devtmpfs`
该选项会使内核自动创建设备节点，即使 Udev 没有运行。Udev 之后才在这些设备节点的基础上运行，管理它们的访问权限并为它们建立符号链接。所有 Udev 用户都需要启用该选项。

`Automount devtmpfs at /dev`
该选项使得内核在切换到根文件系统之后，执行 init 前，将内核获知的设备信息挂载到 /dev。

`Display a user-friendly message when a kernel panic occurs`
该选项使得内核在内核恐慌时正确显示消息。如果不启用它，诊断内核恐慌原因会更加困难：如果没有运行 DRM 驱动，则我们只能使用 VGA 控制台，而它只能显示 24 行，因此有用的内核消息往往已被刷新到屏幕以外；而如果正在运行 DRM 驱动，在内核恐慌时屏幕显示往往会完全错乱。在 Linux-6.12 中，适用于主流 GPU 型号的驱动程序都无法真正支持该选项提供的功能，但在专用的 GPU 驱动加在之前，VESA (或 EFI) 帧缓冲之上运行的简单帧缓冲驱动 (“Simple framebuffer driver”) 能够支持它。如果将专用的 GPU 驱动构建为内核模块 (而非内核映像的一部分)，且没有使用 initramfs，则这一功能在根文件系统成功挂载之前会正常工作，而这足以提供诊断大多数由错误配置 LFS (例如，第 10.4 节 “使用 GRUB 设定引导过程” 中 `root=` 的设定值不正确) 导致的内核恐慌。

`Panic screen formatter`
将该选项设为 `kmsg`，以确保在内核恐慌时显示恐慌前的内核消息。该选项的默认值，`user`，会导致内核只显示一条“用户友好”的，对于诊断毫无帮助的消息。还有一个可选的值 `qr_code`，会使得内核将恐慌前的消息压缩到一个二维码中，并显示该二维码。和文本输出相比，使用外部设备 (如智能手机) 解码二维码能得到更多行内核消息。但二维码输出功能需要 Rust 编译器才能构建，而 LFS 不提供 Rust 编译器。

`Mark VGA/VBE/EFI FB as generic system framebuffer` 和 `Simple framebuffer driver`
它们允许使用 VESA 帧缓冲 (或者如果使用 UEFI 引导 LFS 系统，则使用 EFI 帧缓冲) 作为 DRM 设备。GRUB 会设置好 VESA 帧缓冲 (在使用 UEFI 时，EFI 固件会设置好 EFI 帧缓冲)，这样在 GPU 专用的 DRM 驱动加载前，基于 DRM 的内核恐慌处理程序也能正常工作。

`Enable legacy fbdev support for your modesetting driver` 和 `Framebuffer Console support`
它们允许使用基于 DRI (Direct Rendering Infrastructure) 的 GPU 驱动显示 Linux 控制台。鉴于已经启用了 `CONFIG_DRM` (Direct Rendering Manager)，需要同时启用这两项特性，否则一旦 DRI 驱动被加载，就只能看到空白屏幕。

`Support x2apic`
支持以 x2APIC 模式运行 64 位 x86 处理器的中断控制器。64 位 x86 系统的固件可能启用了 x2APIC，此时未启用该选项的内核在引导时会发生内核恐慌。该选项在固件禁用 x2APIC 时没有作用，但无害。

某些情况下，**make oldconfig** 更为合适。阅读 `README` 文件了解更多信息。

如果希望的话，也可以将宿主系统的内核配置文件 `.config` 拷贝到解压出的 `linux-6.16.1` 目录 (前提是可以找到该文件)。然而我们不推荐这样做，一般来说，浏览整个配置目录，并从头创建内核配置是更好的选择。

编译内核映像和模块：

```
make
```

如果要使用内核模块，可能需要在 `/etc/modprobe.d` 中写入模块配置。讨论模块和内核配置的信息位于第 9.3 节 “设备和模块管理概述”和 `linux-6.16.1/Documentation` 目录下的内核文档中。另外 modprobe.d(5) 也可以作为参考。

如果内核配置使用了模块，安装它们：

```
make modules_install
```

在内核编译完成后，需要进行额外步骤完成安装，一些文件需要拷贝到 `/boot` 目录中。

小心

如果要使用单独的 /boot 分区 (包括和宿主发行版共用一个 /boot 分区的情况)，需要将这些文件拷贝到该分区中。最简单的方法是首先在 /etc/fstab 中加入 /boot 分区的条目 (详见前一节)，然后在 chroot 环境中以 root 身份执行以下命令:

```
mount /boot
```

该命令中省略了指向设备节点的路径，因为 **mount** 可以从 /etc/fstab 中读取它。

指向内核映像的路径可能随机器平台的不同而变化。下面使用的文件名可以依照您的需要改变，但文件名的开头应该保持为 vmlinuz，以保证和下一节描述的引导过程自动设定相兼容。下面的命令假定机器是 x86 体系结构:

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-6.16.1-lfs-12.4-systemd
```

System.map 是内核符号文件，它将内核 API 的每个函数入口点和运行时数据结构映射到它们的地址。它被用于调查分析内核可能出现的问题。执行以下命令安装该文件:

```
cp -iv System.map /boot/System.map-6.16.1
```

内核配置文件 .config 由上述的 **make menuconfig** 步骤生成，包含编译好的内核的所有配置选项。最好能将它保留下来以供日后参考:

```
cp -iv .config /boot/config-6.16.1
```

安装 Linux 内核文档:

```
cp -r Documentation -T /usr/share/doc/linux-6.16.1
```

需要注意的是，在内核源代码目录中可能有不属于 root 的文件。在以 root 身份解压源代码包时 (就像我们在 chroot 环境中所做的那样)，这些文件会获得它们之前在软件包创建者的计算机上的用户和组 ID。这一般不会造成问题，因为在安装后通常会删除源代码目录树。然而，Linux 源代码目录树一般会被保留较长时间，这样创建者当时使用的用户 ID 就可能被分配给本机的某个用户，导致该用户拥有内核源代码的写权限。

注意

之后在 BLFS 中安装软件包时往往需要修改内核配置。因此，和其他软件包不同，我们在安装好内核后可以不移除源代码树。

如果要保留内核源代码树，对内核源代码目录运行 **chown -R 0:0** 命令，以保证 linux-6.16.1 目录中所有文件都属于 root。

如果正在使用保留的内核源码树和新的内核配置重新构建内核，则一般不>应该执行 **make mrproper** 命令。该命令会清理 .config 文件和之前构建的所有 .o 文件。尽管可以从 /boot 中的副本恢复 .config，清理所有 .o 文件仍然会造成巨大的浪费：对于简单的内核配置变更，通常只需要 (重新) 构建少数几个 .o 文件，在不进行清理时，内核构建系统会正确跳过其他 .o 文件。

然而，如果更新了 GCC，则应该执行 **make clean** 命令，以清理之前构建的所有 .o 文件，否则可能导致构建失败。

警告

有的内核文档建议创建符号链接 /usr/src/linux 指向内核源代码目录，这仅仅适用于 2.6 系列之前的内核。在 LFS 系统上绝对不要创建它，因为在构建完基本 LFS 系统后，它可能在您构建其他软件包时引起问题。

10.3.2. 配置 Linux 内核模块加载顺序

多数情况下 Linux 内核模块可以自动加载，但有时需要指定加载顺序。负责加载内核模块的程序 **modprobe** 和 **insmod** 从 `/etc/modprobe.d` 下的配置文件中读取加载顺序，例如，如果 USB 驱动程序 (ehci_hcd、ohci_hcd 和 uhci_hcd) 被构建为模块，则必须按照先加载 echi_hcd，再加载 ohci_hcd 和 uhci_hcd 的正确顺序，才能避免引导时出现警告信息。

为此，执行以下命令创建文件 `/etc/modprobe.d/usb.conf`：

```
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 的内容

安装的文件: config-6.16.1, vmlinuz-6.16.1-lfs-12.4-systemd, 以及 System.map-6.16.1
安装的目录: /lib/modules 和 /usr/share/doc/linux-6.16.1

简要描述

`config-6.16.1`	包含所有内核配置选项的值
`vmlinuz-6.16.1-lfs-12.4-systemd`	Linux 系统的引擎，在启动计算机时，它是操作系统中被最早加载的部分。它检测并初始化计算机硬件，将它们以目录树的形式提供给软件，并将单个 CPU 封装成多任务系统，使多个用户程序看上去在同时执行
`System.map-6.16.1`	地址和符号列表；它将内核函数和数据结构映射为入口点和地址

10.4. 使用 GRUB 设定引导过程

注意

如果您的系统支持 UEFI，且您希望通过 UEFI 引导 LFS，您应该跳过本页中的指令，但仍然阅读本页以学习 `grub.cfg` 的语法和在该文件中指定分区的方法，并按照 BLFS 页面中的说明，配置支持 UEFI 的 GRUB。

10.4.1. 概述

警告

如果您不小心错误地配置了 GRUB，可能导致您的系统完全无法使用，除非使用 CD-ROM 或可引导的 USB 存储器等备用引导设备。本节不是引导您的 LFS 系统的唯一方案，您可能只要修改现有的启动加载器 (如 Grub-Legacy、GRUB2 或 LILO) 配置即可引导 LFS。

您务必保证自己拥有一个紧急引导磁盘，它在计算机不可用 (无法引导) 时能够 “抢修” 计算机。如果您现在还没有引导设备，您可以执行以下命令创建一个。在运行下列命令前，您需要跳到 BLFS，安装包含 **`xorriso`** 的 libisoburn 软件包：

```
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 命名惯例

GRUB 使用一种独特的命名结构，为驱动器和分区命名。分区名的形式为 (hdn,m)，这里 n 是硬盘驱动器编号，m 是分区编号。硬盘驱动器编号从 0 开始，但分区号对于主分区来说从 1 开始 (对于扩展分区来说从 5 开始)。例如，分区 `sda1` 在 GRUB 中的名字是 (hd0,1)，而 `sdb3` 的名字是 (hd1,3)。和 Linux 不同，GRUB 不认为 CD-ROM 驱动器属于硬盘驱动器。例如，如果在 `hdb` 上有一个 CD-ROM 驱动器，而 `hdc` 上有第二个硬盘驱动器，则第二个硬盘驱动器仍然名为 hd1。

10.4.3. 设定 GRUB 配置

GRUB 的工作方式是，将数据写入硬盘的第一个物理磁道。这里不属于任何文件系统，在启动时，第一个物理磁道中的程序从引导分区加载 GRUB 模块，默认在 /boot/grub 中查找模块。

引导分区的位置由负责进行配置的用户自己决定，作者推荐创建一个小的 (建议大小为 200 MB) 分区，专门存放引导信息。这样，不同的 Linux 系统 (无论是 LFS 还是商业发行版) 在启动时和启动后都能访问相同的引导文件。如果您选择这样做，您需要挂载这个单独的分区，将 `/boot` 中已有的文件 (例如上一节中构建的 Linux 内核) 移动到新的分区中。之后，解除该分区的挂载，并将它挂载为 `/boot`。另外，还要注意更新 `/etc/fstab`。

直接将 `/boot` 目录保留在 LFS 分区也是可以的，但这样在配置多系统启动时比较麻烦。

根据以上信息，确定 LFS 根分区 (或 boot 分区，如果使用了独立的 boot 分区) 的名称。下面假设 LFS 根分区 (或 boot 分区) 是 `sda2`。

将 GRUB 文件安装到 `/boot/grub` 并设定引导磁道：

警告

以下命令会覆盖当前启动引导器，如果这不是您希望的，不要运行该命令。例如，如果您使用第三方启动引导器管理主引导记录 (MBR)。

```
grub-install /dev/sda
```

注意

如果系统是使用 UEFI 引导的，**grub-install** 会试图为 x86_64-efi 目标安装文件，但它们并未在第 8 章中安装。如果出现了这类问题，请在以上命令中添加 `--target i386-pc` 选项。

10.4.4. 创建 GRUB 配置文件

生成 `/boot/grub/grub.cfg`：

```
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 6.16.1-lfs-12.4-systemd" {
        linux   /boot/vmlinuz-6.16.1-lfs-12.4-systemd root=/dev/sda2 ro
}
EOF
```

insmod 命令加载 GRUB 模块 `part_gpt` and `ext2`。`ext2` 模块尽管如此命名，实际上却支持 `ext2`，`ext3`，以及 `ext4` 文件系统。**grub-install** 命令已经将一些模块嵌入 GRUB 主映像 (它安装在 MBR 或 GRUB BIOS 分区中)，以便在访问其他模块 (这些模块在 `/boot/grub/i386-pc` 中) 时避免“先有鸡还是先有蛋”的问题。因此，在典型的系统配置中，这两个模块已经被嵌入主映像，此时这两条 **insmod** 命令不会产生任何效果。但是无论如何它们不会造成损害，而且在一些少见的系统配置中它们可能是必要的。

set gfxpayload=1024x768x32 命令设置 VESA 帧缓冲的分辨率和色深。内核 SimpleDRM 驱动需要它才能使用 VESA 帧缓冲。可以自行改用更适合显示器的分辨率和色深值。

注意

从 GRUB的视角来看，内核文件的位置相对于它使用的分区。如果您使用了单独的 /boot 分区，需要从上面的 linux 行删除 /boot，然后修改 set root 行，指向 /boot 分区。

注意

如果新增或移除了一些存储设备 (包括 USB 闪存盘等可移动存储设备)，则 GRUB 赋予分区的编号可能发生改变。这可能导致引导失败，因为 `grub.cfg` 仍然在使用“旧的”编号。如果希望避免这种问题，可以使用分区和文件系统的 UUID 指定分区，以代替 GRUB 编号。运行 **lsblk -o UUID,PARTUUID,PATH,MOUNTPOINT** 以显示文件系统 (在 `UUID` 列) 和分区 (在 `PARTUUID` 列) 的 UUID。之后将 `set root=(hdx,y)` 替换为 `search --set=root --fs-uuid <内核所在文件系统的 UUID>`，并将 `root=/dev/sda2` 替换为 `root=PARTUUID=<构建 LFS 使用的分区的 UUID>`。

注意分区的 UUID 和该分区中文件系统的 UUID 是完全不同的。一些在线资料可能建议使用 `root=UUID=<文件系统 UUID>` 代替`root=PARTUUID=<分区 UUID>`，但是这种方法依赖于 initramfs，而 initramfs 超出了 LFS 的范畴。

`/dev` 中分区对应的设备节点名也可能发生改变 (和 GRUB 分区编号的变化相比较为少见)。在 `/etc/fstab` 中，也可以将 `/dev/sda1` 这样的设备节点路径改为 `PARTUUID=<分区 UUID>`，从而避免设备节点命名发生改变时可能导致的引导失败。

GRUB 是一个很强大的程序，它提供了非常多的选项，可以支持多种设备、操作系统和分区类型，还有很多用于定制启动屏幕、声音、鼠标输入等的选项。这些选项的细节超过了本书的范围，不予讨论。

小心

有一个命令 grub-mkconfig 被用于自动创建配置文件。它使用 /etc/grub.d 中的脚本创建新配置文件，这会覆盖您手动编写的 grub.cfg。这些脚本主要是为非源代码发行版设计的，在 LFS 中不推荐使用。但是，如果您安装了商业发行版，它很可能在发行版中被运行，记得备份 grub.cfg 以防它被覆盖。

第 11 章 收尾工作

11.1. 收尾工作

很好！现在新的 LFS 系统已经安装好了！我们祝愿您的全新的，自定义的 Linux 系统能够成功启动！

创建一个 `/etc/lfs-release` 文件似乎是一个好主意。通过使用它，您 (或者我们，如果您向我们寻求帮助的话) 能够容易地找出当前安装的 LFS 系统版本。运行以下命令创建该文件：

```
echo 12.4-systemd > /etc/lfs-release
```

后续安装在系统上的软件包可能需要两个描述当前安装的系统的文件，这些软件包可能是二进制包，也可能是需要构建的源代码包。

第一个文件根据 Linux Standards Base (LSB) 的规则描述系统状态。运行命令创建该文件：

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="12.4-systemd"
DISTRIB_CODENAME="<your name here>"
DISTRIB_DESCRIPTION="Linux From Scratch"
EOF
```

第二个文件基本上包含相同的信息，systemd 和一些图形桌面环境会使用它。运行命令创建该文件：

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="12.4-systemd"
ID=lfs
PRETTY_NAME="Linux From Scratch 12.4-systemd"
VERSION_CODENAME="<your name here>"
HOME_URL="https://www.linuxfromscratch.org/lfs/"
RELEASE_TYPE="stable"
EOF
```

您需要修改 'DISTRIB_CODENAME' 和 'VERSION_CODENAME' 域，体现您的系统的独特性。

11.2. 增加 LFS 用户计数

现在您已经完成了本书中的构建过程，那么问题来了，您希望被计为一名 LFS 用户吗？前往 https://www.linuxfromscratch.org/cgi-bin/lfscounter.php 并输入您的姓名和第一次使用的 LFS 版本，即可注册成为 LFS 用户。

下面重启计算机进入 LFS。

11.3. 重启系统

现在所有软件包已经安装完成，可以重新启动计算机了。但是，还有一些需要检查的事项。下面是一些建议：

- 安装固件，如果您的设备的内核驱动程序需要一些固件文件才能正常工作的话。
- 确认已经为 `root` 用户设置了密码。
- 此时可以再次检查一些配置文件。
  - /etc/fstab
  - /etc/hosts
  - /etc/inputrc

- /etc/profile
- /etc/resolv.conf (可选)
- /etc/vimrc

现在，正如我们之前保证的，您可以引导全新的 LFS 系统了！首先退出 chroot 环境：

```
logout
```

解除虚拟文件系统的挂载：

```
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
```

如果为 LFS 创建了其他的分区，需要在解除挂载 LFS 分区之前，先解除挂载它们，例如：

```
umount -v $LFS/home
umount -v $LFS
```

解除 LFS 文件系统的挂载：

```
umount -v $LFS
```

现在重新启动系统。

如果 GRUB 引导加载器如同本书前文所述安装，并配置正确，GRUB 目录应该已经配置为自动引导 LFS 12.4-systemd 启动项。

重新启动后，就可以使用 LFS 系统了。您会看到一个简单的 “login” 提示符。此时，您可以阅读 BLFS 手册，它包含安装更多您所需软件的方法。

如果**没能**重启成功，则需要进行故障排除。阅读 https://www.linuxfromscratch.org/lfs/troubleshooting.html 以获得更多系统启动阶段出现问题的解决方法。

11.4. 附加资源

感谢您阅读本书。我们希望您觉得本书对您有用，而且您能够从构建系统的过程中学到一些知识。

现在 LFS 系统已经安装好了，您可能想问“然后呢？”为了解答这个问题，我们为您汇集了一份有用资源的列表。

- 维护

  所有软件都会定期发布 Bug 报告和安全注意事项。由于 LFS 系统是从源代码构建的，您必须自己留意它们。有一些跟踪它们的在线网站，下面列出一些：

  - LFS 安全公告

    这是 LFS 手册发布后发现的安全缺陷的列表。

  - Open Source Security 邮件列表

    该邮件列表用于讨论安全缺陷和相关概念，以及开源社区中与安全相关的实践。

- LFS Hints

  LFS Hints 是一组由 LFS 社区志愿者提交的帮助文档，它位于 https://www.linuxfromscratch.org/hints/downloads/files/。

· 邮件列表

如果您需要帮助，希望跟踪 LFS 开发进度，或者希望参与该项目，访问第 1 章 - 邮件列表了解一下 LFS 邮件列表。

· Linux 文档计划 (The Linux Documentation Project)

Linux 文档计划 (The Linux Documentation Project, TLDP) 的目标是通过协作解决 Linux 文档的所有问题，它包含大量 HOWTO 文档、指南和手册页。它的网址是 https://tldp.org/。

11.5. 开始使用新构建的 LFS

11.5.1. 决定在新系统中做什么

现在 LFS 已经构建完成，而且可以引导。那么，要在新的系统中做什么呢？首先要确定准备使用新的系统达成什么目的。通常来说，计算机系统可以分成两大类：工作站和服务器。当然，这两类系统并不是完全互斥的。用于这两类系统的应用程序完全可以安装在同一个系统上，但是我们首先分别考虑这两类系统。

服务器是较为简单的一类系统。通常它需要包含一个网页服务程序，例如 Apache HTTP 服务器，和一个数据库服务器，例如 MariaDB。当然，服务器也可能提供其他服务，嵌入式系统也可以视为服务器。

另一方面，工作站系统较为复杂。它通常需要一个图形用户环境，例如 LXDE，XFCE，KDE，或者 Gnome，这些用户环境又基于基本的图形环境和其他图形界面应用，例如 Firefox 浏览器，Thunderbird 邮件客户端，或 LibreOffice 办公套件。这些应用需要很多 (如果需要比较完备的功能，可能需要几百个) 额外的软件包中的应用和库的支持。

另外，一些应用提供系统管理功能，它们适用于所有系统。这些应用的安装方法都在 BLFS 手册中给出。其中，并非所有软件包在所有环境下都必须安装。例如，dhcpcd 对于服务器一般并不合适，wireless_tools 一般只用于笔记本电脑。

11.5.2. 在基本的 LFS 环境中工作

一旦启动 LFS 系统，就可以使用它提供的工具构建更多软件包。然而，基本 LFS 系统的交互环境十分简略，可以使用一些方法改善工作环境：

11.5.2.1. 从 LFS 宿主系统中通过 chroot 进行操作

这种方法能够提供完整的图形环境，包括功能完整的网页浏览器，而且可以进行复制/粘贴操作。而且可以使用宿主系统的 wget 等应用程序下载软件包，并将其放入 chroot 环境能够访问的位置。

为了在 chroot 环境中正常构建软件包，如果虚拟文件系统尚未挂载，需要挂载它们。为此，可以在**宿主**系统中创建一个脚本：

```
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
```

注意脚本中的后三条命令被注释掉了。如果它们涉及的目录是宿主系统中的挂载点，而且在 LFS/BLFS 系统中也会同样挂载它们，可以使用这些命令。

可以用普通用户 (推荐) 或者 `root` 身份，执行 **bash ~/mount-virt.sh** 以运行该脚本。如果是以普通用户身份运行它，宿主系统需要安装 sudo。

另外，该脚本给出了一些可能用于存放下载的软件包文件的位置。存放位置可以是任意的。例如，可以存放在某个用户的主目录中，例如 ~/sources，或者使用一个所有用户可访问的位置，如 /usr/src。我们建议不要将 BLFS 使用的源代码包和 LFS 使用的源代码包混合存放在 /sources (chroot 环境中的路径) 中。无论如何，必须使用 chroot 环境中能够访问的位置。

最后，可以编码进入 chroot 环境的过程，以更方便地完成这一工作。可以在宿主系统中用户的 ~/.bashrc 文件中加入命令别名设定：

```
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'
```

别名设定看上去比较复杂，这是由于引号和反斜线符号的嵌套使用。为了美观，上面给出的命令分成了两行，但实际输入该命令时必须将其写在一行内。

11.5.2.2. 使用 ssh 远程工作

这种方法同样允许使用完整的图形环境，但首先需要在 LFS 系统上安装 sshd (通常可以在 chroot 中安装它)。另外，这种方法需要第二台计算机。这种方法相比于使用 chroot 环境更简单。而且，它在构建软件包时使用为 LFS 系统构建的内核，却仍然能够像使用 chroot 一样在一个完整的系统上工作。

可以使用 **scp** 命令将软件包源代码上传到 LFS 系统。如果希望直接在 LFS 系统上下载源代码，可以在 chroot 环境中安装 libtasn1，p11-kit，make-ca，以及 wget (或者在引导 LFS 系统后用 **scp** 上传它们的源代码)。

11.5.2.3. 使用 LFS 系统的命令行工作

这种方法需要先使用 chroot 安装 libtasn1，p11-kit，make-ca，wget，gpm，以及 links (或者 lynx)，然后重启计算机，进入新的 LFS 系统。LFS 系统默认有六个虚拟控制台。可以使用 **Alt**+**Fx** 组合键切换控制台，其中 **Fx** 是 **F1** 到 **F6** 之间的键。**Alt**+← 和 **Alt**+→ 组合键也可以切换控制台。

此时，可以在两个不同的虚拟控制台登录，在其中一个运行 links 或 lynx 浏览器，另一个运行 bash。GPM 允许使用鼠标左键选择并复制浏览器中的命令，然后可以切换终端，将命令粘贴到另一个终端中运行。

注意

顺带一提，从 X 窗口环境也可以用 **Ctrl**+**Alt**+**Fx** 组合键切换虚拟终端，但是不能在图形接口和虚拟终端之间进行复制操作。也可以用 **Ctrl**+**Alt**+**Fx** 组合键返回窗口环境，其中 **Fx** 一般是 **F1**，但也可能是 **F7**。

第 V 部分 附录

附录 A. 缩写和术语

ABI 应用程序二进制接口 (Application Binary Interface)
ALFS Automated Linux From Scratch
API 应用程序编程接口 (Application Programming Interface)
ASCII 美国标准信息交换代码 (American Standard Code for Information Interchange)
BIOS 基本输入输出系统 (Basic Input/Output System)
BLFS Beyond Linux From Scratch
BSD Berkeley 软件发行版 (Berkeley Software Distribution)
chroot 切换根目录 (change root)
CMOS 互补金属氧化物半导体 (Complementary Metal Oxide Semiconductor)
COS 服务类型 (Class Of Service)
CPU 中央处理器 (Central Processing Unit)
CRC 循环冗余检查 (Cyclic Redundancy Check)
CVS 并行版本系统 (Concurrent Versions System)
DHCP 动态主机配置协议 (Dynamic Host Configuration Protocol)
DNS 域名服务 (Domain Name Service)
EGA 增强图形适配器 (Enhanced Graphics Adapter)
ELF 可执行与可链接格式 (Executable and Linkable Format)
EOF 文件结束 (End of File)
EQN 公式 (equation)
ext2 第二代增强文件系统 (second extended file system)
ext3 第三代增强文件系统 (third extended file system)
ext4 第四代增强文件系统 (fourth extended file system)
FAQ 常见问题 (Frequently Asked Questions)
FHS 文件系统目录结构标准 (Filesystem Hierarchy Standard)
FIFO 先进先出 (First-In, First Out)
FQDN 全限定域名 (Fully Qualified Domain Name)
FTP 文件传输协议 (File Transfer Protocol)
GB 吉字节 (Gigabytes)
GCC GNU 编译器集合 (GNU Compiler Collection)
GID 组标识符 (Group Identifier)
GMT 格林尼治标准时间 (Greenwich Mean Time)
HTML 超文本标记语言 (Hypertext Markup Language)
IDE 集成驱动电子设备 (Integrated Drive Electronics)
IEEE 电子电气工程师学会 (Institute of Electrical and Electronic Engineers)
IO 输入/输出 (Input/Output)
IP 因特网协议 (Internet Protocol)
IPC 进程间通信 (Inter-Process Communication)

IRC 互联网中继聊天 (Internet Relay Chat)
ISO 国际标准化组织 (International Organization for Standardization)
ISP 互联网服务提供商 (Internet Service Provider)
KB 千字节 (Kilobytes)
LED 发光二极管 (Light Emitting Diode)
LFS Linux From Scratch
LSB Linux 标准规范 (Linux Standard Base)
MB 兆字节 (Megabytes)
MBR 主引导记录 (Master Boot Record)
MD5 消息摘要算法第五版 (Message Digest 5)
NIC 网络接口卡 (Network Interface Card)
NLS 本地语言支持 (Native Language Support)
NNTP 网络新闻传输协议 (Network News Transport Protocol)
NPTL 原生 POSIX 线程库 (Native POSIX Threading Library)
OSS 开放音频系统 (Open Sound System)
PCH 预编译头文件 (Pre-Compiled Headers)
PCRE Perl 兼容的正则表达式 (Perl Compatible Regular Expression)
PID 进程标识符 (Process Identifier)
PTY 伪终端 (pseudo terminal)
QOS 服务质量 (Quality of Service)
RAM 随机访问存储器 (Random Access Memory)
RPC 远程过程调用 (Remote Procedure Call)
RTC 实时时钟 (Real Time Clock)
SBU 标准构建单位 (Standard Build Unit)
SCO Santa Cruz 作业公司 (The Santa Cruz Operation)
SHA1 安全散列算法第一版 (Secure-Hash Algorithm 1)
TLDP Linux 文档计划 (The Linux Documentation Project)
TFTP 简单文件传输协议 (Trivial File Transfer Protocol)
TLS 线程本地存储 (Thread-Local Storage)
UID 用户标识符 (User Identifier)
umask 用户文件创建掩码 (user file-creation mask)
USB 通用串行总线 (Universal Serial Bus)
UTC 协调世界时 (Coordinated Universal Time)
UUID 通用唯一识别码 (Universally Unique Identifier)
VC 虚拟控制台 (Virtual Console)
VGA 视频图形阵列 (Video Graphics Array)
VT 虚拟终端 (Virtual Terminal)

附录 B. 致谢

我们希望感谢以下人员和组织对 Linux From Scratch 项目作出的贡献:

- Gerard Beekmans <gerard@linuxfromscratch.org> – LFS 创始人
- Bruce Dubbs <bdubbs@linuxfromscratch.org> – LFS 执行编辑
- Jim Gifford <jim@linuxfromscratch.org> – CLFS 共同负责人
- Pierre Labastie <pierre@linuxfromscratch.org> – BLFS 编辑及 ALFS 负责人
- DJ Lucas <dj@linuxfromscratch.org> – LFS 和 BLFS 编辑
- Ken Moffat <ken@linuxfromscratch.org> – BLFS 编辑
- 在 LFS 和 BLFS 的相关邮件列表中还有无数朋友，他们为本书进行提供了宝贵的建议，对本书中的安装说明进行了测试，提供了问题报告和安装说明，还分享了在安装各种软件包时获得的宝贵经验。他们的工作使得本书得以发布。

翻译人员

- Manuel Canales Esparcia <macana@macana-es.com> – 西班牙语 LFS 翻译项目
- Johan Lenglet <johan@linuxfromscratch.org> – 2008 年以前的 LFS 法语翻译项目
- Jean-Philippe Mengual <jmengual@linuxfromscratch.org> – 2008-2016 年的 LFS 法语翻译项目
- Julien Lepiller <jlepiller@linuxfromscratch.org> – 2017 年后的 LFS 法语翻译项目
- Anderson Lizardo <lizardo@linuxfromscratch.org> – 曾经的葡萄牙语 LFS 翻译项目
- Jamenson Espindula <jafesp@gmail.com> – 2022 年后的葡萄牙语 LFS 翻译项目
- Thomas Reitelbach <tr@erdfunkstelle.de> – 德语 LFS 翻译项目

镜像站维护者

北美镜像站

- Scott Kveton <scott@osuosl.org> – lfs.oregonstate.edu 镜像站
- William Astle <lost@l-w.net> – ca.linuxfromscratch.org 镜像站
- Eujon Sellers <jpolen@rackspace.com> – lfs.introspeed.com 镜像站
- Justin Knierim <tim@idge.net> – lfs-matrix.net 镜像站

南美镜像站

- Manuel Canales Esparcia <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info 镜像站
- Luis Falcon <Luis Falcon> – torredehanoi.org 镜像站

欧洲镜像站

- Guido Passet <guido@primerelay.net> – nl.linuxfromscratch.org 镜像站
- Bastiaan Jacques <baafie@planet.nl> – lfs.pagefault.net 镜像站
- Sven Cranshoff <sven.cranshoff@lineo.be> – lfs.lineo.be 镜像站
- Scarlet Belgium – lfs.scarlet.be 镜像站
- Sebastian Faulborn <info@aliensoft.org> – lfs.aliensoft.org 镜像站

- Stuart Fox <stuart@dontuse.ms> – lfs.dontuse.ms 镜像站
- Ralf Uhlemann <admin@realhost.de> – lfs.oss-mirror.org 镜像站
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org 镜像站
- Fredrik Danerklint <fredan-lfs@fredan.org> – se.linuxfromscratch.org 镜像站
- Franck <franck@linuxpourtous.com> – lfs.linuxpourtous.com 镜像站
- Philippe Baque <baque@cict.fr> – lfs.cict.fr 镜像站
- Vitaly Chekasin <gyouja@pilgrims.ru> – lfs.pilgrims.ru 镜像站
- Benjamin Heil <kontakt@wankoo.org> – lfs.wankoo.org 镜像站
- Anton Maisak <info@linuxfromscratch.org.ru> – linuxfromscratch.org.ru 镜像站

亚洲镜像站

- Satit Phermsawang <satit@wbac.ac.th> – lfs.phayoune.org 镜像站
- Shizunet Co.,Ltd. <info@shizu-net.jp> – lfs.mirror.shizu-net.jp 镜像站

澳大利亚镜像站

- Jason Andrade <jason@dstc.edu.au> – au.linuxfromscratch.org 镜像站

曾经的项目组成员

- Christine Barczak <theladyskye@linuxfromscratch.org> – LFS 手册编辑
- Archaic <archaic@linuxfromscratch.org> – LFS 技术作家/编辑，HLFS 项目领导者，BLFS 编辑，Hints 和补丁项目维护者
- Matthew Burgess <matthew@linuxfromscratch.org> – LFS 项目领导者，LFS 技术作家/编辑
- Nathan Coulson <nathan@linuxfromscratch.org> – LFS-Bootscripts 维护者
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- Jeroen Coumans <jeroen@linuxfromscratch.org> – 网站开发者，FAQ 维护者
- Manuel Canales Esparcia <manuel@linuxfromscratch.org> – LFS/BLFS/HLFS XML 和 XSL 维护者
- Alex Groenewoud – LFS 技术作家
- Marc Heerdink
- Jeremy Huntwork <jhuntwork@linuxfromscratch.org> – LFS 技术作家，LFS LiveCD 维护者
- Bryan Kadzban <bryan@linuxfromscratch.org> – LFS 技术作家
- Mark Hymers
- Seth W. Klein – FAQ 维护者
- Nicholas Leippe <nicholas@linuxfromscratch.org> – Wiki 维护者
- Anderson Lizardo <lizardo@linuxfromscratch.org> – 网站后台脚本维护者
- Randy McMurchy <randy@linuxfromscratch.org> – BLFS 项目领导者，LFS 编辑
- Dan Nicholson <dnicholson@linuxfromscratch.org> – LFS 和 BLFS 编辑
- Alexander E. Patrakov <alexander@linuxfromscratch.org> – LFS 技术作家，LFS 国际化编辑，LFS LiveCD 维护者

· Simon Perreault
· Scot Mc Pherson <scot@linuxfromscratch.org> – LFS NNTP 网关维护者
· Douglas R. Reno <renodr@linuxfromscratch.org> – Systemd 编辑
· Ryan Oliver <ryan@linuxfromscratch.org> – CLFS 项目共同负责人
· Greg Schafer <gschafer@zip.com.au> – LFS 技术作家，新一代启用 64 位构建方法设计者
· Jesse Tie-Ten-Quee – LFS 技术作家
· James Robertson <jwrober@linuxfromscratch.org> – Bugzilla 维护者
· Tushar Teredesai <tushar@linuxfromscratch.org> – BLFS 手册编辑，Hints 和补丁计划领导者
· Jeremy Utley <jeremy@linuxfromscratch.org> – LFS 技术作家，Bugzilla 维护者，LFS-Bootscripts 维护者
· Zack Winkles <zwinkles@gmail.com> – LFS 技术作家

附录 C. 依赖关系

LFS 中构建的每个软件包都依赖于一个或多个其他软件包，才能正确地构建和安装。某些软件包甚至存在循环依赖，即第一个软件包依赖于第二个软件包，而第二个软件包反过来又依赖第一个。由于这些依赖关系的存在，在 LFS 中构建软件包的顺序非常关键。本页面的目的就是记录 LFS 中每个软件包构建时的依赖关系。

对于我们构建的每个软件包，我们都列出了三到五种依赖关系。第一种列出了编译和安装该软件包需要的其他软件包。第二种列出了在运行该软件包提供的程序或库时必要的软件包。第三种列出了不属于第一种情况，但在运行该软件包测试套件时需要的软件包。第四种列出了在构建和安装前，需要该软件包已经构建并安装到最终位置的其他软件包。

最后一种依赖关系是 LFS 中没有提到的可选软件包，但它们对用户可能很有用。这些软件包本身可能还有必要或可选的依赖关系。对于这些依赖关系，推荐的方法是在完成 LFS 手册后，安装可选依赖项，再重新构建相关的 LFS 软件包。BLFS 提到了几个软件包的重新安装方法。

Acl

安装依赖于: Attr, Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, 以及 Texinfo
运行时依赖于: Attr 和 Glibc
测试依赖于: Automake, Diffutils, Findutils, 以及 Libtool
必须在下列软件包之前安装: Coreutils, Sed, Tar, 以及 Vim
可选依赖项: 无

Attr

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed, 以及 Texinfo
运行时依赖于: Glibc
测试依赖于: Automake, Diffutils, Findutils, 以及 Libtool
必须在下列软件包之前安装: Acl, Libcap, 以及 Patch
可选依赖项: 无

Autoconf

安装依赖于: Bash, Coreutils, Grep, M4, Make, Perl, Sed, 以及 Texinfo
运行时依赖于: Bash, Coreutils, Grep, M4, Make, Sed, 以及 Texinfo
测试依赖于: Automake, Diffutils, Findutils, GCC, 以及 Libtool
必须在下列软件包之前安装: Automake 和 Coreutils
可选依赖项: Emacs

Automake

安装依赖于: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed, 以及 Texinfo
运行时依赖于: Bash, Coreutils, Grep, M4, Sed, 以及 Texinfo
测试依赖于: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool, 以及 Tar
必须在下列软件包之前安装: Coreutils
可选依赖项: 无

Bash

安装依赖于: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed, 以及 Texinfo

运行时依赖于: Glibc, Ncurses, 以及 Readline

测试依赖于: Expect 和 Shadow

必须在下列软件包之前安装: 无

可选依赖项: Xorg

Bc

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, 以及 Readline

运行时依赖于: Glibc, Ncurses, 以及 Readline

测试依赖于: Gawk

必须在下列软件包之前安装: Linux

可选依赖项: 无

Binutils

安装依赖于: Bash, Binutils, Coreutils, Diffutils, File, Flex, Gawk, GCC, Glibc, Grep, Make, Perl, Pkgconf, Sed, Texinfo, Zlib, 以及 Zstd

运行时依赖于: Glibc, Zlib, 以及 Zstd

测试依赖于: DejaGNU 和 Expect

必须在下列软件包之前安装: 无

可选依赖项: Elfutils 和 Jansson

Bison

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, 以及 Sed

运行时依赖于: Glibc

测试依赖于: Diffutils, Findutils, 以及 Flex

必须在下列软件包之前安装: Kbd 和 Tar

可选依赖项: Doxygen

Bzip2

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, 以及 Patch

运行时依赖于: Glibc

测试依赖于: 无

必须在下列软件包之前安装: File 和 Libelf

可选依赖项: 无

Coreutils

安装依赖于: Autoconf, Automake, Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Libcap, Make, OpenSSL, Patch, Perl, Sed, 以及 Texinfo
运行时依赖于: Glibc
测试依赖于: Diffutils, E2fsprogs, Findutils, Shadow, 以及 Util-linux
必须在下列软件包之前安装: Bash, Diffutils, Findutils, Man-DB, 以及 Systemd
可选依赖项: Expect.pm 和 IO::Tty

D-Bus

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Pkgconf, Sed, Systemd, 以及 Util-linux
运行时依赖于: Glibc 和 Systemd
测试依赖于: 一些 BLFS 软件包
必须在下列软件包之前安装: 无
可选依赖项: Xorg 库

DejaGNU

安装依赖于: Bash, Coreutils, Diffutils, Expect, GCC, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Expect 和 Bash
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Diffutils

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Glibc
测试依赖于: Perl
必须在下列软件包之前安装: 无
可选依赖项: 无

E2fsprogs

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Pkgconf, Sed, Systemd, Texinfo, 以及 Util-linux
运行时依赖于: Glibc 和 Util-linux
测试依赖于: Procps-ng 和 Psmisc
必须在下列软件包之前安装: 无
可选依赖项: 无

Expat

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, 以及 Sed
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: Python 和 XML::Parser
可选依赖项: 无

Expect

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed, 以及 Tcl
运行时依赖于: Glibc 和 Tcl
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: Tk

File

安装依赖于: Bash, Binutils, Bzip2, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Xz, 以及 Zlib
运行时依赖于: Glibc, Bzip2, Xz, 以及 Zlib
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: libseccomp

Findutils

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Bash 和 Glibc
测试依赖于: DejaGNU, Diffutils, 以及 Expect
必须在下列软件包之前安装: 无
可选依赖项: 无

Flex

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed, 以及 Texinfo
运行时依赖于: Bash, Glibc, 以及 M4
测试依赖于: Bison 和 Gawk
必须在下列软件包之前安装: Binutils, IProute2, Kbd, Kmod, 以及 Man-DB
可选依赖项: 无

Flit-Core

安装依赖于: Python
运行时依赖于: Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Packaging 和 Wheel
可选依赖项: pytest 和 testpath

Gawk

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, MPFR, Patch, Readline, Sed, 以及 Texinfo
运行时依赖于: Bash, Glibc, 以及 Mpfr
测试依赖于: Diffutils
必须在下列软件包之前安装: 无
可选依赖项: libsigsegv

GCC

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, GMP, Grep, M4, Make, MPC, MPFR, Patch, Perl, Sed, Tar, Texinfo, 以及 Zstd
运行时依赖于: Bash, Binutils, Glibc, Mpc, 以及 Python
测试依赖于: DejaGNU, Expect, 以及 Shadow
必须在下列软件包之前安装: 无
可选依赖项: GDC, GNAT, 以及 ISL

GDBM

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, 以及 Sed
运行时依赖于: Bash, Glibc, 以及 Readline
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Gettext

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed, 以及 Texinfo
运行时依赖于: Acl, Bash, Gcc, 以及 Glibc
测试依赖于: Diffutils, Perl, 以及 Tcl
必须在下列软件包之前安装: Automake 和 Bison
可选依赖项: libunistring 和 libxml2

Glibc

安装依赖于: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API Headers, Make, Perl, Python, Sed, 以及 Texinfo
运行时依赖于: 无
测试依赖于: File
必须在下列软件包之前安装: 无
可选依赖项: 无

GMP

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, M4, Make, Sed, 以及 Texinfo
运行时依赖于: GCC 和 Glibc
测试依赖于: 无
必须在下列软件包之前安装: MPFR 和 GCC
可选依赖项: 无

Gperf

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, 以及 Make
运行时依赖于: GCC 和 Glibc
测试依赖于: Diffutils 和 Expect
必须在下列软件包之前安装: 无
可选依赖项: 无

Grep

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, 以及 Texinfo
运行时依赖于: Glibc
测试依赖于: Gawk
必须在下列软件包之前安装: Man-DB
可选依赖项: PCRE2 和 libsigsegv

Groff

安装依赖于: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed, 以及 Texinfo
运行时依赖于: GCC, Glibc, 以及 Perl
测试依赖于: 无
必须在下列软件包之前安装: Man-DB
可选依赖项: ghostscript 和 Uchardet

GRUB

安装依赖于: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Texinfo, 以及 Xz
运行时依赖于: Bash, GCC, Gettext, Glibc, Xz, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Gzip

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Bash 和 Glibc
测试依赖于: Diffutils 和 Less
必须在下列软件包之前安装: Man-DB
可选依赖项: 无

Iana-Etc

安装依赖于: Coreutils
运行时依赖于: 无
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Perl
可选依赖项: 无

Inetutils

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo, 以及 Zlib
运行时依赖于: GCC, Glibc, Ncurses, 以及 Readline
测试依赖于: 无
必须在下列软件包之前安装: Tar
可选依赖项: 无

Intltool

安装依赖于: Bash, Gawk, Glibc, Make, Perl, Sed, 以及 XML::Parser
运行时依赖于: Autoconf, Automake, Bash, Glibc, Grep, Perl, 以及 Sed
测试依赖于: Perl
必须在下列软件包之前安装: 无
可选依赖项: 无

IProute2

安装依赖于: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, Libcap, Libelf, Linux API 头文件, Pkgconf, 以及 Zlib
运行时依赖于: Bash, Coreutils, Glibc, Libcap, Libelf, 以及 Zlib
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: Berkeley DB, iptables, libbpf, libmnl, 以及 libtirpc

Jinja2

安装依赖于: MarkupSafe, Python, Setuptools, 以及 Wheel
运行时依赖于: MarkupSafe 和 Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Systemd
可选依赖项: 无

Kbd

安装依赖于: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, 以及 Sed
运行时依赖于: Bash, Coreutils, 以及 Glibc
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: Linux-PAM

Kmod

安装依赖于: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, OpenSSL, Pkgconf, Sed, Xz, 以及 Zlib
运行时依赖于: Glibc, Xz, 以及 Zlib
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Systemd
可选依赖项: scdoc (用于生成手册页)

Less

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 以及 Sed
运行时依赖于: Glibc 和 Ncurses
测试依赖于: 无
必须在下列软件包之前安装: Gzip
可选依赖项: PCRE2 或 PCRE

Libcap

安装依赖于: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Perl, Make, 以及 Sed
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: IProute2 和 Shadow
可选依赖项: Linux-PAM

Libelf

安装依赖于: Bash, Binutils, Bzip2, Coreutils, GCC, Glibc, Make, Xz, Zlib, 以及 Zstd
运行时依赖于: Bzip2, Glibc, Xz, Zlib, 以及 Zstd
测试依赖于: 无
必须在下列软件包之前安装: IProute2 和 Linux
可选依赖项: 无

Libffi

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Make, 以及 Sed
运行时依赖于: Glibc
测试依赖于: DejaGnu
必须在下列软件包之前安装: Python
可选依赖项: 无

Libpipeline

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Glibc
测试依赖于: Check 和 Pkgconf
必须在下列软件包之前安装: Man-DB
可选依赖项: 无

Libtool

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, 以及 SedAutoconf, Automake, Bash, Binutils, Coreutils, File, GCC, Glibc, Grep, Make, 以及 Sed
测试依赖于: Autoconf, Automake, 以及 Findutils
必须在下列软件包之前安装: 无
可选依赖项: 无

Libxcrypt

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Perl, 以及 Sed
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: Perl, Python, Shadow, 以及 Systemd
可选依赖项: 无

Linux

安装依赖于: Bash, Bc, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Kmod, Libelf, Make, Ncurses, OpenSSL, Perl, 以及 Sed
运行时依赖于: 无
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: cpio, LLVM (包括 Clang), 以及 Rust-bindgen

Linux API 头文件

安装依赖于: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Perl, 以及 Sed
运行时依赖于: 无
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

Lz4

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, 以及 Make
运行时依赖于: Glibc
测试依赖于: Python
必须在下列软件包之前安装: Zstd 和 Systemd
可选依赖项: 无

M4

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Bash 和 Glibc
测试依赖于: Diffutils
必须在下列软件包之前安装: Autoconf 和 Bison
可选依赖项: libsigsegv

Make

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Glibc
测试依赖于: Perl 和 Procps-ng
必须在下列软件包之前安装: 无
可选依赖项: Guile

Man-DB

安装依赖于: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, GDBM, Gettext, Glibc, Grep, Groff, Gzip, Less, Libpipeline, Make, Pkgconf, Sed, Systemd, 以及 Xz
运行时依赖于: Bash, GDBM, Groff, Glibc, Gzip, Less, Libpipeline, 以及 Zlib
测试依赖于: Util-linux
必须在下列软件包之前安装: 无
可选依赖项: libseccomp 和 po4a

Man-Pages

安装依赖于: Bash, Coreutils, Make, 以及 Sed
运行时依赖于: 无
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

MarkupSafe

安装依赖于: Python, Setuptools, 以及 Wheel
运行时依赖于: Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Jinja2
可选依赖项: 无

Meson

安装依赖于: Ninja, Python, Setuptools, 以及 Wheel
运行时依赖于: Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Systemd
可选依赖项: 无

MPC

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, MPFR, Sed, 以及 Texinfo
运行时依赖于: Glibc, GMP, 以及 MPFR
测试依赖于: 无
必须在下列软件包之前安装: GCC
可选依赖项: 无

MPFR

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, Sed, 以及 Texinfo
运行时依赖于: Glibc 和 GMP
测试依赖于: 无
必须在下列软件包之前安装: Gawk 和 GCC
可选依赖项: 无

Ncurses

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch, 以及 Sed
运行时依赖于: Glibc
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Bash, GRUB, Inetutils, Less, Procps-ng, Psmisc, Readline, Texinfo, Util-linux, 以及 Vim
可选依赖项: 无

Ninja

安装依赖于: Binutils, Coreutils, GCC, 以及 Python
运行时依赖于: GCC 和 Glibc
测试依赖于: cmake
必须在下列软件包之前安装: Meson
可选依赖项: Asciidoc, Doxygen, Emacs, 以及 re2c

OpenSSL

安装依赖于: Binutils, Coreutils, Gcc, Make, 以及 Perl
运行时依赖于: Glibc 和 Perl
测试依赖于: 无
必须在下列软件包之前安装: Coreutils, Kmod, Linux, 以及 Systemd
可选依赖项: 无

Packaging

安装依赖于: Flit-core 和 Python
运行时依赖于: Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Wheel
可选依赖项: pytest

Patch

安装依赖于: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, 以及 Sed
运行时依赖于: Attr 和 Glibc
测试依赖于: Diffutils
必须在下列软件包之前安装: 无
可选依赖项: Ed

Perl

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Libxcrypt, Make, Sed, 以及 Zlib
运行时依赖于: GDBM, Glibc, 以及 Libxcrypt
测试依赖于: Iana-Etc, Less, 以及 Procps-ng
必须在下列软件包之前安装: Autoconf
可选依赖项: Berkeley DB

Pkgconf

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, 以及 Sed
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: Binutils, D-Bus, E2fsprogs, IProute2, Kmod, Man-DB, Procps-ng, Python, Systemd, 以及 Util-linux
可选依赖项: 无

Procps-ng

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses, Pkgconf, 以及 Systemd
运行时依赖于: Glibc
测试依赖于: DejaGNU
必须在下列软件包之前安装: 无
可选依赖项: 无

Psmisc

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, 以及 Sed
运行时依赖于: Glibc 和 Ncurses
测试依赖于: Expect
必须在下列软件包之前安装: 无
可选依赖项: 无

Python

安装依赖于: Bash, Binutils, Coreutils, Expat, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Libxcrypt, Make, Ncurses, OpenSSL, Pkgconf, Sed, 以及 Util-linux
运行时依赖于: Bzip2, Expat, Gdbm, Glibc, Libffi, Libxcrypt, Ncurses, OpenSSL, 以及 Zlib
测试依赖于: GDB 和 Valgrind
必须在下列软件包之前安装: Ninja
可选依赖项: Berkeley DB, libnsl, SQLite, 以及 Tk

Readline

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, 以及 Texinfo
运行时依赖于: Glibc 和 Ncurses
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Bash, Bc, 以及 Gawk
可选依赖项: 无

Sed

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
运行时依赖于: Acl, Attr, 以及 Glibc
测试依赖于: Diffutils 和 Gawk
必须在下列软件包之前安装: E2fsprogs, File, Libtool, 以及 Shadow
可选依赖项: 无

Setuptools

安装依赖于: Python 和 Wheel
运行时依赖于: Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Jinja2, MarkupSafe, 以及 Meson
可选依赖项: 无

Shadow

安装依赖于: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Libcap, Libxcrypt, Make, 以及 Sed
运行时依赖于: Glibc 和 Libxcrypt
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Coreutils
可选依赖项: CrackLib 和 Linux-PAM

Systemd

安装依赖于: Acl, Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Gperf, Grep, Jinja2, Libcap, Libxcrypt, Lz4, Meson, OpenSSL, Pkgconf, Sed, Util-linux, 以及 Zstd
运行时依赖于: Acl, Glibc, Libcap, Libxcrypt, OpenSSL, Util-linux, Xz, Zlib, 以及 Zstd
测试依赖于: 无
必须在下列软件包之前安装: D-Bus, E2fsprogs, Man-DB, Procps-ng, 以及 Util-linux
可选依赖项: 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, PCRE2, pefile, Polkit, pyelftools, qemu, qrencode, quota-tools, rpm, rsync, SELinux, Sphinx, systemtap, tpm2-tss, Valgrind, Xen, 以及 zsh

Tar

安装依赖于: Acl, Attr, Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, 以及 Texinfo
运行时依赖于: Acl, Attr, Bzip2, Glibc, Gzip, 以及 Xz
测试依赖于: Autoconf, Diffutils, Findutils, Gawk, 以及 Gzip
必须在下列软件包之前安装: 无
可选依赖项: 无

Tcl

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, 以及 Sed
运行时依赖于: Glibc 和 Zlib
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Texinfo

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, 以及 Sed
运行时依赖于: Glibc 和 Ncurses
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Util-linux

安装依赖于: Bash, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Pkgconf, Sed, Systemd, 以及 Zlib
运行时依赖于: Glibc, Ncurses, Readline, Systemd, 以及 Zlib
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: Asciidoctor, Libcap-NG, libeconf, libuser, libutempter, Linux-PAM, smartmontools, po4a, 以及 slang

Vim

安装依赖于: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 以及 Sed
运行时依赖于: Acl, Attr, Glibc, Python, Ncurses, 以及 Tcl
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: Xorg, GTK+2, LessTif, Ruby, 以及 GPM

Wheel

安装依赖于: Python, Flit-core, 以及 packaging
运行时依赖于: Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Jinja2, MarkupSafe, Meson, 以及 Setuptools
可选依赖项: 无

XML::Parser

安装依赖于: Bash, Binutils, Coreutils, Expat, GCC, Glibc, Make, 以及 Perl
运行时依赖于: Expat, Glibc, 以及 Perl
测试依赖于: Perl
必须在下列软件包之前安装: Intltool
可选依赖项: LWP::UserAgent

Xz

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, 以及 Make.
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: File, GRUB, Kmod, Libelf, Man-DB, 以及 Systemd
可选依赖项: 无

Zlib

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Make, 以及 Sed
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: File, Kmod, Libelf, Perl, 以及 Util-linux
可选依赖项: 无

Zstd

安装依赖于: Binutils, Coreutils, GCC, Glibc, Gzip, Lz4, Make, Xz, 以及 Zlib
运行时依赖于: Glibc
测试依赖于: 无
必须在下列软件包之前安装: Binutils, GCC, Libelf, 以及 Systemd
可选依赖项: 无

附录 D. LFS 授权许可

本书按照 Creative Commons Attribution-NonCommercial-ShareAlike 2.0 许可协议的规定授权使用。

从本书中能够提取的计算机指令根据 MIT 许可协议的规定授权使用。

D.1. Creative Commons License

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. The MIT License

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.

索引

软件包

程序

库

脚本

其他

/usr/include/drm/*.h: 47，47
/usr/include/linux/*.h: 47，47
/usr/include/misc/*.h: 47，47
/usr/include/mtd/*.h: 47，47
/usr/include/rdma/*.h: 47，47
/usr/include/scsi/*.h: 47，47
/usr/include/sound/*.h: 47，47
/usr/include/video/*.h: 47，47
/usr/include/xen/*.h: 47，47
/var/log/btmp: 76
/var/log/lastlog: 76
/var/log/wtmp: 76
/etc/shells: 238
man pages: 92，92
Systemd 自定义设置: 239