

Linux 从零开始

版本 12.3

2025 年 3 月 5 日发布

创建者：杰拉德·比克曼斯
执行编辑：布鲁斯·杜布斯

Linux 从零开始：第 12.3 版：2025 年 3 月 5 日发布

作者：杰拉德·比克曼斯 主编：布鲁斯·杜布斯 版权所有 © 1999-2025

杰拉德·比克曼斯

版权所有 © 1999-2025，杰拉德·比克曼斯

保留所有权利。

本书采用知识共享许可协议授权。

书中涉及的计算机指令可依据 MIT 许可证提取使用。

Linux®是 Linus Torvalds 的注册商标。

目录

前言 viii i. 序言 viii ii. 读者对象 viii iii. LFS 目标架构 ix iv. 先决条件 x v. LFS 与标准 x vi. 本书选用软件包的理由 xi vii. 排版约定 xvii viii. 结构说明 xviii ix. 勘误与安全公告 xix I. 简介 1 1.1. 如何构建 LFS 系统 2

1.2. 自上一版本以来的更新内容 2
1.3. 更新日志 4
1.4. 资源 8
1.5. 帮助 9
II. 构建前的准备工作 11 2. 准备宿主系统 12 2.1. 简介 12

2.2. 宿主系统要求 12
2.3. 分阶段构建 LFS 系统 15
2.4. 创建新分区 15
2.5. 在分区上创建文件系统 17
2.6. 设置\$LFS 变量与 umask 值 18
2.7. 挂载新分区 19
3. 软件包与补丁 21 3.1. 简介 21
3.2. 所有软件包 22
3.3. 所需补丁 31
4. 最终准备工作 32 4.1. 简介 32

4.2. 在 LFS 文件系统中创建有限的目录结构 32
4.3. 添加 LFS 用户 32
4.4. 设置环境 33
4.5. 关于 SBUs (标准构建单位) 36
4.6. 关于测试套件 36

第三部分 构建 LFS 交叉工具链与临时工具 38 重要预备知识 39

i. 引言 39
ii. 工具链技术说明 39
iii. 通用编译指南 44
5. 编译交叉工具链 46 5.1. 简介 46

5.2. Binutils-2.44 - 第 1 遍 47

5.3. GCC-14.2.0 - 第 1 遍编译 49	5.4. Linux-6.13.4 API 头文件 52	
5.5. Glibc-2.41		53
5.6. 来自 GCC-14.2.0 的 Libstdc++ 56		
6. 交叉编译临时工具 58		
6.1. 简介 58		
6.2. M4-1.4.19 59		
6.3. Ncurses-6.5 60		
6.4. Bash-5.2.37 62		
6.5. Coreutils-9.6 63		
6.6. Diffutils-3.11 64		
6.7. File-5.46 65		
6.8. Findutils-4.10.0 66		
6.9. Gawk-5.3.1 67		
6.10. Grep-3.11 68		
6.11. Gzip-1.13 69		
6.12. Make-4.4.1 70		
6.13. Patch-2.7.6 71		
6.14. Sed-4.9 72		
6.15. Tar-1.35 73		
6.16. Xz-5.6.4 74		
6.17. Binutils-2.44 - 第 2 阶段 75		
6.18. GCC-14.2.0 - 第 2 阶段 76		
7. 进入 Chroot 环境并构建额外临时工具 78	7.1. 简介 78	
7.2. 更改所有权 78		
7.3. 准备虚拟内核文件系统 78		
7.4. 进入 Chroot 环境 79		
7.5. 创建目录 80		
7.6. 创建基础文件与符号链接 81		
7.7. Gettext-0.24 84		
7.8. Bison-3.8.2 85		
7.9. Perl-5.40.1 86		
7.10. Python-3.13.2 87		
7.11. Texinfo-7.2 88		
7.12. Util-linux-2.40.4 89		
7.13. 清理并保存临时系统 90		
IV. 构建 LFS 系统 92	8. 安装基本系统软件 93	8.1. 简介 93
8.2. 软件包管理 94		
8.3. 手册页-6.12 98		
8.4. Iana-Etc-20250123 99	8.5. Glibc-2.41	
8.6. Zlib-1.3.1 108		100
8.7. Bzip2-1.0.8 109		
8.8. Xz-5.6.4 111		

- 8.9. Lz4-1.10.0 113
- 8.10. Zstd-1.5.7 114
- 8.11. File-5.46 115
- 8.12. Readline-8.2.13 116
- 8.13. M4-1.4.19 118
- 8.14. Bc-7.0.3 119
- 8.15. Flex-2.6.4 120
- 8.16. Tcl-8.6.16 121
- 8.17. Expect-5.45.4 第 123 页
- 8.18. DejaGNU-1.6.3 第 125 页
- 8.19. Pkgconf-2.3.0 第 126 页
- 8.20. Binutils-2.44 第 127 页
- 8.21. GMP-6.3.0 130
- 8.22. MPFR-4.2.1 132
- 8.23. MPC-1.3.1 133
- 8.24. Attr-2.5.2 134
- 8.25. Acl-2.3.2 第 135 页
- 8.26. Libcap-2.73 第 136 页
- 8.27. Libxcrypt-4.4.38 第 137 页
- 8.28. Shadow-4.17.3 第 139 页
- 8.29. GCC-14.2.0 第 143 页
- 8.30. Ncurses-6.5 第 149 页
- 8.31. Sed-4.9 第 152 页
- 8.32. Psmisc-23.7 第 153 页
- 8.33. Gettext-0.24 154
- 8.34. Bison-3.8.2 156
- 8.35. Grep-3.11 157
- 8.36. Bash-5.2.37 158
- 8.37. Libtool-2.5.4 160
- 8.38. GDBM-1.24 161
- 8.39. Gperf-3.1 162
- 8.40. Expat-2.6.4 163
- 8.41. Inetutils-2.6 164
- 8.42. Less-668 166
- 8.43. Perl-5.40.1 167
- 8.44. XML::Parser-2.47 170
- 8.45. Intltool-0.51.0 171
- 8.46. Autoconf-2.72 172
- 8.47. Automake-1.17 173
- 8.48. OpenSSL-3.4.1 174
- 8.49. Elfutils-0.192 中的 Libelf 176
- 8.50. Libffi-3.4.7 177
- 8.51. Python-3.13.2 178
- 8.52. Flit-Core-3.11.0 181
- 8.53. Wheel-0.45.1 182
- 8.54. Setuptools-75.8.1 183
- 8.55. Ninja-1.12.1 184

- 8.56. Meson-1.7.0 185
 - 8.57. Kmod-34 186
 - 8.58. Coreutils-9.6 187
 - 8.59. Check-0.15.2 192
 - 8.60. Diffutils-3.11 第 193 页
 - 8.61. Gawk-5.3.1 第 194 页
 - 8.62. Findutils-4.10.0 第 195 页
 - 8.63. Groff-1.23.0 第 196 页
 - 8.64. GRUB-2.12 第 199 页
 - 8.65. Gzip-1.13 第 201 页
 - 8.66. IPRoute2-6.13.0 第 202 页
 - 8.67. Kbd-2.7.1 第 204 页
 - 8.68. Libpipeline-1.5.8 206
 - 8.69. Make-4.4.1 207
 - 8.70. Patch-2.7.6 208
 - 8.71. Tar-1.35 209
 - 8.72. Texinfo-7.2 210
 - 8.73. Vim-9.1.1166 212
 - 8.74. MarkupSafe-3.0.2 215
 - 8.75. Jinja2-3.1.5 216
 - 8.76. Systemd-257.3 中的 Udev 217
 - 8.77. Man-DB-2.13.0 220
 - 8.78. Procs-ng-4.0.5 223
 - 8.79. Util-linux-2.40.4 225
 - 8.80. E2fsprogs-1.47.2 第 230 页
 - 8.81. Sysklogd-2.7.0 第 233 页
 - 8.82. SysVinit-3.14 第 234 页
 - 8.83. 关于调试符号 第 235 页
 - 8.84. 清理调试符号 235
 - 8.85. 清理工作 237
9. 系统配置 238 9.1. 简介 238

- 9.2. LFS-Bootscripts-20240825 239
- 9.3. 设备与模块处理概述 241
- 9.4. 设备管理 244
- 9.5. 通用网络配置 247
- 9.6. System V 启动脚本使用与配置 249
- 9.7. 配置系统区域设置 257
- 9.8. 创建/etc/inputrc 文件 259
- 9.9. 创建/etc/shells 文件 260

10. 使 LFS 系统可启动 262 10.1. 简介 262

- 10.2. 创建 /etc/fstab 文件 262
- 10.3. Linux-6.13.4 264
- 10.4. 使用 GRUB 设置启动流程 270

11. 尾声 273 11.1. 尾声 273 vi

11.2. 统计用户数量	273
11.3. 重启系统	273
11.4. 附加资源	274
11.5. LFS 完成后的后续步骤	275

五、附录 278 A. 缩略语与术语表 279

B. 致谢	282
C. 依赖关系	285
D. 启动与系统配置脚本 version-20240825	299
D.1. /etc/rc.d/init.d/rc	299
D.2. /lib/lsb/init-functions	302
D.3. /etc/rc.d/init.d/mountvirtfs	316
D.4. /etc/rc.d/init.d/modules	318
D.5. /etc/rc.d/init.d/udev	319
D.6. /etc/rc.d/init.d/swap	321
D.7. /etc/rc.d/init.d/setclock	322
D.8. /etc/rc.d/init.d/checkfs	323
D.9. /etc/rc.d/init.d/mountfs	325
D.10. /etc/rc.d/init.d/udev_retry	326
D.11. /etc/rc.d/init.d/cleanfs	328
D.12. /etc/rc.d/init.d/console	330
D.13. /etc/rc.d/init.d/localnet	331
D.14. /etc/rc.d/init.d/sysctl	333
D.15. /etc/rc.d/init.d/sysklogd	334
D.16. /etc/rc.d/init.d/network	335
D.17. /etc/rc.d/init.d/sendsignals	336
D.18. /etc/rc.d/init.d/reboot	337
D.19. /etc/rc.d/init.d/halt	338
D.20. /etc/rc.d/init.d/template	339
D.21. /etc/sysconfig/modules 模块配置文件	340
D.22. /etc/sysconfig/createfiles 创建文件配置	340
D.23. /etc/sysconfig/udev-retry udev 重试配置	341
D.24. /sbin/ifup 网络接口启用脚本	341
D.25. /sbin/ifdown	344
D.26. /lib/services/ipv4-static	346
D.27. /lib/services/ipv4-static-route	347
E. Udev 配置规则	349
E.1. 55-lfs.rules	349
F. LFS 许可证	350
F.1. 知识共享许可协议	350

F.2. MIT 许可证	354
--------------	-----

前言

序言

我学习和深入理解 Linux 的旅程始于 1998 年。那时我刚安装了第一个 Linux 发行版，很快就被 Linux 背后的整体理念和哲学深深吸引。

完成同一项任务总有多 种方法。Linux 发行版亦是如此。多年来出现过无数发行版，有些至今犹存，有些已改头换面，还有些则永远留存在我们的记忆中。它们各具特色，以满足不同用户群体的需求。面对如此多实现相同终极目标的不同途径，我开始意识到自己不必再受限于任何单一实现方式。在接触 Linux 之前，我们只能忍受其他操作系统的种种缺陷——你别无选择。无论喜欢与否，现实就是如此。而 Linux 带来了选择的可能性。如果你对某些设计不满意，你完全有自由——甚至被鼓励——去改变它。

我尝试过多种发行版，却始终无法选定其中任何一个。这些系统本身都很出色，问题已不再关乎对错，而纯粹取决于个人偏好。面对如此丰富的选择，显然没有哪个现成系统能完全符合我的需求。于是我决定打造一个完全契合个人喜好的 Linux 系统。

为了真正让它成为我的专属系统，我决心所有组件都从源代码编译，而非使用预编译的二进制包。这个“完美”的 Linux 系统将兼具备各发行版之长而无其短。最初这个想法令人望而生畏，但我始终坚信这样的系统能够构建成功。

在解决了循环依赖和编译错误等问题后，我终于完成了这个定制版 Linux 系统。它与其他任何 Linux 系统一样功能完备、稳定可用，但这是我的独创成果。亲手组装出这样的系统让我倍感满足。唯一能超越这种成就感的，或许就是亲自编写所有软件——而这已是次优的完美方案。

当我与 Linux 社区的其他成员分享我的目标和经验时，这些理念显然引起了持续的关注。很快我们就清楚地认识到，这种定制构建的 Linux 系统不仅能满足用户的特定需求，还为程序员和系统管理员提供了提升（现有）Linux 技能的绝佳学习机会。正是基于这种广泛兴趣，“Linux From Scratch”项目应运而生。

这本《Linux From Scratch》手册是该项目的核心指南。它为你提供了设计和构建自己系统所需的背景知识和操作指导。虽然本书提供的模板能确保构建出正常运行的系统，但你完全可以自由调整操作步骤——这本身就是本项目的重要特色之一。

你始终掌握着主动权；我们只是在你启程时提供必要的帮助。

衷心希望你在构建自己的 Linux From Scratch 系统时收获快乐，并享受拥有完全属于自己系统的诸多优势。

--
杰勒德·比克曼斯
gerard@linuxfromscratch.o
rg

目标读者

您可能出于多种原因想要阅读本书。许多人常提出的一个疑问是：“既然可以直接下载安装现成的 Linux 系统，为什么还要费时费力从头开始手动构建呢？”

这个项目存在的一个重要原因，是帮助你从内部了解 Linux 系统的工作原理。构建 LFS 系统有助于展示 Linux 的运行机制，以及各个组件如何协同工作和相互依赖。这种学习体验能提供的最佳收获之一，就是能够根据自身独特需求定制 Linux 系统。

LFS 的另一个关键优势是让你完全掌控系统，而无需依赖他人实现的 Linux 发行版。使用 LFS 时，你掌握着绝对主导权。系统的每个细节都由你决定。

LFS 能让你创建极其精简的 Linux 系统。使用其他发行版时，你常常被迫安装大量既不会使用也不理解的程序，这些程序只会浪费资源。或许你会辩称，在当今天大容量硬盘和多核 CPU 的时代，资源浪费已无需考虑。但某些情况下，系统体积仍然会成为制约因素——比如可启动光盘、U 盘和嵌入式系统领域，这正是 LFS 能大显身手的地方。

定制构建 Linux 系统的另一大优势在于安全性。通过从源代码编译整个系统，您能够全面审查所有组件并应用所需的安全补丁。无需等待他人发布修复安全漏洞的二进制包——除非您亲自检查补丁并实施，否则无法确保新二进制包是否正确构建并有效解决问题。

《Linux 从零开始》的目标是构建一个完整可用的基础级系统。即便您不打算从头构建自己的 Linux 系统，本书提供的信息仍将使您获益良多。

自主构建 LFS 系统的益处不胜枚举，难以在此尽述。归根结底，教育价值是最重要的原因。随着 LFS 实践的深入，您将切身感受到信息与知识带来的强大力量。

LFS 目标架构

LFS 的主要目标架构是 AMD/Intel 的 x86（32 位）和 x86_64（64 位）处理器。另一方面，本书的指令经过适当修改后，也已知可在 Power PC 和 ARM 处理器上成功运行。要构建适用于这些替代处理器的系统，除下一页列出的要求外，主要前提条件是需要一个现有 Linux 系统作为基础——例如针对该架构的早期 LFS 安装、Ubuntu、Red Hat/Fedora、SuSE 或其他发行版。（需要注意的是，在 64 位 AMD/Intel 计算机上可以安装并使用 32 位发行版作为宿主系统。）

与 32 位系统相比，在 64 位系统上构建的性能提升微乎其微。例如，在基于 Core i7-4790 处理器（使用 4 个核心）的测试系统中构建 LFS-9.1 时，测得以下数据：

架构类型	构建耗时	构建体积
32 位	239.9 分钟	3.6GB
64 位	233.2 分钟	4.4GB

如你所见，在相同硬件条件下，64 位构建仅比 32 位构建快 3%（但体积大 22%）。若计划将 LFS 用作 LAMP 服务器或防火墙，32 位 CPU 可能已足够。但值得注意的是，当前 BLFS 中的部分软件包在编译或运行时需要超过 4GB 内存；若计划将 LFS 作为桌面系统使用，LFS 作者推荐构建 64 位系统。

LFS 默认生成的 64 位构建是“纯”64 位系统，即仅支持 64 位可执行文件。构建“多库”系统需要将许多应用程序分别编译两次——一次针对 32 位系统，一次针对 64 位系统。由于这会干扰提供基础 Linux 系统最小化教学的核心目标，LFS 并未直接支持该功能。部分 LFS/BLFS 编辑者维护了 LFS 的多库分支版本，可通过 <https://www.linuxfromscratch.org/~thomas/multilib/index.html> 访问。但这属于进阶主题。

前提条件

构建 LFS 系统并非易事。它要求使用者具备一定的 Unix 系统管理基础知识，以便解决问题并正确执行列出的命令。至少需要掌握以下基本技能：能够使用命令行（shell）复制或移动文件及目录、查看目录和文件内容、切换当前工作目录。同时还应了解如何安装和使用 Linux 软件。

由于 LFS 手册默认读者已掌握这些基础技能，因此各类 LFS 支持论坛通常不会在这些方面提供过多帮助。您会发现涉及这些基础知识的提问很可能得不到回复（或仅会收到建议查阅 LFS 必读预备资料的指引）。

在构建 LFS 系统之前，我们强烈建议您阅读以下文章：

- 软件构建指南 <https://tldp.org/HOWTO/Software-Building-HOWTO.html>

这是一份在 Linux 系统下构建和安装“通用”Unix 软件包的完整指南。尽管编写时间较早，但仍对软件构建与安装的基本技术提供了很好的总结。

- 源代码安装新手指南 <https://moi.vonos.net/linux/beginners-installing-from-source/>

本指南很好地总结了从源代码构建软件所需的基本技能与技术。

LFS 与标准

LFS 的结构尽可能遵循 Linux 标准。主要标准包括：

- POSIX.1-2008 标准
- 文件系统层次结构标准(FHS) 3.0 版
- Linux 标准基础 (LSB) 5.0 版 (2015 年)

LSB 包含四个独立规范：核心 (Core)、桌面 (Desktop)、语言 (Languages) 和成像 (Imaging)。其中核心与桌面规范的部分内容与特定架构相关。另有两个试验性规范：Gtk3 和图形 (Graphics)。LFS 力求符合前文讨论的 IA32 (32 位 x86) 或 AMD64 (x86_64) 架构的 LSB 规范要求。

注意

许多人对这些规范要求持有异议。LSB 的主要目的是确保专有软件能在合规系统上安装运行。由于 LFS 采用源码构建方式，用户可完全自主决定需要安装的软件包，您可以选择不安装 LSB 指定的某些软件包。

虽然可以从零开始构建一个能通过 LSB 认证测试的完整系统，但这需要许多超出 LFS 书籍范围的额外软件包。其中部分额外软件包的安装指南可在 BLFS 中找到。

满足 LSB 要求所需的 LFS 提供软件包

LSB 核心组件：

Bash、Bc、Binutils、Coreutils、Diffutils、File、Findutils、Gawk、GCC、Gettext、Glibc、Grep、Gzip、M4、Man-DB、Procps、Psmisc、Sed、Shadow、

LSB 桌面环境: 无
 LSB 支持语言: Perl
 LSB 图像处理: 无
 LSB Gt3 及图形组件 (试用阶段): 无

满足 LSB 规范所需的 BLFS 提供软件包

LSB 核心组件:	定时任务工具 At (含 Batch 功能)、BLFS 版 Bash 启动文件、归档工具 Cpio、行编辑器 Ed、定时任务工具 Fcrontab、LSB 专用工具集、Netscape 便携运行时 NSPR、网络安全服务
LSB 桌面环境:	Alsa、ATK、Cairo、桌面文件工具、Freetype、字体配置、Gdk-pixbuf、Glib2、GLU、图标命名工具、Libjpeg-turbo、Libxml2、Mesa、Pango、Xdg-utils、Xorg
LSB 语言支持: Libxml2、Libxslt	
LSB 图像处理: CUPS、Cups-filters、Ghostscript、SANE	
LSB Gt3 与 LSB 图形组件 (试用版):	GTK+3

满足 LSB 规范要求但 LFS 或 BLFS 未提供或选择性提供的组件

LSB 核心组件:	install_initd、libcrypt.so.1 (可通过 LFS Libcrypt 软件包的可选指令提供)、libncurses.so.5 (可通过 LFS Ncurses 软件包的可选指令提供) libncursesw.so.5 (但 libncursesw.so.6 由 LFS Ncurses 软件包提供)
LSB 桌面环境:	libgdk-x11-2.0.so (但 libgdk-3.so 由 BLFS GTK+3 软件包提供), libgtk-x11-2.0.so (但 libgtk-3.so 和 libgtk-4.so 由 BLFS GTK+3 及 GTK-4 软件包提供), libpng12.so (但 libpng16.so 由 BLFS Libpng 软件包提供), libQt*.so.4 (但 libQt6*.so.6 由 BLFS Qt6 软件包提供), libtiff.so.4 (但 libtiff.so.6 由 BLFS Libtiff 软件包提供)
LSB 语言支持:	/usr/bin/python (LSB 要求 Python2, 但 LFS 和 BLFS 仅提供 Python3)
LSB 图像处理: 无 LSB Gt3 与 LSB 图形组件 (试用版):	libpng15.so (但 BLFS 的 Libpng 软件包提供的是 libpng16.so)

本书选用软件包的理由说明

LFS 的目标是构建一个完整可用的基础级系统——包含所有能自我复制的必要软件包——并提供一个相对精简的基础平台, 让用户能基于此根据个人需求定制更完整的系统。这并不意味着 LFS 是可能的最小系统。其中包含了一些严格来说非必需但非常重要的软件包。下方列表详细说明了书中每个软件包的入选理由。

该软件包包含用于管理访问控制列表(ACL)的工具，这些列表用于为文件和目录定义细粒度的自主访问权限。

- 属性管理工具

该软件包包含用于管理文件系统对象扩展属性的程序。

- 自动配置工具

该软件包提供用于生成能自动配置开发者模板源代码的 shell 脚本程序。当构建流程更新后，通常需要用它来重新构建软件包。

- Automake

该软件包包含从模板生成 Makefile 的程序。当构建流程更新后，通常需要用它来重新构建软件包。

- Bash

该软件包满足 LSB 核心要求，为系统提供 Bourne Shell 接口。因其广泛使用和强大功能，它被选为替代其他 Shell 软件包的方案。

- Bc

该软件包提供任意精度数值处理语言，是构建 Linux 内核的必要组件。

- Binutils

该软件包提供了链接器、汇编器以及其他处理目标文件的工具。构建 LFS 系统中大多数软件包时都需要该软件包内的程序。

- Bison

此软件包包含构建多个 LFS 程序所需的 yacc (Yet Another Compiler Compiler) 的 GNU 版本。

- Bzip2

该软件包包含用于压缩和解压缩文件的程序，解压许多 LFS 软件包时需要用到它。

- 检查

该软件包为其他程序提供测试框架。

- 核心工具集

该软件包包含一系列查看和操作文件及目录的基础程序。这些程序是命令行文件管理所必需的，也是 LFS 中每个软件包安装过程中不可或缺的工具。

- DejaGNU

该软件包提供了测试其他程序的框架工具。

- Diffutils

该软件包包含用于显示文件或目录差异的程序。这些程序可用于创建补丁，也被许多软件包的构建过程所使用。

- E2fsprogs

该软件包提供了处理 ext2、ext3 和 ext4 文件系统的实用工具。这些是 Linux 支持的最常见且经过充分测试的文件系统。

- Expat

该软件包提供了一个相对较小的 XML 解析库。它是 Perl 模块 XML::Parser 所需的组件。

- Expect

该软件包包含一个用于与其他交互式程序进行脚本化对话的程序，常用于测试其他软件包。

- 文件

该软件包包含一个用于判断给定文件类型的实用工具，部分软件包在构建脚本中需要用到它。

- 查找工具

该软件包提供在文件系统中查找文件的程序，被众多软件包的构建脚本所使用。

- Flex

该软件包包含用于生成文本模式识别程序的工具，是 lex（词法分析器）程序的 GNU 版本。构建多个 LFS 软件包时需要此工具。

- Gawk

该软件包提供用于操作文本文件的程序。它是 GNU 版本的 awk (Aho-Weinberg-Kernighan)。许多其他软件包的构建脚本都会使用它。

- GCC

这是 GNU 编译器集合。它包含 C 和 C++ 编译器，以及 LFS 未构建的其他几种编译器。

- GDBM

该软件包包含 GNU 数据库管理器库，被另一个 LFS 软件包 Man-DB 所使用。

- Gettext

该软件包为众多软件包提供国际化和本地化的工具与库支持。

- Glibc

该软件包包含主要的 C 语言库。没有它，Linux 程序将无法运行。

- GMP

该软件包提供数学函数库，支持任意精度算术运算。构建 GCC 时需要此组件。

- Gperf

该软件包生成的程序能够根据一组键值生成完美哈希函数，是 Udev 所需的组件。

- Grep

该软件包包含用于文件搜索的程序，多数软件包的构建脚本都会使用这些程序。

- Groff

该软件包提供用于处理和格式化文本的程序，其重要功能之一是格式化手册页。

- GRUB

这是 GRand Unified Bootloader（大一统引导加载程序），它是现有多种引导加载程序中灵活性最高的方案。

- Gzip

该软件包包含用于压缩和解压缩文件的程序。在 LFS 中解压许多软件包时需要用到它。

- Iana-etc

该软件包提供网络服务和协议数据。启用完整网络功能时需要此组件。

- Inetutils

该软件包提供基础网络管理程序。

- Intltool

该软件包提供从源文件中提取可翻译字符串的工具。

- IProute2

该软件包包含基础及高级 IPv4 和 IPv6 网络工具程序。因其支持 IPv6 的特性，我们选择它而非其他常见网络工具包（如 net-tools）。

- Kbd

该软件包生成键位映射表文件、为非美式键盘提供键盘实用工具，以及多种控制台字体。

- Kmod

该软件包提供管理 Linux 内核模块所需的程序。

- Less

该软件包包含一个出色的文本文件查看器，可在浏览文件时上下滚动。许多软件包使用它来分页输出内容。

- Libcap

该软件包实现了 Linux 内核中 POSIX 1003.1e 能力特性的用户空间接口。

- Libelf

elfutils 项目提供了处理 ELF 文件和 DWARF 数据的库与工具。虽然该软件包中的多数工具在其他软件包中也可获取，但构建 Linux 内核时若采用默认（且最高效）配置则必须使用此库。

- Libffi

该软件包实现了可移植的高级编程接口，支持多种调用约定。某些程序在编译时可能无法确定需要传递给函数的参数。例如，解释器可能在运行时才获知调用特定函数所需的参数数量和类型。此类程序可利用 Libffi 在解释程序与编译代码之间建立桥梁。

- Libpipeline

Libpipeline 软件包提供了一个库，用于以灵活便捷的方式操作子进程管道。该库是 Man-DB 软件包的必备依赖项。

- Libtool

该软件包包含 GNU 通用库支持脚本，它将使用共享库的复杂性封装成统一且可移植的接口。该工具是其他 LFS 软件包中测试套件的必要组件。

- Libxcrypt

该软件包提供了多种程序（尤其是 Shadow）所需的 libcrypt 库，用于密码哈希处理。它取代了 Glibc 中过时的 libcrypt 实现。

- Linux 内核

该软件包是操作系统本身，即 GNU/Linux 环境中的 Linux 核心组件。

- M4

该软件包提供了一个通用的文本宏处理器，可作为其他程序的构建工具使用。

- Make

该软件包包含一个用于指导软件包构建的程序。LFS 中几乎每个软件包都需要它。

- Man-DB

该软件包包含用于查找和查看手册页的程序。选择它而非 man 软件包的原因是它具有更出色的国际化支持能力。它提供了 man 程序。

- Man-pages

该软件包提供了基础 Linux 手册页的实际内容。

- Meson

该软件包提供了一个用于自动化构建软件的工具。Meson 的主要目标是尽量减少开发人员在配置构建系统上花费的时间。构建 Systemd 以及许多 BLFS 软件包时都需要它。

- MPC

该软件包提供复数运算功能。GCC 编译器需要依赖此软件包。

- MPFR

该软件包包含用于多精度算术运算的函数，是 GCC 的必需组件。

- Ninja

该软件包提供了一套专注于速度的小型构建系统，其设计目标是通过高级构建系统生成输入文件，并以最快速度执行构建任务。该软件包是 Meson 的必需组件。

- Ncurses

该软件包包含用于独立于终端的字符屏幕处理的库。它常用于为菜单系统提供光标控制功能，是 LFS 中多个软件包所需的依赖项。

- Openssl

该软件包提供与密码学相关的管理工具和库，为包括 Linux 内核在内的其他软件包提供加密功能支持。

- 补丁

该软件包包含一个通过应用通常由 diff 程序创建的补丁文件来修改或创建文件的程序。多个 LFS 软件包的构建过程需要它。

- Perl

该软件包是运行时语言 PERL 的解释器。多个 LFS 软件包的安装和测试套件需要它。

- Pkgconf

该软件包包含一个帮助配置开发库编译器与链接器标志的程序。该程序可作为 pkg-config 的替代品使用，许多软件包的构建系统都需要此功能。相比原版 Pkg-config 软件包，它的维护更活跃且运行速度略快。

- Procs-NG

此软件包包含进程监控程序。这些程序对系统管理非常有用，同时也被 LFS 启动脚本所使用。

- Psmisc

该软件包提供用于显示运行进程信息的程序，这些程序对系统管理非常有用。

- Python 3

该软件包提供一种解释型编程语言，其设计理念强调代码可读性。

- Readline

该软件包提供了一套实现命令行编辑和历史功能的库，被 Bash 所使用。

- Sed

该软件包支持无需打开文本编辑器即可进行文本编辑，同时也是许多 LFS 软件包配置脚本的必需组件。

- 影子密码

该软件包包含用于安全处理密码的程序。

- 系统日志守护进程

该软件包提供记录系统消息的程序，例如当异常事件发生时由内核或守护进程发出的消息。

- SysVinit

该软件包提供 init 程序，这是运行中的 Linux 系统上所有其他进程的父进程。

- Udev

该软件包是设备管理器。当设备被添加至系统或从系统中移除时，它能动态控制/dev 目录下设备节点的所有权、权限、名称及符号链接。

- Tar

该软件包提供几乎所有 LFS 所用软件包的归档与解压功能。

- Tcl

该软件包包含用于多个测试套件的工具命令语言(Tcl)。

- Texinfo

该软件包提供用于读取、写入和转换 info 页面的程序，被众多 LFS 软件包的安装过程所使用。

- Util-linux

该软件包包含各类实用工具程序，其中包括用于处理文件系统、控制台、分区和消息的实用工具。

- Vim

该软件包提供了一个编辑器。选择它的原因在于其与经典 vi 编辑器的兼容性以及强大的功能集。对许多用户而言，编辑器是非常个人化的选择。如有需要，您可以选择替换为其他任意编辑器。

- Wheel

该软件包提供了一个 Python 模块，是 Python wheel 打包标准的参考实现。

- XML::解析器

该软件包是一个与 Expat 交互的 Perl 模块。

- XZ 工具集

该软件包包含用于压缩和解压缩文件的程序。它提供了当前普遍可用的最高压缩率，对于解压 XZ 或 LZMA 格式的软件包非常有用。

- Zlib 压缩库

该软件包包含某些程序使用的压缩和解压缩例程。

- Zstd 压缩工具

该软件包提供了一些程序使用的压缩与解压例程，能够实现高压缩比，并提供广泛的压缩速度与压缩率权衡方案。

排版规范

为便于理解，本书通篇采用若干排版约定。本节列举了《Linux From Scratch》中常见的排版格式示例。

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

此类文本格式要求严格按原样输入，除非上下文另有说明。在解释性章节中，这种格式也用于标识所引用的具体命令。

某些情况下，逻辑行会通过行尾的反斜杠延伸至两个或多个物理行。

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

请注意反斜杠后必须直接换行。若出现空格或制表符等其他空白字符，将导致错误结果。

```
install-info: 未知选项 '--dir-file=/mnt/lfs/usr/info/dir'
```

这种固定宽度的文本格式用于显示屏幕输出，通常是执行命令后的结果。该格式也用于显示文件名，例如/etc/ld.so.conf。

注意

请将浏览器配置为使用良好的等宽字体（字号 9pt）显示固定宽度文本，以便清晰区分 Il1 或 O0 等字符的形态差异。

强调

本书中此类文本形式用于多种用途，其主要目的是强调重点内容或条目。

<https://www.linuxfromscratch.org/>

该格式用于 LFS 社区内部及外部页面的超链接，包含操作指南、下载地址及网站链接。

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

EOF

该格式用于创建配置文件时。第一条命令指示系统创建文件\$LFS/etc/group，内容将来自随后输入的行，直到遇到文件结束符(EOF)为止。因此，通常直接按原文输入整个部分即可。

<替换文本>

此格式用于封装无需按原样输入或进行复制粘贴操作的文本。

[可选文本]

此格式用于封装可选文本。

passwd(5)

这种格式用于引用特定的手册页（man page）。括号内的数字表示手册中的特定章节。例如，passwd 有两个手册页。根据 LFS 安装说明，这两个手册页

将分别位于/usr/share/man/man1/passwd.1 和/usr/share/man/man5/passwd.5。当书中使用 passwd(5)时

特指/usr/share/man/man5/passwd.5。直接运行 man passwd 会显示第一个匹配"passwd"的手册页，即/usr/share/man/man1/passwd.1。因此要查看指定章节，需要运行 man 5 passwd。注意大多数手册页在不同章节中不会有重复名称，通常直接 man <程序名>即可。在 LFS 手册中，这些手册页引用也是超链接，点击可直接打开 Arch Linux 手册页渲染的 HTML 版本。

结构

本书分为以下几个部分。

第一部分 - 简介

第一部分阐述了进行 LFS 安装时需要注意的几个重要事项。本部分还提供了关于本书的元信息。

第二部分 - 构建前的准备工作

第二部分描述如何为构建过程做准备——包括创建分区、下载软件包以及编译临时工具。

第三部分 - 构建 LFS 交叉工具链与临时工具

第三部分提供构建最终 LFS 系统所需工具的具体指导。

第四部分 - 构建 LFS 系统

第四部分将引导读者逐步构建 LFS 系统——逐个编译安装所有软件包、设置启动脚本以及安装内核。最终构建完成的 Linux 系统将成为基础平台，可根据需要在此基础上扩展安装其他软件。本书末尾附有便捷的参考列表，详细记录了所有已安装的程序、库文件及重要文件。

第五部分 - 附录

第五部分提供与本书相关的参考信息，包括术语缩写表、致谢声明、软件包依赖关系、LFS 启动脚本清单、本书发行许可协议，以及完整的软件包/程序/库文件/脚本索引。

勘误与安全通告

用于构建 LFS 系统的软件会持续更新和优化。在 LFS 手册发布后，可能会出现安全警告和错误修复。在开始构建前，请访问 <https://www.linuxfromscratch.org/lfs/errata/12.3/> 以检查本版 LFS 中的软件包版本或操作说明是否需要修改——无论是修复安全漏洞还是修正其他错误。在构建 LFS 系统时，您应当记录所有显示的变更，并将其应用到手册的相关章节中。

此外，Linux From Scratch 编辑团队会维护手册发布后发现的安全漏洞列表。在开始构建前，请访问 <https://www.linuxfromscratch.org/lfs/advisories/> 查阅该列表。构建 LFS 系统时，您应当根据通告建议对手册相关章节进行相应修改。如果您计划将 LFS 系统用作实际桌面或服务器系统，即使系统已完全构建完成，也应持续关注安全通告并修复所有安全漏洞。

第一部分 简介

第 1 章 简介

1.1 如何构建 LFS 系统

LFS 系统将通过已安装的 Linux 发行版（如 Debian、OpenMandriva、Fedora 或 openSUSE）进行构建。这个现有的 Linux 系统（称为宿主系统）将作为起点，提供必要的程序（包括编译器、链接器和 shell）来构建新系统。在发行版安装过程中选择“开发”选项以包含这些工具。

注意

安装 Linux 发行版有多种方式，但默认设置通常不适合构建 LFS 系统。关于如何设置商业发行版的建议，请参阅：<https://www.linuxfromscratch.org/hints/downloads/files/partitioning-for-lfs.txt>

除了在您的机器上安装一个独立的发行版之外，您也可以考虑使用商业发行版的 LiveCD。

本书第 2 章将指导您如何创建一个新的 Linux 原生分区和文件系统，用于编译和安装新的 LFS 系统。第 3 章详细列出了构建 LFS 系统所需的软件包和补丁，并说明如何将它们存储在新文件系统中。第 4 章讨论了如何配置合适的工作环境。请仔细阅读第 4 章，因为它阐述了在开始第 5 章及后续章节之前您需要了解的若干重要事项。

第 5 章将介绍如何使用交叉编译技术安装初始工具链（包括 binutils、gcc 和 glibc），从而将新工具与宿主系统隔离开来。

第 6 章将展示如何利用刚刚构建的交叉工具链来编译基础工具程序。

第七章将进入“chroot”环境，我们会使用新构建的工具来创建 LFS 系统所需的所有剩余工具。

这种将新系统与宿主发行版隔离的做法看似有些过度。关于为何要这样做的完整技术说明，请参阅工具链技术注释部分。

第八章将构建完整的 LFS 系统。chroot 环境提供的另一个优势是，在构建 LFS 的同时，您仍可以继续使用宿主系统。在等待软件包编译完成时，您可以照常使用计算机。

为了完成安装，第九章将进行基本系统配置，第十章将创建内核和引导加载程序。第十一章包含如何继续深入 LFS 体验的信息。完成本章所述步骤后，计算机就可以启动进入全新的 LFS 系统了。

简而言之，这就是整个构建流程。后续章节将详细阐述每个步骤。那些现在看似复杂的环节都会得到清晰解释，当您开启 LFS 之旅时，一切都会变得井然有序。

1.2. 新版更新内容

以下是自 LFS 上一版本发布以来更新的软件包列表。

升级至：

- Bash-5.2.37
- Bc-7.0.3
- Binutils-2.44
- Coreutils-9.6
- Diffutils-3.11
- E2fsprogs-1.47.2
- Expat-2.6.4
- File-5.46
- Flit-core-3.11.0
- Gawk-5.3.1
- Gettext-0.24
- Glibc-2.41
- Iana-Etc-20250123
- Inetutils-2.6
- IPRoute2-6.13.0
- Jinja2-3.1.5
- Kbd-2.7.1
- Kmod-34
- Less-668
- Libcap-2.73
- Elfutils-0.192 中的 Libelf 库
- Libffi-3.4.7
- Libpipeline-1.5.8
- Libtool-2.5.4
- Libxcrypt-4.4.38
- Linux-6.13.4
- Man-DB-2.13.0
- Man-pages-6.12
- MarkupSafe-3.0.2
- Meson-1.7.0
- OpenSSL-3.4.1
- Perl-5.40.1
- Procps-ng-4.0.5
- Python-3.13.2
- Setuptools-75.8.1
- Shadow-4.17.3
- Sysklogd-2.7.0

- Systemd-257.3
- SysVinit-3.14
- Tcl-8.6.16
- Texinfo-7.2
- 时区数据-2025a
- Systemd-257.3 中的 Udev
- 工具集-linux-2.40.4
- Vim-9.1.1166
- Wheel-0.45.1
- Xz-5.6.4
- Zstd-1.5.7

新增:

.

已移除:

.

1.3. 更新日志 本文档为 2025 年 3 月 5 日发布的《Linux 从零开始》第 12.3 版。若您获取的版本已超过六个月，可能有更新的版本可供使用。请通过 <https://www.linuxfromscratch.org/mirrors.html> 访问镜像站点查询最新版本。

linuxfromscratch.org/mirrors.html 访问镜像站点查询最新版本。

以下列出的是自上一版本发布以来的变更内容。

更新日志条目:

- 2025-03-05
 - [bdubbs] - 发布 LFS-12.3 版本。
- 2025-03-02
 - [bdubbs] - 更新至 vim-9.1.1166 (安全更新)。修复问题 #5666。
- 2025-02-27
 - [bdubbs] - 更新至 zstd-1.5.7。修复问题 #5652。
 - [bdubbs] - 更新至 systemd-257.3。修复问题 #5612。
 - [bdubbs] - 更新至 shadow-4.17.3。修复问题 #5660。
 - [bdubbs] - 更新至 setuptools-75.8.1。修复问题 #5662。
 - [bdubbs] - 更新至 linux-6.13.4。修复问题 #5647。
 - [bdubbs] - 更新至 kmod-34。修复问题 #5657。
 - [bdubbs] - 更新至 inetutils-2.6。修复问题 #5656。
 - [bdubbs] - 更新至 gettext-0.24。修复问题 #5661。
 - [bdubbs] - 更新至 flit_core-3.11.0。修复问题 #5654。

- 2025-02-24
 - [bdubbs] - 更新至 man-pages-6.12。修复问题 #5658。
- 2025-02-19
 - [xry111] - 更新至 vim-9.1.1122 (安全更新)。处理问题 #4500。
 - [xry111] - 更新至 man-pages-6.11。修复问题 #5646。
- 2025-02-13
 - [bdubbs] - 更新至 vim-9.1.1106。处理问题 #4500。
 - [bdubbs] - 更新至 diffutils-3.11。修复问题 #5639。
 - [bdubbs] - 更新至 libffi-3.4.7。修复问题 #5642。
 - [bdubbs] - 更新至 linux-6.13.2。修复问题 #5643。
 - [bdubbs] - 更新至 Python3-3.13.2。修复问题 #5640。
 - [bdubbs] - 更新至 sysvinit-3.14。修复问题 #5641。
- 2025-02-02
 - [bdubbs] - 更新至 vim-9.1.1071。处理问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20250123。处理问题 #5006。
 - [bdubbs] - 更新至 binutils-2.44.0。修复问题 #5634。
 - [bdubbs] - 更新至 coreutils-9.6。修复问题 #5628。
 - [bdubbs] - 更新至 e2fsprogs-1.47.2。修复问题 #5637。
 - [bdubbs] - 更新至 glibc-2.41。修复问题 #5638。
 - [bdubbs] - 更新至 iproute2-6.13.0。修复问题 #5631。
 - [bdubbs] - 更新至 libxcrypt-4.4.38。修复问题 #5626。
 - [bdubbs] - 更新至 linux-6.13.1。修复问题 #5629。
 - [bdubbs] - 更新至 man-pages-6.10。修复问题 #5632。
 - [bdubbs] - 更新至 meson-1.7.0。修复问题 #5636。
 - [bdubbs] - 更新至 perl-5.40.1。修复问题 #5630。
 - [bdubbs] - 更新至 tcl8.6.16。修复问题 #5635。
 - [bdubbs] - 更新至 tzdata2025a。修复问题 #5627。
 - [bdubbs] - 更新至 xz-5.6.4。修复问题 #5633。
- 2025-01-15
 - [bdubbs] - 更新至 vim-9.1.1016。修复问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20250108。修复问题 #5006。
 - [bdubbs] - 更新至 util-linux-2.40.4。修复问题 #5624。
 - [bdubbs] - 更新至 sysvinit-3.13。修复问题 #5621。
 - [bdubbs] - 更新至 sysklogd-2.7.0。修复问题 #5623。
 - [bdubbs] - 更新至 shadow-4.17.2。修复问题 #5625。
 - [bdubbs] - 更新至 setuptools-75.8.0。修复问题 #5622。

- [bdubbs] - 更新至 linux-6.12.9。修复问题 #5620。
- [bdubbs] - 更新至 gettext-0.23.1。修复问题 #5619。
- 2025-01-01
 - [renodr] - 更新至 libxcrypt-4.4.37。修复问题 #5618。
 - [bdubbs] - 更新至 iana-etc-20241220。处理问题 #5006。
 - [bdubbs] - 更新至 texinfo-7.2。修复问题 #5616。
 - [bdubbs] - 更新至 sysvinit-3.12。修复问题 #5615。
 - [bdubbs] - 更新至 shadow-4.17.1。修复问题 #5617。
 - [bdubbs] - 更新至 procps-ng-4.0.5。修复问题 #5611。
 - [bdubbs] - 更新至 meson-1.6.1。修复问题 #5610。
 - [bdubbs] - 更新至 linux-6.12.7。修复问题 #5613。
 - [bdubbs] - 更新至 kbd-2.7.1。修复问题 #5608。
 - [bdubbs] - 更新至 jinja2-3.1.5 (安全更新)。修复问题 #5614。
- 2024-12-15
 - [bdubbs] - 更新至 vim-9.1.0927。修复问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20241206。修复问题 #5006。
 - [bdubbs] - 更新至 systemd-257。修复问题 #5559。
 - [bdubbs] - 更新至 Python-3.13.1 (安全更新)。修复问题 #5605。
 - [bdubbs] - 更新至 libcap-2.73。修复问题 #5604。
 - [bdubbs] - 更新至 linux-6.12.5。修复问题 #5607。
 - [bdubbs] - 更新至 kbd-2.7。修复问题 #5608。
 - [bdubbs] - 更新至 gettext-0.23。修复问题 #5603。
- 2024-12-01
 - [bdubbs] - 更新至 iana-etc-20241122。修复问题 #5006。
 - [bdubbs] - 更新至 file-5.46。修复问题 #5601。
 - [bdubbs] - 更新至 iproute2-6.12.0。修复问题 #5597。
 - [bdubbs] - 更新至 libtool-2.5.4。修复问题 #5598。
 - [bdubbs] - 更新至 linux-6.12.1。修复问题 #5586。
 - [bdubbs] - 更新至 setuptools-75.6.0 (Python 模块)。修复问题 #5599。
 - [bdubbs] - 更新至 wheel-0.45.1 (Python 模块)。修复问题 #5600。
- 2024-11-15
 - [bdubbs] - 更新至 vim-9.1.0866。处理问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20241024。解决 #5006 问题。
 - [bdubbs] - 更新至 wheel-0.45.0 (Python 模块)。修复 #5593 问题。
 - [bdubbs] - 更新至 setuptools-75.5.0 (Python 模块)。修复 #5595 问题。
 - [bdubbs] - 更新至 linux-6.11.8。修复 #5582 问题。

- [bdubbs] - 更新至 libcap-2.72。修复问题 #5594。
- 2024-11-08
 - [bdubbs] - 添加 binutils-2.43.1-upstream_fix-1.patch 补丁。修复问题 #5591。
 - [bdubbs] - 更新至 flit_core-3.10.1。修复问题 #5589。
 - [bdubbs] - 更新至 expat-2.6.4。修复问题 #5590。
- 2024-10-25
 - [bdubbs] - 更新至 linux-6.11.6。修复 #5588。
 - [bdubbs] - 更新至 libcap-2.71。修复 #5584。
 - [bdubbs] - 更新至 setuptools-75.3.0。修复 #5585。
 - [bdubbs] - 更新至 flit_core-3.10.0。修复 #5587。
- 2024-10-25
 - [bdubbs] - 更新至 iana-etc-20241015。处理问题#5006。
 - [bdubbs] - 更新至 vim-9.1.0813。处理问题#4500。
 - [bdubbs] - 更新至 xz-5.6.3。修复问题#5572。
 - [bdubbs] - 更新至 sysvinit-3.11。修复问题#5581。
 - [bdubbs] - 更新至 setuptools-75.2.0。修复问题 #5577。
 - [bdubbs] - 更新至 Python3-3.13.0。修复问题 #5575。
 - [bdubbs] - 更新至 openssl-3.4.0。修复问题 #5582。
 - [bdubbs] - 更新至 meson-1.6.0。修复问题 #5580。
 - [bdubbs] - 更新至 markupsafe-3.0.2。修复问题 #5576。
 - [bdubbs] - 更新至 linux-6.11.5。修复问题 #5574。
 - [bdubbs] - 更新至 less-668。修复问题 #5578。
 - [bdubbs] - 更新至 elfutils-0.192。修复问题 #5579。
- 2024-10-03
 - [bdubbs] - 回退至 tcl8.6.15 版本。
- 2024-10-01
 - [bdubbs] - 更新至 Python3-3.12.7 版本。修复问题 #5571。
 - [bdubbs] - 更新至 tcl9.0.0 版本。修复问题 #5570。
 - [bdubbs] - 更新至 linux-6.11.1 版本。修复问题 #5556。
 - [bdubbs] - 更新至 libtool-2.5.3。修复问题 #5569。
 - [bdubbs] - 更新至 iproute2-6.11.0。修复问题 #5561。
 - [bdubbs] - 更新至 bash-5.2.37。修复问题 #5567。
 - [bdubbs] - 更新至 bc-7.0.3。修复问题 #5568。
- 2024-09-20
 - [bdubbs] - 更新至 vim-9.1.0738 版本。解决 #4500 问题。
 - [bdubbs] - 更新至 texinfo-7.1.1 版本。修复 #5558 问题。

- [bdubbs] - 更新至 tcl8.6.15。修复问题 #5562。
- [bdubbs] - 更新至 sysklogd-2.6.2。修复问题 #5557。
- [bdubbs] - 更新至 setuptools-75.1.0 版本。修复问题#5560。
- [bdubbs] - 更新至 meson-1.5.2 版本。修复问题#5566。
- [bdubbs] - 更新至 iana-etc-20240912。修复问题#5006。
- [bdubbs] - 更新至 gawk-5.3.1。修复问题#5564。
- [bdubbs] - 更新至 bc-7.0.2。修复问题#5563。

· 2024-09-07

- [bdubbs] - 更新至 tzdata-2024b。修复问题#5554。
- [bdubbs] - 更新至 systemd-256.5。修复问题 #5551。
- [bdubbs] - 更新至 setuptools-74.1.2。修复问题 #5546。
- [bdubbs] - 更新至 python3-3.12.6。修复问题 #5555。
- [bdubbs] - 更新至 openssl-3.3.2。修复问题 #5552。
- [bdubbs] - 更新至 man-db-2.13.0。修复问题 #5550。
- [bdubbs] - 更新至 linux-6.10.8。修复问题 #5545。
- [bdubbs] - 更新至 libpipeline-1.5.8。修复问题 #5548。
- [bdubbs] - 更新至 expat-2.6.3。修复问题 #5553。
- [bdubbs] - 更新至 bc-7.0.1 版本。修复问题 #5547。

· 2024-09-01

- [bdubbs] - LFS-12.2 版本已发布。

1.4. 资源

1.4.1. 常见问题解答 在构建 LFS 系统过程中，若遇到任何错误、疑问或认为书中存在排版错误，请首先查阅位于 <https://www.linuxfromscratch.org/FAQ.html> 的常见问题解答（FAQ）列表。

org/faq/。

1.4.2. 邮件列表 [linuxfromscratch.org](https://www.linuxfromscratch.org/) 服务器托管了多个用于 LFS 项目开发的邮件列表。这些列表包括主要的开发和支持列表等。如果您在 FAQ 页面找不到问题的答案，下一步可以访问 <https://www.linuxfromscratch.org/search.html> 搜索邮件列表。

有关不同列表的信息、订阅方式、存档位置及其他详情，请访问 <https://www.linuxfromscratch.org/mail.html>。

1.4.3. IRC 聊天 LFS 社区的若干成员通过互联网中继聊天(IRC)提供帮助。在使用此支持渠道前，请确认您的问题未在 LFS 常见问题解答或邮件列表存档中得到解答。您可以在 [irc.libera.chat](irc://irc.libera.chat) 找到 IRC 网络，支持频道名为#lfs-support。

1.4.4. 镜像站点 LFS 项目在全球设有多个镜像站点，以便更便捷地访问网站和下载所需软件包。请访问 LFS 官网 <https://www.linuxfromscratch.org/mirrors.html> 获取最新镜像列表。

1.4.5. 联系方式 请将所有问题与建议发送至 LFS 邮件列表（参见上文）。

1.5. 帮助说明

若您在按照 LFS 指南构建某个软件包时遇到问题，我们强烈建议您先通过 1.4 节“资源”中列出的 LFS 支持渠道进行讨论，而非直接向上游支持渠道提交问题。这种做法通常效率极低，因为上游维护者往往不熟悉 LFS 的构建流程。即便您确实遇到了上游问题，LFS 社区仍能帮助提取上游维护者所需的关键信息，并提交规范的问题报告。

若您必须直接通过上游支持渠道提问，至少应当注意：许多上游项目将支持渠道与缺陷追踪系统分开管理。在这些项目中，将提问作为“缺陷”报告提交会被视为无效，并可能干扰上游开发者的工作。

若在使用本书过程中遇到问题或疑问，请先查阅常见问题解答页面：

<https://www.linuxfromscratch.org/faq/#generalfaq>。该页面通常已涵盖多数常见问题。若您的问题未获解答，请尝试定位问题根源。以下提示文档可提供故障排查指导：

<https://www.linuxfromscratch.org/hints/downloads/files/errors.txt>。

如果在 FAQ 中找不到您遇到的问题，请访问 <https://www.linuxfromscratch.org/search> 搜索邮件列表。

html.

我们还有一个优秀的 LFS 社区，愿意通过邮件列表和 IRC 提供帮助（参见本书 1.4 节“资源”部分）。然而，我们每天都会收到多个支持请求，其中许多问题本可以通过查阅 FAQ 或先搜索邮件列表轻松解决。因此，为了让我们能提供最佳协助，您应该先自行进行一些研究。这样我们才能专注于处理更特殊的 support 需求。如果您的搜索未能找到解决方案，请在求助时包含以下所有相关信息。

1.5.1. 需要说明的事项 除了简要描述遇到的问题外，任何求助请求都应包含以下关键

事物

- 所使用的书籍版本（此处为 12.3）
- 用于构建 LFS 的主机发行版及其版本
- 主机系统需求脚本的输出内容
- 遇到问题的软件包或相关章节
- 确切的错误信息，或对问题的清晰描述

- 记录你是否完全按照本书操作

注意

偏离本书指导并不意味着我们不会提供帮助。毕竟，LFS 的核心在于个人定制。事先说明对既定流程的任何改动，将有助于我们评估和确定问题的潜在原因。

1.5.2. 配置脚本问题 若运行 `configure` 脚本时出现错误，请检查 `config.log` 文件。该文件可能包含配置过程中未显示在屏幕上的错误信息。如需寻求帮助，请附上相关行内容。

1.5.3. 编译问题 屏幕输出和各种文件内容对于诊断编译问题都很有价值。

配置脚本和 `make` 运行的屏幕输出可能有所帮助。不必包含全部输出，但需包含所有相关信息。以下是 `make` 屏幕输出中应包含的信息类型示例。

```
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 signature.o variable.o
vpath.o default.o remote-stub.o version.o getopt.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-recurse] Error 1 make[1]: Leaving
directory `/lfs/tmp/make-3.79.1' make: *** [all-recurse-am] Error 2
```

这种情况下，多数人只需包含底部部分：

```
make [2]: *** [make] 错误 1
```

仅凭这条信息不足以诊断问题，因为它仅表明出现了错误，而未指明具体错误原因。完整的上下文（如上文示例所示）应当被保存，因为它包含了执行的命令以及所有相关的错误信息。

关于如何在互联网上有效求助，有一篇出色的在线文章可供参考：<http://catb.org/~esr/faqs/smarty-questions.html>。

阅读该文档并遵循其中的建议，这将显著提高您获得所需帮助的可能性。

第二部分 构建前的准备工作

第二章 准备宿主系统

2.1. 简介

本章将检查构建 LFS 所需的宿主工具，并在必要时进行安装。随后我们将准备用于存放 LFS 系统的分区，包括创建分区本身、在其上建立文件系统并挂载该分区。

2.2. 宿主系统要求

2.2.1. 硬件要求 LFS 编辑团队建议系统 CPU 至少具备四核处理器，内存至少 8GB。不满足这些要求的旧系统仍可运行，但软件包编译时间将显著长于文档所述时长。

2.2.2. 软件要求 宿主系统应安装以下最低版本要求的软件。这对大多数现代 Linux 发行版不成问题。需注意许多发行版会将软件头文件放在单独的包中，通常命名为`-devel` 或 `-dev`。若发行版提供此类包，请务必安装。

所列软件包的早期版本可能可用，但未经测试验证。

- Bash-3.2 (`/bin/sh` 应是 `bash` 的符号链接或硬链接)
- Binutils-2.13.1 (不建议使用高于 2.44 的版本，因未经测试)
- Bison-2.7 (`/usr/bin/yacc` 应链接至 `bison` 或设置为调用 `bison` 的小脚本)
- Coreutils-8.1
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-4.0.1 (`/usr/bin/awk` 应链接至 `gawk`)
- GCC-5.2 (包含 C++ 编译器 `g++`，不建议使用高于 14.2.0 的版本，因未经测试)。必须同时安装 C 和 C++ 标准库 (含头文件)，以便 C++ 编译器能构建宿主程序
- Grep-2.5.1a
- Gzip-1.3.12
- Linux 内核-5.4

设定该内核版本要求的原因是：我们在第五章和第八章构建 glibc 时指定了该版本，因此不会启用针对旧版内核的兼容方案，这样编译出的 glibc 会略微更快且更小。截至 2024 年 12 月，5.4 版本仍是内核开发者维护的最旧发行版。某些早于 5.4 版本的内核可能仍由第三方团队维护，但它们不被视为官方上游

内核版本；详情请参阅 <https://kernel.org/category/releases.html>。

如果主机内核版本早于 5.4，您需要将其替换为更新的版本。有两种方法可以实现这一点。首先，查看您的 Linux 供应商是否提供 5.4 或更高版本的内核包。如果有，您可以选择安装它。如果供应商未提供合适的内核包，或者您不想安装供应商提供的内核，您可以自行编译内核。编译内核及配置引导加载程序（假设主机使用 GRUB）的相关说明位于第 10 章。

我们要求宿主内核支持 UNIX 98 伪终端(PTY)。所有搭载 Linux 5.4 或更新内核的桌面/服务器发行版都应默认启用该功能。若您正在构建自定义宿主内核，请确保在内核配置中将 `UNIX98_PTYS` 参数设置为 `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

重要提示

请注意，上述符号链接是使用本书所含指令构建 LFS 系统所必需的。指向其他软件（如 dash、mawk 等）的符号链接或许能工作，但未经 LFS 开发团队测试或支持，可能需要偏离指令或对某些软件包应用额外补丁。

要检查宿主系统是否具备所有合适版本及程序编译能力，请运行以下命令：

```
cat > version-check.sh << "EOF"
#!/bin/bash # 列出关键开发工具版本号的脚本

# 若工具安装在其他目录，请在此处调整 PATH 变量 # 并同步修改~lfs/.bashrc 文件（参见 4.4 节）
```

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

bail() { echo "致命错误: $1"; exit 1; } grep --version > /dev/null 2> /dev/null || bail "grep
无法正常工作" sed '' /dev/null || bail "sed 无法正常工作" sort /dev/null || bail "sort 无法正
常工作"
```

版本检查()

```
{ if ! type -p $2 &>/dev/null then

echo "错误: 找不到 $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 "通
过: %-9s %-6s >= $3\n" "$1" "$v"; return 0; else printf "错误: %-9s 版本过旧 (需要$3 或更高版
本)\n" "$1"; return 1; fi }
```

内核版本
{

```
kver=$(uname -r | grep -E -o '[0-9\.\.]+') if printf '%s\n' $1 $kver | sort --version-sort --check &>/dev/null
then printf "OK: Linux 内核版本 $kver >= $1\n"; return 0; else printf "错误: Linux 内核版本 ($kver) 过低 (需要 $1
或更高版本)\n" "$kver"; return 1; fi }
```

```
# 首先检查 Coreutils, 因为--version-sort 需要 Coreutils >= 7.0
版本检查 Coreutils sort 8.1 || 终止 "Coreutils 版本过低, 终止"
版本检查 Bash bash 3.2
版本检查 Binutils ld 2.13.1
版本检查 Bison bison 2.7
版本检查 Diffutils diff 2.8.1
版本检查 Findutils find 4.2.31
版本检查 Gawk gawk 4.0.1
版本检查 GCC gcc 5.2
版本检查 "GCC (C++)" g++ 5.2
版本检查 Grep grep 2.5.1a
版本检查 Gzip gzip 1.3.12
版本检查 M4 m4 1.4.10
版本检查 Make make 4.0
版本检查 Patch patch 2.5.4
版本检查 Perl perl 5.8.8
版本检查 Python python3 3.4
版本检查 Sed sed 4.1.5
版本检查 Tar tar 1.22
版本检查 Texinfo texi2any 5.0
版本检查 Xz xz 5.0.0
内核版本检查 5.4
```

```
if mount | grep -q 'devpts on /dev/pts' && [ -e /dev/ptmx ] then echo "OK: Linux 内核支
持 UNIX 98 PTY"; else echo "错误: Linux 内核不支持 UNIX 98 PTY"; fi
```

```
别名检查() { if $1 --version 2>&1 | grep -qi $2 then printf "通过:
%-4s 是 $2\n" "$1"; else printf "错误: %-4s 不是 $2\n" "$1"; fi }
echo "别名检查:" 别名检查 awk GNU 别名检查 yacc Bison 别名检查 sh
Bash
```

```
echo "编译器检查:" if printf "int main() {}" | g++ -x
c++ then echo "通过: g++ 工作正常"; else echo "错误:
g++ 无法工作"; fi rm -f a.out
```

```
if [ "$(nproc)" = "" ]; then echo "错误: nproc 不可用或其输出为空" else echo "通过: nproc
报告当前可用逻辑核心数为 $(nproc)" fi
```

EOF

bash version-check.sh

2.3. 分阶段构建 LFS

LFS 设计为单次会话中完成构建。这意味着操作指南假设系统在整个过程中不会关机。但这并不要求必须一次性完成所有构建步骤。关键在于：当在不同阶段恢复 LFS 构建时，某些操作必须在重启后重复执行。

2.3.1. 第 1-4 章

这些章节需要在宿主系统上执行命令。重启后继续时，必须确保：

- 以 root 用户身份执行 2.4 节之后的操作时，必须为 root 用户设置 LFS 环境变量。

2.3.2. 第 5-6 章

- 必须挂载 `/mnt/lfs` 分区。
- 这两章必须作为用户 lfs 完成。在执行这些章节的任何任务前，必须执行 `su - lfs` 命令。如果不这样做，可能会将软件包安装到宿主机上，甚至可能导致宿主机无法使用。
- 《通用编译指南》中的步骤至关重要。如果对软件包是否正确安装存在任何疑问，请确保删除之前解压的压缩包，然后重新解压该软件包，并完整执行该章节的所有指令。

2.3.3. 第 7-10 章

- 必须挂载/mnt/lfs 分区。
- 从"准备虚拟内核文件系统"到"进入 Chroot 环境"的若干操作，必须以 root 用户身份执行，并为 root 用户设置 LFS 环境变量。
- 进入 chroot 时，必须为 root 设置 LFS 环境变量。进入 chroot 环境后就不再使用 LFS 变量。
- 必须挂载虚拟文件系统。这可以在进入 chroot 之前或之后完成，方法是切换到主机虚拟终端，以 root 身份运行 7.3.1 节"挂载并填充/dev"和 7.3.2 节"挂载虚拟内核文件系统"中的命令。

2.4. 创建新分区

与大多数其他操作系统类似，LFS 通常需要安装在独立分区上。构建 LFS 系统的推荐方法是使用现有的空闲分区，或者如果您有足够的未分配空间，可以新建一个分区。

最小系统需要约 10GB 的分区空间，这足以存储所有源代码包并进行编译。但如果 LFS 系统计划作为主 Linux 系统使用，可能还需要安装额外软件，这将需要更多空间。30GB 的分区容量能合理满足后续扩展需求——LFS 系统本身并不会占用如此大的空间，该容量要求主要是为了确保临时存储空间充足，并支持完成 LFS 后添加额外功能。此外，编译软件包时可能暂时需要大量磁盘空间，这些空间在软件包安装完成后可被回收利用。

由于编译过程并不总是有足够的随机存取内存（RAM）可用，使用一个小型磁盘分区作为交换空间是个不错的选择。内核会利用该空间存储不常使用的数据，从而为活跃进程释放更多内存。LFS 系统的交换分区可以与宿主系统共用同一个，这种情况下就无需再创建新的交换分区。

启动磁盘分区工具（如 cfdisk 或 fdisk），通过命令行参数指定要创建新分区的硬盘——例如主磁盘驱动器为/dev/sda。根据需要创建一个 Linux 原生分区和一个交换分区。若您尚不熟悉这些工具的使用方法，请参阅 cfdisk(8)或 fdisk(8)手册页。

注意

对于有经验的用户，也可以采用其他分区方案。新的 LFS 系统可以安装在软件 RAID 阵列或 LVM 逻辑卷上。但请注意，部分方案需要 initramfs 支持，这属于高级主题。不建议首次构建 LFS 系统的用户采用这些分区方法。

请记录新分区的设备标识（例如 sda5），本书将称其为 LFS 分区。同时请记下交换分区的设备标识，后续配置/etc/fstab 文件时需要用到这些名称。

2.4.1. 其他分区注意事项

关于系统分区的建议请求经常出现在 LFS 邮件列表中。这是一个高度主观的话题。

大多数发行版的默认设置是使用整个驱动器，仅保留一个小的交换分区。这对 LFS 来说并非最佳选择，原因如下：它会降低灵活性，使得跨多个发行版或 LFS 构建共享数据更加困难，增加备份耗时，并可能因文件系统结构分配不当而浪费磁盘空间。

2.4.1.1. 根分区

对于大多数系统而言，20GB 的 LFS 根分区（不要与/root 目录混淆）是一个理想的折中方案。它既提供了足够的空间来构建 LFS 和大部分 BLFS，又足够紧凑以便轻松创建多个分区进行实验。

2.4.1.2. 交换分区

多数发行版会自动创建交换分区。通常建议交换分区大小为物理内存的两倍左右，但实际很少需要这么大。若磁盘空间有限，可将交换分区控制在 2GB 以内，并监控磁盘交换量。

若想使用 Linux 的休眠功能（挂起到磁盘），系统会在关机前将内存内容写入交换分区。此时交换分区的大小至少应与系统安装的内存容量相当。

交换行为绝非好事。对于机械硬盘，通常只需监听磁盘活动声响并观察系统对命令的响应速度，就能判断是否发生了交换。而使用固态硬盘时虽听不见交换声响，但可通过运行 top 或 free 程序查看交换空间使用量。应尽可能避免将 SSD 用作交换分区。当出现交换现象时，首先应检查是否执行了不合理命令（例如尝试编辑 5GB 的大文件）。若交换成为常态，最佳解决方案是为系统扩充内存容量。

2.4.1.3. GRUB BIOS 分区

若启动盘采用 GUID 分区表(GPT)进行分区，则需创建一个通常为 1MB 的小分区（若该分区不存在）。此分区无需格式化，但必须可供 GRUB 在安装引导加载程序时使用。使用 fdisk 工具时该分区通常标记为'BIOS Boot'，使用 gdisk 命令时其代码应为 EF02。

注意

GRUB BIOS 分区必须位于 BIOS 用于启动系统的驱动器上，该驱动器不一定是包含 LFS 根分区的驱动器。系统中的磁盘可能采用不同类型的分区表。GRUB BIOS 分区的必要性仅取决于启动盘的分区表类型。

2.4.1.4. 便利性分区 设计磁盘布局时，还有其他几种非必需但值得考虑的分区。以下列表虽非详尽无遗，但可作为参考指南。

- `/boot` – 强烈推荐。该分区用于存储内核及其他启动信息。为避免大容量磁盘可能引发的启动问题，建议将其设为第一块磁盘的首个物理分区。200MB 的分区容量已足够使用。
- `/boot/efi` – EFI 系统分区，采用 UEFI 方式启动系统时必需。详情请参阅 BLFS 页面说明。
- `/home` – 强烈推荐。用于在多发行版或 LFS 构建之间共享主目录及用户个性化配置。该分区通常需要较大容量，具体大小取决于可用磁盘空间。
- `/usr` – 在 LFS 中，`/bin`、`/lib` 和 `/sbin` 都是指向 `/usr` 中对应目录的符号链接。因此 `/usr` 包含了系统运行所需的所有二进制文件。对于 LFS 系统，通常不需要为 `/usr` 单独分区。若坚持创建，则应确保分区容量足以容纳系统中所有程序和库文件。在此配置下，根分区可以非常小（可能仅需 1GB），因此适合瘦客户端或无盘工作站（此时 `/usr` 通过远程服务器挂载）。但需注意，这种独立 `/usr` 分区配置需要 initramfs（LFS 未涵盖该内容）才能启动系统。
- `/opt` – 该目录对 BLFS 最为实用，可安装 KDE 或 Texlive 等大型软件包而无需将文件嵌入 `/usr` 层级结构。若使用此目录，通常 5 到 10GB 空间即可满足需求。
- `/tmp` – 单独划分 `/tmp` 分区较为罕见，但对配置瘦客户端很有帮助。若使用该分区，容量一般不超过几 GB 即可。若内存充足，可将 `tmpfs` 挂载到 `/tmp` 以加速临时文件访问。
- `/usr/src` —— 该分区非常实用，可为存储 BLFS 源码文件提供位置，并支持在多个 LFS 构建之间共享这些文件。它也可用作构建 BLFS 软件包的场所。30-50GB 的合理大分区能提供充足空间。

任何需要在系统启动时自动挂载的独立分区，都必须在 `/etc/fstab` 文件中进行指定。关于如何指定分区的详细信息将在第 10.2 节“创建 `/etc/fstab` 文件”中讨论。

2.5. 在分区上创建文件系统

分区仅是磁盘驱动器上由分区表设定的边界所限定的扇区范围。在操作系统能使用分区存储文件之前，必须将分区格式化为包含文件系统的形式——通常由卷标、目录块、数据块及按需定位特定文件的索引方案构成。文件系统还协助操作系统跟踪分区剩余空间，在创建新文件或扩展现有文件时预留所需扇区，并回收删除文件时产生的空闲数据段。它还可能提供数据冗余和错误恢复支持。

LFS 可以使用 Linux 内核支持的任何文件系统，但最常用的类型是 ext3 和 ext4。选择合适的文件系统可能比较复杂，这取决于文件特性和分区大小。例如：

ext2

适用于更新频率较低的小型分区，例如 /boot。

ext3 是 ext2 的升级版，增加了日志功能，可在非正常关机时帮助恢复分区状态。

它通常被用作通用文件系统。

ext4 是 ext 系列文件系统的最新版本。它提供了多项新功能，包括纳秒级时间戳、超大文件（最大 16TB）的创建与使用，以及速度提升。

其他文件系统如 FAT32、NTFS、JFS 和 XFS 适用于特定用途。更多关于这些文件系统的信息可访问：

https://en.wikipedia.org/wiki/Comparison_of_file_systems

LFS 假定根文件系统(/)采用 ext4 类型。要在 LFS 分区上创建 ext4 文件系统，请执行以下命令：

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

将替换为 LFS 分区的名称。

如果正在使用现有的交换分区，则无需格式化。如果是新建的交换分区，则需要用以下命令进行初始化：

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

将替换为交换分区的名称。

2.6. 设置\$LFS 变量与 umask 值

在本书中，我们会多次使用 LFS 环境变量。请确保在整个 LFS 构建过程中始终定义该变量。该变量应设置为构建 LFS 系统的目录名称——我们将以 /mnt/lfs 为例，但您可以选择任意目录名称。如果在独立分区上构建 LFS，该目录将作为分区的挂载点。请选择一个目录

位置并通过以下命令设置变量：

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

设置此变量的好处在于，可以原样输入诸如 `mkdir -v \$LFS/tools` 这样的命令。当 shell 处理命令行时，会自动将 "\$LFS" 替换为 "/mnt/lfs"（或该变量被设置的其他值）。

现在将文件模式创建掩码（umask）设置为 022，以防宿主发行版使用不同的默认值：

```
umask 022
```

将 umask 设为 022 能确保新创建的文件和目录仅对其所有者可写，但对所有用户可读且可搜索（仅针对目录）（假设 open(2) 系统调用使用默认模式，新文件最终将具有 644 权限模式，目录具有 755 权限模式）。过于宽松的默认设置会在 LFS 系统中留下安全漏洞，而过于严格的默认设置则可能导致构建或使用 LFS 系统时出现奇怪问题。

警告

无论何时离开并重新进入当前工作环境（例如使用 `su` 切换到 root 或其他用户时），切勿忘记检查 LFS 变量是否已设置且 `umask` 值是否为 022。可通过以下命令检查 LFS 变量是否正确设置：

```
echo $LFS
```

请确保输出显示的是 LFS 系统的构建路径，若遵循了提供的示例，则应为 `/mnt/lfs`。

使用以下命令检查 `umask` 设置是否正确：

```
umask
```

输出结果可能是 0022 或 022（前导零的数量取决于宿主发行版）。

如果这两个命令的任何输出不正确，请使用本页前面给出的命令将 `$LFS` 设置为正确的目录名称，并将 `umask` 设为 022。

注意

确保 LFS 变量和 `umask` 始终正确设置的一种方法是，在您个人主目录下的 `.bash_profile` 文件以及 `/root/.bash_profile` 文件中编辑并添加上文提到的 `export` 和 `umask` 命令。此外，所有需要 LFS 变量的用户在 `/etc/passwd` 文件中指定的 shell 必须是 `bash`，以确保 `.bash_profile` 文件能作为登录流程的一部分被加载。

另一个需要考虑的因素是登录主机系统的方式。如果通过图形显示管理器登录，当启动虚拟终端时，用户的 `.bash_profile` 通常不会被加载。这种情况下，需要将相关命令添加到用户和 root 的 `.bashrc` 文件中。另外，某些发行版会使用“if”条件测试，对于非交互式 `bash` 调用不会执行 `.bashrc` 中剩余的指令。请务必在这些命令放置在非交互式使用的条件测试之前。

2.7. 挂载新分区

既然文件系统已经创建完成，现在必须挂载该分区以便主机系统能够访问它。本书假设该文件系统将挂载到前文所述 LFS 环境变量指定的目录下。

严格来说，人们无法“挂载分区”，实际挂载的是该分区内嵌的文件系统。但由于单个分区不能包含多个文件系统，人们常将分区与其关联的文件系统视为同一事物。

使用以下命令创建挂载点并挂载 LFS 文件系统：

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

请将替换为 LFS 分区的名称。

如果你为 LFS 使用多个分区（例如一个用于`/`，另一个用于`/home`），请按以下方式挂载它们：

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

将和替换为适当的分区名称。

将\$LFS 目录（即新创建的 LFS 系统文件系统中的根目录）的所有者和权限模式设置为 root 和 755，以防主机发行版为 mkfs 配置了不同的默认值：

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

确保这个新分区没有以过于严格的权限挂载（例如 nosuid 或 nodev 选项）。运行不带任何参数的 mount 命令，查看已挂载的 LFS 分区设置了哪些选项。

如果设置了 nosuid 和/或 nodev 选项，必须重新挂载该分区。

警告

上述说明假设您在 LFS 构建过程中不会重启计算机。如果关闭系统，您需要在每次重新开始构建过程时重新挂载 LFS 分区，或者修改宿主系统的/etc/fstab 文件使其在重启时自动挂载。例如，您可以在/etc/fstab 文件中添加如下行：

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

如果使用了额外的可选分区，请确保也添加它们。

如果正在使用交换分区，请确保通过 swapon 命令启用它：

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

将 替换为交换分区的名称。

现在新的 LFS 分区已准备就绪，是时候下载软件包了。

第 3 章 软件包与补丁

3.1. 简介

本章列出了构建基础 Linux 系统需要下载的软件包清单。所列版本号均经过验证可正常使用，本书内容基于这些版本编写。我们强烈建议不要使用其他版本，因为针对某个版本的构建指令可能不适用于其他版本（除非该版本由 LFS 勘误或安全公告特别指定）。最新版本的软件包可能存在需要规避的问题，这些规避方案将在本书的开发版本中逐步完善并稳定。

对于某些软件包，其发布压缩包和对应版本的（Git 或 SVN）代码库快照压缩包可能会使用相似甚至完全相同的文件名。但发布压缩包除了包含对应代码库快照的内容外，还可能包含一些虽未存储在代码库中却至关重要的文件（例如由 autoconf 生成的 configure 脚本）。本书在可能的情况下均使用发布压缩包。若使用代码库快照替代本书指定的发布压缩包，将会引发问题。

下载地址可能并非始终可用。若某个下载地址自本书发布后已变更，可通过谷歌搜索引擎

[\(https://www.google.com/\)](https://www.google.com/) 查找大多数软件包。若搜索未果，请尝试访问

<https://www.linuxfromscratch.org/lfs/mirrors.html#files> 获取其他下载方式。

下载的软件包和补丁需要存放在整个构建过程中便于访问的位置。同时还需要一个工作目录来解压源码并进行构建。`$LFS/sources` 目录既可以用来存放压缩包和补丁文件，也可以作为工作目录使用。通过使用该目录，所需的文件都将位于 LFS 分区上，并在构建过程的所有阶段均可访问。

请在开始下载前，以 root 用户身份执行以下命令创建该目录：

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

将该目录设置为可写并启用粘滞位。“粘滞位”意味着即使多个用户对目录拥有写权限，在启用粘滞位的目录中只有文件所有者才能删除该文件。以下命令将启用写权限和粘滞位模式：

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

有几种方法可以获取构建 LFS 所需的所有软件包和补丁：

- 如后两节所述，这些文件可以单独下载。
- 对于稳定版本的书籍，可以从 <https://www.linuxfromscratch.org/mirrors.html#files> 列出的镜像站点下载包含所有必需文件的压缩包。
- 可以通过如下所述的 wget 命令和 wget-list 文件下载所需文件。

要使用 wget-list-sysv 作为 wget 命令的输入来下载所有软件包和补丁，请执行：

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

此外，从 LFS-7.0 版本开始，提供了一个单独的 md5sums 校验文件，用于在继续操作前验证所有软件包是否完整无误。请将该文件放入\$LFS/sources 目录并运行：

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

在使用上述任意方法获取所需文件后，均可进行此项校验。

如果以非 root 用户身份下载软件包和补丁，这些文件将归属于该用户。文件系统通过 UID 记录所有者，而宿主发行版中普通用户的 UID 在 LFS 系统中未被分配。因此这些文件在最终的 LFS 系统中将归属于一个未命名的 UID。如果您不打算在 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更改为 root 以避免此问题：

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

3.2. 所有软件包说明

下载软件包前请阅读安全公告，以确定是否需要使用某些软件包的更新版本以避免安全漏洞。

上游源代码可能会移除旧版本发布，特别是当这些版本存在安全漏洞时。如果下方某个 URL 无法访问，您应首先查阅安全公告，确认是否应使用修复了漏洞的新版本。若无需更新版本，可尝试从镜像站点下载被移除的软件包。请注意：即使能从镜像获取因漏洞被移除的旧版本，但在构建系统时使用已知存在漏洞的版本绝非明智之举。

请下载或通过其他方式获取以下软件包：

- Acl (2.3.2) - 363 KB:

主页: <https://savannah.nongnu.org/projects/acl> 下载:
<https://download.savannah.gnu.org/releases/acl/acl-2.3.2.tar.xz>
MD5 校验值: 590765dee95907dbc3c856f7255bd669

- 属性工具包(2.5.2) - 484 KB:

主页: <https://savannah.nongnu.org/projects/attr> 下载:
<https://download.savannah.gnu.org/releases/attr/attr-2.5.2.tar.gz>
MD5 校验值: 227043ec2f6ca03c0948df5517f9c927

- Autoconf (2.72 版) - 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 校验值: 1be79f7106ab6767f18391c5e22be701

- Automake (1.17 版) - 1,614 KB:

主页: <https://www.gnu.org/software/automake/> 下载地址:
<https://ftp.gnu.org/gnu/automake/automake-1.17.tar.xz>
MD5 校验值: 7ab3a02318fee6f5bd42adfc369abf10

- Bash (5.2.37 版) - 10,868 KB:

主页: <https://www.gnu.org/software/bash/>
下载地址: <https://ftp.gnu.org/gnu/bash/bash-5.2.tar.gz>
MD5 校验值: f21ff65de72ca329c1779684a972

- Bc (7.0.3) - 464 KB:

主页: <https://git.gavinhoward.com/gavin/bc>
下载地址: <https://github.com/gavinhoward/bc/releases/download/7.0.3/bc-7.0.3.tar.xz>
MD5 校验值: 4db5a0eb4fdbb3f6813be4b6b3da74

- Binutils (2.44 版) - 26,647 KB:

主页: <https://www.gnu.org/software/binutils/> 下载地址:
<https://sourceware.org/pub/binutils/releases/binutils-2.44.tar.xz>
MD5 校验值: 49912ce774666a30806141f106124294

- Bison (3.8.2) - 2,752 KB:

官网: <https://www.gnu.org/software/bison/> 下载地
址: <https://ftp.gnu.org/gnu/bison/bison-3.8.2.tar.xz>
MD5 校验值: c28f119f405a2304ff0a7ccdcc629713

- Bzip2 压缩工具 (1.0.8 版) - 792 KB:

下载地址: <https://www.sourceware.org/pub/bzip2/bzip2-1.0.8.tar.gz>
MD5 校验值: 67e051268d0c475ea773822f7500d0e5

- Check (0.15.2) - 760 KB:

主页: <https://libcheck.github.io/check> 下载地址:
<https://github.com/libcheck/check/releases/download/0.15.2/check-0.15.2.tar.gz>
MD5 校验值: 50fcacfecde5a380415b12e9c574e0b2

- Coreutils (9.6) - 5,991 KB:

主页: <https://www.gnu.org/software/coreutils/> 下载:
<https://ftp.gnu.org/gnu/coreutils/coreutils-9.6.tar.xz>
MD5 校验值: 0ed6cc983fe02973bc98803155cc1733

- DejaGNU (1.6.3) - 608 KB:

主页: <https://www.gnu.org/software/dejagnu/> 下载:
<https://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.3.tar.gz>
MD5 校验值: 68c5208c58236eba447d7d6d1326b821

- Diffutils (3.11 版) - 1,881 KB:

主页: <https://www.gnu.org/software/diffutils/> 下载地址:
<https://ftp.gnu.org/gnu/diffutils/diffutils-3.11.tar.xz>
MD5 校验值: 75ab2bb7b5ac0e3e10cece85bd1780c2

- E2fsprogs (1.47.2) - 9,763 KB:

主页: <https://e2fsprogs.sourceforge.net/> 下载地址:
<https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.47.2/e2fsprogs-1.47.2.tar.gz>
MD5 校验值: 752e5a3ce19aea060d8a203f2fae9baa

- Elfutils (0.192) - 11,635 KB:

主页: <https://sourceware.org/elfutils/> 下载地址:
<https://sourceware.org/ftp/elfutils/0.192/elfutils-0.192.tar.bz2>
MD5 校验值: a6bb1efc147302cfcc15b5c2b827f186a

- Expat (2.6.4) - 476 KB:

主页: <https://libexpat.github.io/> 下载地址:
<https://prdownloads.sourceforge.net/expat/expat-2.6.4.tar.xz>
MD5 校验值: 101fe3e320a2800f36af8cf4045b45c7

- Expect 软件包(5.45.4 版) - 618 KB:

主页: <https://core.tcl.tk/expect/> 下载地址:
<https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz>
MD5 校验值: 00fce8de158422f5cccd2666512329bd2

· 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 校验值: 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 校验值: 870cf71c07d37ebe56f9f4aaf4ad872

· 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 校验值: 2882e3179748cc9f9c23ec593d6adc8d

· Flit-core (3.11.0) - 51 KB:

主页: <https://pypi.org/project/flit-core/> 下载:

https://pypi.org/packages/source/f/flit-core/flit_core-3.11.0.tar.gz

MD5 校验值: 6d677b1acef1769c4c7156c7508e0dbd

· Gawk (5.3.1) - 3,428 KB:

主页: <https://www.gnu.org/software/gawk/> 下载:

<https://ftp.gnu.org/gnu/gawk/gawk-5.3.1.tar.xz>

MD5 校验值: 4e9292a06b43694500e0620851762eec

· GCC (14.2.0) - 90,144 KB:

主页: <https://gcc.gnu.org/> 下载地址:

<https://ftp.gnu.org/gnu/gcc/gcc-14.2.0/gcc-14.2.0.tar.xz>

MD5 校验值: 2268420ba02dc01821960e274711bde0

· GDBM (1.24 版) - 1,168 KB:

主页: <https://www.gnu.org/software/gdbm/> 下载地

址: <https://ftp.gnu.org/gnu/gdbm/gdbm-1.24.tar.gz>

MD5 校验值: c780815649e52317be48331c1773e987

· Gettext (0.24 版) - 8,120 KB:

主页: <https://www.gnu.org/software/gettext/> 下载地址:

<https://ftp.gnu.org/gnu/gettext/gettext-0.24.tar.xz>

MD5 校验值: 87aea3013802a3c60fa3feb5c7164069

· Glibc (2.41) - 18,892 KB:

主页: <https://www.gnu.org/software/libc/> 下载地址:

<https://ftp.gnu.org/gnu/glibc/glibc-2.41.tar.xz>

MD5 校验值: 19862601af60f73ac69e067d3e9267d4

注意

Glibc 开发者维护着一个 Git 分支，其中包含针对 Glibc-2.41 值得采用的补丁，但这些补丁不幸在 Glibc-2.41 发布后才开发完成。若该分支中添加了任何安全修复补丁，LFS 编辑团队将发布安全公告；对于其他新增补丁则不会采取特别措施。您可以自行审查这些补丁，若认为某些补丁重要，可自行将其整合使用。

· 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 校验值: 956dc04e864001a9c22429f761f2c283

· Gperf (3.1) - 1,188 KB:

主页: <https://www.gnu.org/software/gperf/> 下载地址: <https://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz>

MD5 校验值: 9e251c0a618ad0824b51117d5d9db87e

· Grep (3.11) - 1,664 KB:

主页: <https://www.gnu.org/software/grep/> 下载地址: <https://ftp.gnu.org/gnu/grep/grep-3.11.tar.xz>

MD5 校验值: 7c9bbd74492131245f7cdb291fa142c0

· 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 校验值: 5e4f40315a22bb8a158748e7d5094c7d

· GRUB (2.12 版) - 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 校验值: 60c564b1bdc39d8e43b3aab4bc0fb140

· Gzip (1.13 版) - 819 KB:

主页: <https://www.gnu.org/software/gzip/> 下载地址: <https://ftp.gnu.org/gnu/gzip/gzip-1.13.tar.xz>

MD5 校验值: d5c9fc9441288817a4a0be2da0249e29

· Iana-Etc (20250123) - 591 KB:

主页: <https://www.iana.org/protocols> 下载地址: <https://github.com/Mic92/iana-etc/releases/download/20250123/iana-etc-20250123.tar.gz>

MD5 校验值: f8a0ebdc19a5004cf42d8bdcf614fa5d

· Inetutils 工具包(2.6 版) - 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 校验值: 401d7d07682a193960bcddecaf0d03de94

· 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 校验值: 12e517cac2b57a0121cda351570f1e63

· IPRoute2 (6.13.0) - 906 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/iproute2-6.13.0.tar.xz>

- **Jinja2 (3.1.5) - 239 KB:**

主页: <https://jinja.palletsprojects.com/en/3.1.x/> 下载:
<https://pypi.org/packages/source/J/Jinja2/jinja2-3.1.5.tar.gz>

MD5 校验值: 083d64f070f6f1b5f75971ae60240785

- **Kbd (2.7.1) - 1,438 KB:**

主页: <https://kbd-project.org/> 下载:
<https://www.kernel.org/pub/linux/utils/kbd/kbd-2.7.1.tar.xz>

MD5 校验值: f15673d9f748e58f82fa50cff0d0fd20

- **Kmod (34) - 331 KB:**

主页: <https://github.com/kmod-project/kmod> 下载地址:
<https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-34.tar.xz>

MD5 校验值: 3e6c5c9ad9c7367ab9c3cc4f08dfde62

- **Less (668) - 635 KB:**

主页: <https://www.greenwoodsoftware.com/less/> 下载:
<https://www.greenwoodsoftware.com/less/less-668.tar.gz>

MD5 校验值: d72760386c5f80702890340d2f66c302

- **LFS 启动脚本 (20240825) - 33 KB:**

下载地址: <https://www.linuxfromscratch.org/lfs/downloads/12.3/lfs-bootscripts-20240825.tar.xz>
MD5 校验值: 7b078c594a77e0f9cd53a0027471c3bc

- **Libcap (2.73 版) - 191 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-2.73.tar.xz>

MD5 校验值: 0e186df9de9b1e925593a96684fe2e32

- **Libffi (3.4.7 版) - 1,362 KB:**

主页: <https://sourceware.org/libffi/> 下载地址:
<https://github.com/libffi/libffi/releases/download/v3.4.7/libffi-3.4.7.tar.gz>

MD5 校验值: 696a1d483a1174ce8a477575546a5284

- **Libpipeline (1.5.8) - 1046 KB:**

主页: <https://libpipeline.nongnu.org/> 下载:
<https://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.8.tar.gz>

MD5 校验值: 17ac6969b2015386bcb5d278a08a40b5

- **Libtool (2.5.4) - 1,033 KB:**

主页: <https://www.gnu.org/software/libtool/> 下载:
<https://ftp.gnu.org/gnu/libtool/libtool-2.5.4.tar.xz>

MD5 校验值: 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 校验值: 1796a5d20098e9dd9e3f576803c83000

- Linux (6.13.4) - 145,015 KB:

主页: <https://www.kernel.org/> 下载地址:

<https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.13.4.tar.xz>

MD5 校验值: 13b9e6c29105a34db4647190a43d1810

注意

Linux 内核更新非常频繁，多数情况是由于发现安全漏洞所致。除非勘误页另有说明，否则建议使用最新的稳定内核版本。对于网速较慢或带宽费用较高的用户，若需更新 Linux 内核，可分别下载该软件包的基础版本和补丁文件。这种方式在次要版本内进行补丁级别升级时，可能节省后续更新时间或成本。

- 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 校验值: dead9f5f1966d9ae56e1e32761e4e675

- M4 (1.4.19) - 1,617 KB:

主页: <https://www.gnu.org/software/m4/> 下载地

址: <https://ftp.gnu.org/gnu/m4/m4-1.4.19.tar.xz>

MD5 校验值: 0d90823e1426f1da2fd872df0311298d

- 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 校验值: c8469a3713cbbe04d955d4ae4be23eeb

- Man-DB (2.13.0) - 2,023 KB:

主页: <https://www.nongnu.org/man-db/> 下载地址:

<https://download.savannah.gnu.org/releases/man-db/man-db-2.13.0.tar.xz>

MD5 校验值: 97ab5f9f32914eef2062d867381d8cee

- Man-pages (6.12) - 1,838 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-6.12.tar.xz>

MD5 校验值: 44de430a598605eaba3e36dd43f24298

- 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 校验值: cb0071711b573b155cc8f86e1de72167

- Meson (1.7.0) - 2,241 KB:

主页: <https://mesonbuild.com> 下载地址:

<https://github.com/mesonbuild/meson/releases/download/1.7.0/meson-1.7.0.tar.gz>

MD5 校验值: c20f3e5ebbb007352d22f4fd6ceb925c

- 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 校验值: 5c9bc658c9fd0f940e8e3e0f09530c62

· MPFR (4.2.1) - 1,459 KB:

主页: <https://www.mpfr.org/> 下载地址:

<https://ftp.gnu.org/gnu/mpfr/mpfr-4.2.1.tar.xz>

MD5 校验值: 523c50c6318dde6f9dc523bc0244690a

· Ncurses (6.5) - 2,156 KB:

主页: <https://www.gnu.org/software/ncurses/> 下载:

<https://invisible-mirror.net/archives/ncurses/ncurses-6.5.tar.gz>

MD5 校验值: ac2d2629296f04c8537ca706b6977687

· Ninja (1.12.1) - 235 KB:

主页: <https://ninja-build.org/> 下载: <https://github.com/ninja-build/ninja/archive/v1.12.1/ninja-1.12.1.tar.gz>

MD5 校验值: 6288992b05e593a391599692e2f7e490

· OpenSSL (3.4.1 版) - 17,917 KB:

主页: <https://www.openssl-library.org/> 下载地址:

<https://github.com/openssl/openssl/releases/download/openssl-3.4.1/openssl-3.4.1.tar.gz>

MD5 校验值: fb7a747ac6793a7ad7118eaba45db379

· 补丁工具 (2.7.6) - 766 KB:

主页: <https://savannah.gnu.org/projects/patch/> 下载地

址: <https://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz>

MD5 校验值: 78ad9937e4caadcba1526ef1853730d5

· Perl 语言 (5.40.1) - 13,605 KB:

官方网站: <https://www.perl.org/> 下载地址:

<https://www.cpan.org/src/5.0/perl-5.40.1.tar.xz>

MD5 校验值: bab3547a5cdf2302ee0396419d74a42e

· Pkgconf (2.3.0) - 309 KB:

官方网站: <https://github.com/pkgconf/pkgconf> 下载地址:

<https://distfiles.ariadne.space/pkgconf/pkgconf-2.3.0.tar.xz>

MD5 校验值: 833363e77b5bed0131c7bc4cc6f7747b

· Procps 软件包(4.0.5 版) - 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 校验值: 90803e64f51f192f3325d25c3335d057

· Psmisc (23.7 版) - 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 校验值: 53eae841735189a896d614cba440eb10

· Python (3.13.2 版) - 22,091 KB:

主页: <https://www.python.org/> 下载地址:

<https://www.python.org/ftp/python/3.13.2/Python-3.13.2.tar.xz>

MD5 校验值: 4c2d9202ab4db02c9d0999b14655dfe5

· Python 官方文档(3.13.2 版) - 10,102 KB:

下载地址: <https://www.python.org/ftp/python/doc/3.13.2/python-3.13.2-docs-html.tar.bz2>

MD5 校验值: d6aede88f480a018d26b3206f21654ae

- Readline (8.2.13) - 2,974 KB:

主页: <https://tiswww.case.edu/php/chet/readline/rltop.html>

下载: <https://ftp.gnu.org/gnu/readline/readline-8.2.13.tar.gz>

MD5 校验值: 05080bf3801e6874bb115cd6700b708f

- 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 校验值: 6aac9b2dbafcd5b7a67a8a9bcb8036c3

- Setuptools (75.8.1) - 1,313 KB:

主页: <https://pypi.org/project/setuptools/> 下载地址:

<https://pypi.org/packages/source/s/setuptools/setuptools-75.8.1.tar.gz>

MD5 校验值: 7dc3d3f529b76b10e35326e25c676b30

- Shadow 软件包 (4.17.3 版) - 2,274 KB:

主页: <https://github.com/shadow-maint/shadow/> 下载地址: <https://github.com/shadow-maint/shadow/releases/download/4.17.3/shadow-4.17.3.tar.xz>

MD5 校验值: 0da190e53ecee76237e4c8f3f39531ed

- Sysklogd (2.7.0) - 465 KB:

主页: <https://www.infodrom.org/projects/sysklogd/> 下载地址:

<https://github.com/troglbit/sysklogd/releases/download/v2.7.0/sysklogd-2.7.0.tar.gz>

MD5 校验值: 611c0fa5c138eb7a532f3c13bdf11ebc

- Systemd (257.3) - 15,847 KB:

主页: <https://www.freedesktop.org/wiki/Software/systemd/> 下载地址:

<https://github.com/systemd/systemd/archive/v257.3/systemd-257.3.tar.gz>

MD5 校验值: 8e4fc90c7ead651fa5c50bd1b34abc2

- Systemd 手册页 (257.3 版) - 733 KB:

主页: <https://www.freedesktop.org/wiki/Software/systemd/> 下载地址:

<https://anduin.linuxfromscratch.org/LFS/systemd-man-pages-257.3.tar.xz>

MD5 校验值: 9b77c3b066723d490cb10aed4fb05696

注意

Linux From Scratch 团队使用 systemd 源码自行生成了 man 手册页的压缩包，此举旨在避免引入不必要的依赖项。

- SysVinit (3.14 版) - 236 KB:

主页: <https://savannah.nongnu.org/projects/sysvinit> 下载地址:

<https://github.com/slicer69/sysvinit/releases/download/3.14/sysvinit-3.14.tar.xz>

MD5 校验值: bc6890b975d19dc9db42d0c7364dd092

- Tar (1.35 版) - 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 校验值: a2d8042658cf8ea939e6d911eaf4152

- **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 校验值: eaef5d0a27239fb840f04af8ec608242

- **Tcl 文档(8.6.16 版) - 1,169 KB:**

下载地址: <https://downloads.sourceforge.net/tcl/tcl8.6.16-html.tar.gz>

MD5 校验值: 750c221bcb6f8737a6791c1fbe98b684

- **Texinfo (7.2 版) - 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 校验值: 11939a7624572814912a18e76c8d8972

- **时区数据 (2025a) - 453 KB:**

主页: <https://www.iana.org/time-zones> 下载: <https://www.iana.org/time-zones/repository/releases/tzdata2025a.tar.gz>

MD5 校验值: 404229390c06b7440f5e48d12c1a3251

- **Udev-lfs 压缩包 (udev-lfs-20230818) - 10 KB:**

下载链接: <https://anduin.linuxfromscratch.org/LFS/udev-lfs-20230818.tar.xz>

MD5 校验值: acd4360d8a5c3ef320b9db88d275dae6

- **Util-linux 工具集 (2.40.4 版) - 8,641 KB:**

项目主页: <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/v2.40/util-linux-2.40.4.tar.xz>

MD5 校验值: b1c8315d8ccbeb0ec36d10350304

- **Vim 编辑器 (9.1.1166 版) - 18,077 KB:**

官网: <https://www.vim.org> 下载地址:

<https://github.com/vim/vim/archive/v9.1.1166/vim-9.1.1166.tar.gz>

MD5 校验值: 718d43ce957ab7c81071793de176c2eb

注意

vim 版本每日更新。获取最新版本请访问: <https://github.com/vim/vim/tags>

- **Wheel (0.45.1) - 106 KB:**

主页: <https://pypi.org/project/wheel/> 下载:

<https://pypi.org/packages/source/w/wheel/wheel-0.45.1.tar.gz>

MD5 校验值: dddc505d0573d03576c7c6c5a4fe0641

- **XML::Parser (2.47) - 276 KB:**

主页: <https://github.com/chorny/XML-Parser> 下载地址:

<https://cpn.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.47.tar.gz>

MD5 校验值: 89a8e82cf2ad948b349c0a69c494463

- **Xz 压缩工具 (5.6.4) - 1,310 KB:**

主页: <https://tukaani.org/xz> 下载地址: <https://github.com/tukaani-project/xz/releases/download/v5.6.4/xz-5.6.4.tar.xz>

MD5 校验值: 4b1cf07d45ec7eb90a01dd3c00311a3e

- 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 校验值: 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 校验值: 780fc1896922b1bc52a4e90980cdda48

这些软件包总大小: 约 527 MB

3.3. 所需补丁

除了软件包外, 还需要若干补丁程序。这些补丁用于修正软件包中本应由维护者修复的错误。补丁还会对软件包进行小幅修改, 使其更易于使用

构建 LFS 系统需要以下补丁:

- Bzip2 文档安装补丁 - 1.6 KB:

下载地址: https://www.linuxfromscratch.org/patches/lfs/12.3/bzip2-1.0.8-install_docs-1.patch
MD5 校验值: 6a5ac7e89b791aae556de0f745916f7f

- Coreutils 国际化修复补丁 - 164 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/coreutils-9.6-i18n-1.patch>
MD5 校验值: 6aee45dd3e05b7658971c321d92f44b7

- Expect GCC14 补丁 - 7.8 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/expect-5.45.4-gcc14-1.patch>
MD5 校验值: 0b8b5ac411d011263ad40b0664c669f0

- Glibc FHS 补丁 - 2.8 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/glibc-2.41-fhs-1.patch>
MD5 校验值: 9a5997c3452909b1769918c759eff8a2

- 键盘退格/删除键修复补丁 - 12 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/kbd-2.7.1-backspace-1.patch>
MD5 校验值: f75cca16a38da6caa7d52151f7136895

- SysVinit 综合补丁 - 2.5 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/sysvinit-3.14-consolidated-1.patch>
MD5 校验值: 3af8fd8e13cad481eeeeaa48be4247445

这些补丁的总大小: 约 190.7 KB

除了上述必需的补丁外, LFS 社区还创建了若干可选补丁。这些可选补丁能解决小问题或启用默认未开启的功能。欢迎浏览位于 <https://www.linuxfromscratch.org/patches/downloads/> 的补丁数据库, 根据系统需求获取额外补丁。

第 4 章 最终准备工作

4.1. 简介

本章将执行若干额外任务来为构建临时系统做准备。我们将在\$LFS 中创建一组目录（用于安装临时工具）、添加非特权用户，并为该用户创建合适的构建环境。同时会解释用于衡量 LFS 软件包构建耗时的“SBU”时间单位，并提供有关软件包测试套件的信息。

4.2. 在 LFS 文件系统中创建有限目录结构

本节开始向 LFS 文件系统填充构成最终 Linux 系统的各个组件。

第一步是创建一个有限的目录层次结构，以便将第 6 章编译的程序（以及第 5 章的 glibc 和 libstdc++）安装到它们的最终位置。这样做是为了在第 8 章构建最终版本时覆盖这些临时程序。

以 root 身份执行以下命令创建所需的目录结构：

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

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

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

第六章中的程序将使用交叉编译器进行编译（更多细节可参阅工具链技术说明章节）。这个交叉编译器会被安装在一个特殊目录中，以与其他程序隔离。仍以 root 身份执行以下命令创建该目录：

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

注意

LFS 开发团队特意决定不使用/usr/lib64 目录。我们采取了多项措施确保工具链不会使用该目录。如果该目录因任何原因出现（可能是由于您未遵循操作说明，或在完成 LFS 后安装了创建该目录的二进制包），可能会导致系统故障。您应始终确保该目录不存在。

4.3. 添加 LFS 用户

以 root 用户身份登录时，任何微小失误都可能导致系统损坏。因此，接下来两章的软件包将作为非特权用户进行构建。虽然可以使用您现有的用户名，但为了便于建立纯净的工作环境，我们将创建一个名为 lfs 的新用户（同时创建同名用户组 lfs），并在安装过程中以该用户身份执行命令。

以 root 身份执行以下命令添加新用户：

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

命令行参数含义如下：

-s /bin/bash

这将使 bash 成为用户 lfs 的默认 shell。

```
-g lfs
```

此选项将用户 lfs 添加到 lfs 组中。

-m

这将为 lfs 用户创建主目录。

```
-k /dev/null
```

该参数通过将输入位置改为特殊的空设备，防止可能从骨架目录（默认为/etc/skel）复制文件。

lfs

这是新用户的名称。

如果您希望以 lfs 身份登录，或从非 root 用户切换至 lfs（与以 root 身份登录时切换至 lfs 用户不同，后者无需为 lfs 用户设置密码），则需要为 lfs 设置密码。请以 root 用户身份执行以下命令来设置密码：

```
passwd lfs
```

通过将\$LFS 下所有目录的所有权赋予 lfs 用户，授予其完全访问权限：

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

注意

在某些宿主系统中，以下 su 命令可能无法正常完成，并会将 lfs 用户的登录会话挂起到后台。若提示符“lfs:~\$”未立即出现，输入 fg 命令可解决此问题。

接下来，启动一个以用户 lfs 身份运行的 shell。这可以通过在虚拟控制台以 lfs 身份登录实现，或者使用以下替代/切换用户命令：

```
su - lfs
```

这里的“-”指示 su 启动一个登录 shell，而非非登录 shell。这两种 shell 类型的区别在 bash(1) 和 info bash 中有详细说明。

4.4. 设置环境 通过为 bash shell 创建两个新的启动文件来建立良好的工作环境。在以用户 lfs 身份登录时，

执行以下命令创建新的.bash_profile 文件：

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

当以 lfs 用户身份登录，或使用带“-”选项的 su 命令切换到 lfs 用户时，初始 shell 是登录 shell，它会先读取主机的/etc/profile 文件（可能包含某些设置和环境变量），然后读取.bash_profile。.bash_profile 文件中的 exec env - i.../bin/bash 命令会用全新的空环境替换当前运行的 shell，仅保留 HOME、TERM 和 PS1 变量。这确保不会从主机系统泄漏任何不必要且可能有害的环境变量到构建环境中。

新启动的 shell 实例是一个非登录 shell，它不会读取并执行/etc/profile 或.bash_profile 文件的内容，而是会读取并执行.bashrc 文件。现在创建.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 中设置项的含义

set +h

The set +h command turns off bash's hash function. Hashing is ordinarily a useful feature—bash uses a hash table to remember the full path to executable files to avoid searching the PATH time and again to find the same executable. However, the new tools should be used as soon as they are installed. Switching off the hash function forces the shell to search the PATH whenever a program is to be run. As such, the shell will find the newly compiled tools in \$LFS/tools/bin as soon as they are available without remembering a previous version of the same program provided by the host distro, in /usr/bin

umask 022

Setting the umask as we've already explained in Section 2.6, “Setting the \$LFS Variable and the Umask.”

The LFS variable should be set to the chosen mount point.

LC_ALL=POSIX

The LC_ALL variable controls the localization of certain programs, making their messages follow the conventions of a specified country. Setting LC_ALL to “POSIX” or “C” (the two are equivalent) ensures that everything will work as expected in the cross-compilation environment.

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

The LFS_TGT variable sets a non-default, but compatible machine description for use when building our crosscompiler and linker and when cross-compiling our temporary toolchain. More information is provided by Toolchain Technical Notes.

PATH=/usr/bin

Many modern Linux distributions have merged /bin and /usr/bin. When this is the case, the standard PATH variable should be set to /usr/bin/ for the Chapter 6 environment. When this is not the case, the following path adds /

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

If /bin is not a symbolic link, it must be added to the PATH variable.

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

By putting \$LFS/tools/bin ahead of the standard PATH, the cross-compiler installed at the beginning of Chapter 5 is picked up by the shell immediately after its installation. This, combined with turning off hashing, limits the risk that the compiler from the host is used instead of the cross-compiler.

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

In Chapter 5 and Chapter 6, if this variable is not set, configure scripts may attempt to load configuration items specific to some distributions from /usr/share/config.site on the host system. Override it to prevent potential contamination from the host.

```
export ...
```

While the preceding commands have set some variables, in order to make them visible within any sub-shells, we export them.

重要提示

Several commercial distributions add an undocumented instantiation of /etc/bash.bashrc to the initialization of bash. This file has the potential to modify the lfs user's environment in ways that can affect the building of critical LFS packages. To make sure the lfs user's environment is clean, bash.bashrc and, if present, move it out of the way. As the root user, run:

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

When the lfs user is no longer needed (at the beginning of Chapter 7), you may safely restore /etc/bash.bashrc if desired.

Note that the LFS Bash package we will build in Section 8.36, “Bash-5.2.37” is not configured to load or execute /etc/bash.bashrc, so this file is useless on a completed LFS system.

For many modern systems with multiple processors (or cores) the compilation time for a package can be reduced by performing a "parallel make" by telling the make program how many processors are available via a command line option or an environment variable. For instance, an Intel Core i9-13900K processor has 8 P (performance) cores and 16 E (efficiency) cores, and a P core can simultaneously run two threads so each P core are modeled as two logical cores by the Linux kernel. As the result there are 32 logical cores in total. One obvious way to use all these logical cores is allowing make to spawn up to 32 build jobs. This

```
make -j32
```

Or set the MAKEFLAGS environment variable and its content will be automatically used by make as command line options:

```
export MAKEFLAGS=-j32
```

重要提示

Never pass a -j option without a number to make or set such an option in MAKEFLAGS. Doing so will allow make to spawn infinite build jobs and cause system stability problems.

To use all logical cores available for building packages in Chapter 5 and Chapter 6, set MAKEFLAGS now in .bashrc:

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

Replace \$(nproc) with the number of logical cores you want to use if you don't want to use all the logical cores. Finally, to ensure the environment is fully prepared for building the temporary tools, force the bash shell to read the new user profile:

```
source ~/.bash_profile
```

4.5. About SBUs

Many people would like to know beforehand approximately how long it takes to compile and install each package.

Because Linux From Scratch can be built on many different systems, it is impossible to provide absolute time estimates. The biggest package (gcc) will take approximately 5 minutes on the fastest systems, but could take days on slower systems! Instead of providing actual times, the Standard Build Unit (SBU) measure will be used instead.

The SBU measure works as follows. The first package to be compiled is binutils in Chapter 5. The time it takes to compile using one core is what we will refer to as the Standard Build Unit or SBU. All other compile times will be expressed in terms of this unit of time.

For example, consider a package whose compilation time is 4.5 SBUs. This means that if your system took 4 minutes to compile and install the first pass of binutils, it will take approximately 18 minutes to build. Fortunately, most build times are shorter than one SBU.

SBUs are not entirely accurate because they depend on many factors, including the host system's version of GCC. They are provided here to give an estimate of how long it might take to install a package, but the numbers can vary by as much as dozens of minutes in some cases.

On some newer systems, the motherboard is capable of controlling the system clock speed. This can be controlled with a command such as powerprofilesctl. This is not available in LFS, but may be available on the host distro. After LFS is complete, it can be added to a system with the procedures at the BLFS power-profiles-daemon page. Before measuring the build time of any package it is advisable to use a system power profile set for maximum performance (and maximum power consumption). Otherwise the measured SBU value may be inaccurate because the system may react differently when building binutils-pass1 or other packages. Be aware that a significant inaccuracy can still show up even if the same profile is used for both packages because the system may respond slower if the system is idle when starting the build procedure. Setting the power profile to “performance” will minimize this problem. And obviously doing so will also make the system build LFS faster.

If powerprofilesctl is available, issue the powerprofilesctl set performance command to select the performance profile. Some distros provides the tuned-adm command for managing the profiles instead of powerprofilesctl, on these distros issue the tuned-adm profile throughput-performance command to select the throughput-performance profile.

注意

When multiple processors are used in this way, the SBU units in the book will vary even more than they normally would. In some cases, the make step will simply fail. Analyzing the output of the build process will also be more difficult because the lines from different processes will be interleaved. If you run into a problem with a build step, revert to a single processor build to properly analyze the error messages.

The times presented here for all packages (except binutils-pass1 which is based on one core) are based upon using four cores (-j4). The times in Chapter 8 also include the time to run the regression tests for the package unless specified otherwise.

4.6. About the Test Suites

Most packages provide a test suite. Running the test suite for a newly built package is a good idea because it can provide a “sanity check” indicating that everything compiled correctly. A test suite that passes its set of checks usually proves that the package is functioning as the developer intended. It does not, however, guarantee that the package is totally bug free.

Linux 从零开始

版本 12.3

2025 年 3 月 5 日发布

创建者：杰拉德 · 比克曼斯
执行编
辑：布鲁斯 · 杜布斯

Linux 从零开始：第 12.3 版：2025 年 3 月 5 日发布

作者：杰拉德·比克曼斯 主编：布鲁斯·杜布斯 版权所有 © 1999-2025

杰拉德·比克曼斯

版权所有 © 1999-2025，杰拉德·比克曼斯

保留所有权利。

本书采用知识共享许可协议授权。

书中涉及的计算机指令可依据 MIT 许可证提取使用。

Linux®是 Linus Torvalds 的注册商标。

目录

前言 viii i. 序言 viii ii. 读者对象 viii iii. LFS 目标架构 ix iv. 先决条件 x v. LFS 与标准 x vi. 本书选用软件包的理由 xi vii. 排版约定 xvii viii. 结构说明 xviii ix. 勘误与安全公告 xix I. 简介 1 1.1. 如何构建 LFS 系统 2

1.2. 自上一版本以来的更新内容 2
 1.3. 更新日志 4
 1.4. 资源 8
 1.5. 帮助 9
 II. 构建前的准备工作 11 2. 准备宿主系统 12 2.1. 简介 12

2.2. 宿主系统要求 12
 2.3. 分阶段构建 LFS 系统 15
 2.4. 创建新分区 15
 2.5. 在分区上创建文件系统 17
 2.6. 设置\$LFS 变量与 umask 值 18
 2.7. 挂载新分区 19
 3. 软件包与补丁 21 3.1. 简介 21
 3.2. 所有软件包 22
 3.3. 所需补丁 31
 4. 最终准备工作 32 4.1. 简介 32

4.2. 在 LFS 文件系统中创建有限的目录结构 32
 4.3. 添加 LFS 用户 32
 4.4. 设置环境 33
 4.5. 关于 SBUs (标准构建单位) 36
 4.6. 关于测试套件 36
 第三部分 构建 LFS 交叉工具链与临时工具 38 重要预备知识 39

i. 引言 39
 ii. 工具链技术说明 39
 iii. 通用编译指南 44
 5. 编译交叉工具链 46 5.1. 简介 46
 5.2. Binutils-2.44 - 第 1 遍 47

5.3. GCC-14.2.0 - 第 1 遍编译 49	5.4. Linux-6.13.4 API 头文件 52	
5.5. Glibc-2.41		53
5.6. 来自 GCC-14.2.0 的 Libstdc++ 56		
6. 交叉编译临时工具 58		
6.1. 简介 58		
6.2. M4-1.4.19 59		
6.3. Ncurses-6.5 60		
6.4. Bash-5.2.37 62		
6.5. Coreutils-9.6 63		
6.6. Diffutils-3.11 64		
6.7. File-5.46 65		
6.8. Findutils-4.10.0 66		
6.9. Gawk-5.3.1 67		
6.10. Grep-3.11 68		
6.11. Gzip-1.13 69		
6.12. Make-4.4.1 70		
6.13. Patch-2.7.6 71		
6.14. Sed-4.9 72		
6.15. Tar-1.35 73		
6.16. Xz-5.6.4 74		
6.17. Binutils-2.44 - 第 2 阶段 75		
6.18. GCC-14.2.0 - 第 2 阶段 76		
7. 进入 Chroot 环境并构建额外临时工具 78	7.1. 简介 78	
7.2. 更改所有权 78		
7.3. 准备虚拟内核文件系统 78		
7.4. 进入 Chroot 环境 79		
7.5. 创建目录 80		
7.6. 创建基础文件与符号链接 81		
7.7. Gettext-0.24 84		
7.8. Bison-3.8.2 85		
7.9. Perl-5.40.1 86		
7.10. Python-3.13.2 87		
7.11. Texinfo-7.2 88		
7.12. Util-linux-2.40.4 89		
7.13. 清理并保存临时系统 90		
IV. 构建 LFS 系统 92	8. 安装基本系统软件 93	8.1. 简介 93
8.2. 软件包管理 94		
8.3. 手册页-6.12 98		
8.4. Iana-Etc-20250123 99	8.5. Glibc-2.41	
8.6. Zlib-1.3.1 108		100
8.7. Bzip2-1.0.8 109		
8.8. Xz-5.6.4 111		

- 8.9. Lz4-1.10.0 113
- 8.10. Zstd-1.5.7 114
- 8.11. File-5.46 115
- 8.12. Readline-8.2.13 116
- 8.13. M4-1.4.19 118
- 8.14. Bc-7.0.3 119
- 8.15. Flex-2.6.4 120
- 8.16. Tcl-8.6.16 121
- 8.17. Expect-5.45.4 第 123 页
- 8.18. DejaGNU-1.6.3 第 125 页
- 8.19. Pkgconf-2.3.0 第 126 页
- 8.20. Binutils-2.44 第 127 页
- 8.21. GMP-6.3.0 130
- 8.22. MPFR-4.2.1 132
- 8.23. MPC-1.3.1 133
- 8.24. Attr-2.5.2 134
- 8.25. Acl-2.3.2 第 135 页
- 8.26. Libcap-2.73 第 136 页
- 8.27. Libxcrypt-4.4.38 第 137 页
- 8.28. Shadow-4.17.3 第 139 页
- 8.29. GCC-14.2.0 第 143 页
- 8.30. Ncurses-6.5 第 149 页
- 8.31. Sed-4.9 第 152 页
- 8.32. Psmisc-23.7 第 153 页
- 8.33. Gettext-0.24 154
- 8.34. Bison-3.8.2 156
- 8.35. Grep-3.11 157
- 8.36. Bash-5.2.37 158
- 8.37. Libtool-2.5.4 160
- 8.38. GDBM-1.24 161
- 8.39. Gperf-3.1 162
- 8.40. Expat-2.6.4 163
- 8.41. Inetutils-2.6 164
- 8.42. Less-668 166
- 8.43. Perl-5.40.1 167
- 8.44. XML::Parser-2.47 170
- 8.45. Intltool-0.51.0 171
- 8.46. Autoconf-2.72 172
- 8.47. Automake-1.17 173
- 8.48. OpenSSL-3.4.1 174
- 8.49. Elfutils-0.192 中的 Libelf 176
- 8.50. Libffi-3.4.7 177
- 8.51. Python-3.13.2 178
- 8.52. Flit-Core-3.11.0 181
- 8.53. Wheel-0.45.1 182
- 8.54. Setuptools-75.8.1 183
- 8.55. Ninja-1.12.1 184

- 8.56. Meson-1.7.0 185
 - 8.57. Kmod-34 186
 - 8.58. Coreutils-9.6 187
 - 8.59. Check-0.15.2 192
 - 8.60. Diffutils-3.11 第 193 页
 - 8.61. Gawk-5.3.1 第 194 页
 - 8.62. Findutils-4.10.0 第 195 页
 - 8.63. Groff-1.23.0 第 196 页
 - 8.64. GRUB-2.12 第 199 页
 - 8.65. Gzip-1.13 第 201 页
 - 8.66. IPRoute2-6.13.0 第 202 页
 - 8.67. Kbd-2.7.1 第 204 页
 - 8.68. Libpipeline-1.5.8 206
 - 8.69. Make-4.4.1 207
 - 8.70. Patch-2.7.6 208
 - 8.71. Tar-1.35 209
 - 8.72. Texinfo-7.2 210
 - 8.73. Vim-9.1.1166 212
 - 8.74. MarkupSafe-3.0.2 215
 - 8.75. Jinja2-3.1.5 216
 - 8.76. Systemd-257.3 中的 Udev 217
 - 8.77. Man-DB-2.13.0 220
 - 8.78. Procs-ng-4.0.5 223
 - 8.79. Util-linux-2.40.4 225
 - 8.80. E2fsprogs-1.47.2 第 230 页
 - 8.81. Sysklogd-2.7.0 第 233 页
 - 8.82. SysVinit-3.14 第 234 页
 - 8.83. 关于调试符号 第 235 页
 - 8.84. 清理调试符号 235
 - 8.85. 清理工作 237
9. 系统配置 238 9.1. 简介 238

- 9.2. LFS-Bootscripts-20240825 239
- 9.3. 设备与模块处理概述 241
- 9.4. 设备管理 244
- 9.5. 通用网络配置 247
- 9.6. System V 启动脚本使用与配置 249
- 9.7. 配置系统区域设置 257
- 9.8. 创建/etc/inputrc 文件 259
- 9.9. 创建/etc/shells 文件 260

10. 使 LFS 系统可启动 262 10.1. 简介 262

- 10.2. 创建 /etc/fstab 文件 262
- 10.3. Linux-6.13.4 264
- 10.4. 使用 GRUB 设置启动流程 270

11. 尾声 273 11.1. 尾声 273 vi

11.2. 统计用户数量	273
11.3. 重启系统	273
11.4. 附加资源	274
11.5. LFS 完成后的后续步骤	275

五、附录 278 A. 缩略语与术语表 279

B. 致谢	282
C. 依赖关系	285
D. 启动与系统配置脚本 version-20240825	299
D.1. /etc/rc.d/init.d/rc	299
D.2. /lib/lsb/init-functions	302
D.3. /etc/rc.d/init.d/mountvirtfs	316
D.4. /etc/rc.d/init.d/modules	318
D.5. /etc/rc.d/init.d/udev	319
D.6. /etc/rc.d/init.d/swap	321
D.7. /etc/rc.d/init.d/setclock	322
D.8. /etc/rc.d/init.d/checkfs	323
D.9. /etc/rc.d/init.d/mountfs	325
D.10. /etc/rc.d/init.d/udev_retry	326
D.11. /etc/rc.d/init.d/cleanfs	328
D.12. /etc/rc.d/init.d/console	330
D.13. /etc/rc.d/init.d/localnet	331
D.14. /etc/rc.d/init.d/sysctl	333
D.15. /etc/rc.d/init.d/sysklogd	334
D.16. /etc/rc.d/init.d/network	335
D.17. /etc/rc.d/init.d/sendsignals	336
D.18. /etc/rc.d/init.d/reboot	337
D.19. /etc/rc.d/init.d/halt	338
D.20. /etc/rc.d/init.d/template	339
D.21. /etc/sysconfig/modules 模块配置文件	340
D.22. /etc/sysconfig/createfiles 创建文件配置	340
D.23. /etc/sysconfig/udev-retry udev 重试配置	341
D.24. /sbin/ifup 网络接口启用脚本	341
D.25. /sbin/ifdown	344
D.26. /lib/services/ipv4-static	346
D.27. /lib/services/ipv4-static-route	347
E. Udev 配置规则	349
E.1. 55-lfs.rules	349
F. LFS 许可证	350
F.1. 知识共享许可协议	350

F.2. MIT 许可证	354
--------------	-----

前言

序言

我学习和深入理解 Linux 的旅程始于 1998 年。那时我刚安装了第一个 Linux 发行版，很快就被 Linux 背后的整体理念和哲学深深吸引。

完成同一项任务总有多 种方法。Linux 发行版亦是如此。多年来出现过无数发行版，有些至今犹存，有些已改头换面，还有些则永远留存在我们的记忆中。它们各具特色，以满足不同用户群体的需求。面对如此多实现相同终极目标的不同途径，我开始意识到自己不必再受限于任何单一实现方式。在接触 Linux 之前，我们只能忍受其他操作系统的种种缺陷——你别无选择。无论喜欢与否，现实就是如此。而 Linux 带来了选择的可能性。如果你对某些设计不满意，你完全有自由——甚至被鼓励——去改变它。

我尝试过多种发行版，却始终无法选定其中任何一个。这些系统本身都很出色，问题已不再关乎对错，而纯粹取决于个人偏好。面对如此丰富的选择，显然没有哪个现成系统能完全符合我的需求。于是我决定打造一个完全契合个人喜好的 Linux 系统。

为了真正让它成为我的专属系统，我决心所有组件都从源代码编译，而非使用预编译的二进制包。这个“完美”的 Linux 系统将兼具备各发行版之长而无其短。最初这个想法令人望而生畏，但我始终坚信这样的系统能够构建成功。

在解决了循环依赖和编译错误等问题后，我终于完成了这个定制版 Linux 系统。它与其他任何 Linux 系统一样功能完备、稳定可用，但这是我的独创成果。亲手组装出这样的系统让我倍感满足。唯一能超越这种成就感的，或许就是亲自编写所有软件——而这已是次优的完美方案。

当我与 Linux 社区的其他成员分享我的目标和经验时，这些理念显然引起了持续的关注。很快我们就清楚地认识到，这种定制构建的 Linux 系统不仅能满足用户的特定需求，还为程序员和系统管理员提供了提升（现有）Linux 技能的绝佳学习机会。正是基于这种广泛兴趣，“Linux From Scratch”项目应运而生。

这本《Linux From Scratch》手册是该项目的核心指南。它为你提供了设计和构建自己系统所需的背景知识和操作指导。虽然本书提供的模板能确保构建出正常运行的系统，但你完全可以自由调整操作步骤——这本身就是本项目的重要特色之一。

你始终掌握着主动权；我们只是在你启程时提供必要的帮助。

衷心希望你在构建自己的 Linux From Scratch 系统时收获快乐，并享受拥有完全属于自己系统的诸多优势。

--
杰勒德·比克曼斯
gerard@linuxfromscratch.o
rg

目标读者

您可能出于多种原因想要阅读本书。许多人常提出的一个疑问是：“既然可以直接下载安装现成的 Linux 系统，为什么还要费时费力从头开始手动构建呢？”

这个项目存在的一个重要原因，是帮助你从内部了解 Linux 系统的工作原理。构建 LFS 系统有助于展示 Linux 的运行机制，以及各个组件如何协同工作和相互依赖。这种学习体验能提供的最佳收获之一，就是能够根据自身独特需求定制 Linux 系统。

LFS 的另一个关键优势是让你完全掌控系统，而无需依赖他人实现的 Linux 发行版。使用 LFS 时，你掌握着绝对主导权。系统的每个细节都由你决定。

LFS 能让你创建极其精简的 Linux 系统。使用其他发行版时，你常常被迫安装大量既不会使用也不理解的程序，这些程序只会浪费资源。或许你会辩称，在当今天大容量硬盘和多核 CPU 的时代，资源浪费已无需考虑。但某些情况下，系统体积仍然会成为制约因素——比如可启动光盘、U 盘和嵌入式系统领域，这正是 LFS 能大显身手的地方。

定制构建 Linux 系统的另一大优势在于安全性。通过从源代码编译整个系统，您能够全面审查所有组件并应用所需的安全补丁。无需等待他人发布修复安全漏洞的二进制包——除非您亲自检查补丁并实施，否则无法确保新二进制包是否正确构建并有效解决问题。

《Linux 从零开始》的目标是构建一个完整可用的基础级系统。即便您不打算从头构建自己的 Linux 系统，本书提供的信息仍将使您获益良多。

自主构建 LFS 系统的益处不胜枚举，难以在此尽述。归根结底，教育价值是最重要的原因。随着 LFS 实践的深入，您将切身感受到信息与知识带来的强大力量。

LFS 目标架构

LFS 的主要目标架构是 AMD/Intel 的 x86（32 位）和 x86_64（64 位）处理器。另一方面，本书的指令经过适当修改后，也已知可在 Power PC 和 ARM 处理器上成功运行。要构建适用于这些替代处理器的系统，除下一页列出的要求外，主要前提条件是需要一个现有 Linux 系统作为基础——例如针对该架构的早期 LFS 安装、Ubuntu、Red Hat/Fedora、SuSE 或其他发行版。（需要注意的是，在 64 位 AMD/Intel 计算机上可以安装并使用 32 位发行版作为宿主系统。）

与 32 位系统相比，在 64 位系统上构建的性能提升微乎其微。例如，在基于 Core i7-4790 处理器（使用 4 个核心）的测试系统中构建 LFS-9.1 时，测得以下数据：

架构类型	构建耗时	构建体积
32 位	239.9 分钟	3.6GB
64 位	233.2 分钟	4.4GB

如你所见，在相同硬件条件下，64 位构建仅比 32 位构建快 3%（但体积大 22%）。若计划将 LFS 用作 LAMP 服务器或防火墙，32 位 CPU 可能已足够。但值得注意的是，当前 BLFS 中的部分软件包在编译或运行时需要超过 4GB 内存；若计划将 LFS 作为桌面系统使用，LFS 作者推荐构建 64 位系统。

LFS 默认生成的 64 位构建是“纯”64 位系统，即仅支持 64 位可执行文件。构建“多库”系统需要将许多应用程序分别编译两次——一次针对 32 位系统，一次针对 64 位系统。由于这会干扰提供基础 Linux 系统最小化教学的核心目标，LFS 并未直接支持该功能。部分 LFS/BLFS 编辑者维护了 LFS 的多库分支版本，可通过 <https://www.linuxfromscratch.org/~thomas/multilib/index.html> 访问。但这属于进阶主题。

前提条件

构建 LFS 系统并非易事。它要求使用者具备一定的 Unix 系统管理基础知识，以便解决问题并正确执行列出的命令。至少需要掌握以下基本技能：能够使用命令行（shell）复制或移动文件及目录、查看目录和文件内容、切换当前工作目录。同时还应了解如何安装和使用 Linux 软件。

由于 LFS 手册默认读者已掌握这些基础技能，因此各类 LFS 支持论坛通常不会在这些方面提供过多帮助。您会发现涉及这些基础知识的提问很可能得不到回复（或仅会收到建议查阅 LFS 必读预备资料的指引）。

在构建 LFS 系统之前，我们强烈建议您阅读以下文章：

- 软件构建指南 <https://tldp.org/HOWTO/Software-Building-HOWTO.html>

这是一份在 Linux 系统下构建和安装“通用”Unix 软件包的完整指南。尽管编写时间较早，但仍对软件构建与安装的基本技术提供了很好的总结。

- 源代码安装新手指南 <https://moi.vonos.net/linux/beginners-installing-from-source/>

本指南很好地总结了从源代码构建软件所需的基本技能与技术。

LFS 与标准

LFS 的结构尽可能遵循 Linux 标准。主要标准包括：

- POSIX.1-2008 标准
- 文件系统层次结构标准(FHS) 3.0 版
- Linux 标准基础 (LSB) 5.0 版 (2015 年)

LSB 包含四个独立规范：核心 (Core)、桌面 (Desktop)、语言 (Languages) 和成像 (Imaging)。其中核心与桌面规范的部分内容与特定架构相关。另有两个试验性规范：Gtk3 和图形 (Graphics)。LFS 力求符合前文讨论的 IA32 (32 位 x86) 或 AMD64 (x86_64) 架构的 LSB 规范要求。

注意

许多人对这些规范要求持有异议。LSB 的主要目的是确保专有软件能在合规系统上安装运行。由于 LFS 采用源码构建方式，用户可完全自主决定需要安装的软件包，您可以选择不安装 LSB 指定的某些软件包。

虽然可以从零开始构建一个能通过 LSB 认证测试的完整系统，但这需要许多超出 LFS 书籍范围的额外软件包。其中部分额外软件包的安装指南可在 BLFS 中找到。

满足 LSB 要求所需的 LFS 提供软件包

LSB 核心组件：

Bash、Bc、Binutils、Coreutils、Diffutils、File、Findutils、Gawk、GCC、Gettext、Glibc、Grep、Gzip、M4、Man-DB、Procps、Psmisc、Sed、Shadow、

LSB 桌面环境: 无
 LSB 支持语言: Perl
 LSB 图像处理: 无
 LSB Gt3 及图形组件 (试用阶段): 无

满足 LSB 规范所需的 BLFS 提供软件包

LSB 核心组件:	定时任务工具 At (含 Batch 功能)、BLFS 版 Bash 启动文件、归档工具 Cpio、行编辑器 Ed、定时任务工具 Fcrontab、LSB 专用工具集、Netscape 便携运行时 NSPR、网络安全服务
LSB 桌面环境:	Alsa、ATK、Cairo、桌面文件工具、Freetype、字体配置、Gdk-pixbuf、Glib2、GLU、图标命名工具、Libjpeg-turbo、Libxml2、Mesa、Pango、Xdg-utils、Xorg
LSB 语言支持: Libxml2、Libxslt	
LSB 图像处理: CUPS、Cups-filters、Ghostscript、SANE	
LSB Gt3 与 LSB 图形组件 (试用版):	GTK+3

满足 LSB 规范要求但 LFS 或 BLFS 未提供或选择性提供的组件

LSB 核心组件:	install_initd、libcrypt.so.1 (可通过 LFS Libcrypt 软件包的可选指令提供)、libncurses.so.5 (可通过 LFS Ncurses 软件包的可选指令提供) libncursesw.so.5 (但 libncursesw.so.6 由 LFS Ncurses 软件包提供)
LSB 桌面环境:	libgdk-x11-2.0.so (但 libgdk-3.so 由 BLFS GTK+3 软件包提供), libgtk-x11-2.0.so (但 libgtk-3.so 和 libgtk-4.so 由 BLFS GTK+3 及 GTK-4 软件包提供), libpng12.so (但 libpng16.so 由 BLFS Libpng 软件包提供), libQt*.so.4 (但 libQt6*.so.6 由 BLFS Qt6 软件包提供), libtiff.so.4 (但 libtiff.so.6 由 BLFS Libtiff 软件包提供)
LSB 语言支持:	/usr/bin/python (LSB 要求 Python2, 但 LFS 和 BLFS 仅提供 Python3)
LSB 图像处理: 无 LSB Gt3 与 LSB 图形组件 (试用版):	libpng15.so (但 BLFS 的 Libpng 软件包提供的是 libpng16.so)

本书选用软件包的理由说明

LFS 的目标是构建一个完整可用的基础级系统——包含所有能自我复制的必要软件包——并提供一个相对精简的基础平台, 让用户能基于此根据个人需求定制更完整的系统。这并不意味着 LFS 是可能的最小系统。其中包含了一些严格来说非必需但非常重要的软件包。下方列表详细说明了书中每个软件包的入选理由。

该软件包包含用于管理访问控制列表(ACL)的工具，这些列表用于为文件和目录定义细粒度的自主访问权限。

- 属性管理工具

该软件包包含用于管理文件系统对象扩展属性的程序。

- 自动配置工具

该软件包提供用于生成能自动配置开发者模板源代码的 shell 脚本程序。当构建流程更新后，通常需要用它来重新构建软件包。

- Automake

该软件包包含从模板生成 Makefile 的程序。当构建流程更新后，通常需要用它来重新构建软件包。

- Bash

该软件包满足 LSB 核心要求，为系统提供 Bourne Shell 接口。因其广泛使用和强大功能，它被选为替代其他 Shell 软件包的方案。

- Bc

该软件包提供任意精度数值处理语言，是构建 Linux 内核的必要组件。

- Binutils

该软件包提供了链接器、汇编器以及其他处理目标文件的工具。构建 LFS 系统中大多数软件包时都需要该软件包内的程序。

- Bison

此软件包包含构建多个 LFS 程序所需的 yacc (Yet Another Compiler Compiler) 的 GNU 版本。

- Bzip2

该软件包包含用于压缩和解压缩文件的程序，解压许多 LFS 软件包时需要用到它。

- 检查

该软件包为其他程序提供测试框架。

- 核心工具集

该软件包包含一系列查看和操作文件及目录的基础程序。这些程序是命令行文件管理所必需的，也是 LFS 中每个软件包安装过程中不可或缺的工具。

- DejaGNU

该软件包提供了测试其他程序的框架工具。

- Diffutils

该软件包包含用于显示文件或目录差异的程序。这些程序可用于创建补丁，也被许多软件包的构建过程所使用。

- E2fsprogs

该软件包提供了处理 ext2、ext3 和 ext4 文件系统的实用工具。这些是 Linux 支持的最常见且经过充分测试的文件系统。

- Expat

该软件包提供了一个相对较小的 XML 解析库。它是 Perl 模块 XML::Parser 所需的组件。

- Expect

该软件包包含一个用于与其他交互式程序进行脚本化对话的程序，常用于测试其他软件包。

- 文件

该软件包包含一个用于判断给定文件类型的实用工具，部分软件包在构建脚本中需要用到它。

- 查找工具

该软件包提供在文件系统中查找文件的程序，被众多软件包的构建脚本所使用。

- Flex

该软件包包含用于生成文本模式识别程序的工具，是 lex（词法分析器）程序的 GNU 版本。构建多个 LFS 软件包时需要此工具。

- Gawk

该软件包提供用于操作文本文件的程序。它是 GNU 版本的 awk (Aho-Weinberg-Kernighan)。许多其他软件包的构建脚本都会使用它。

- GCC

这是 GNU 编译器集合。它包含 C 和 C++ 编译器，以及 LFS 未构建的其他几种编译器。

- GDBM

该软件包包含 GNU 数据库管理器库，被另一个 LFS 软件包 Man-DB 所使用。

- Gettext

该软件包为众多软件包提供国际化和本地化的工具与库支持。

- Glibc

该软件包包含主要的 C 语言库。没有它，Linux 程序将无法运行。

- GMP

该软件包提供数学函数库，支持任意精度算术运算。构建 GCC 时需要此组件。

- Gperf

该软件包生成的程序能够根据一组键值生成完美哈希函数，是 Udev 所需的组件。

- Grep

该软件包包含用于文件搜索的程序，多数软件包的构建脚本都会使用这些程序。

- Groff

该软件包提供用于处理和格式化文本的程序，其重要功能之一是格式化手册页。

- GRUB

这是 GRand Unified Bootloader（大一统引导加载程序），它是现有多种引导加载程序中灵活性最高的方案。

- Gzip

该软件包包含用于压缩和解压缩文件的程序。在 LFS 中解压许多软件包时需要用到它。

- Iana-etc

该软件包提供网络服务和协议数据。启用完整网络功能时需要此组件。

- Inetutils

该软件包提供基础网络管理程序。

- Intltool

该软件包提供从源文件中提取可翻译字符串的工具。

- IProute2

该软件包包含基础及高级 IPv4 和 IPv6 网络工具程序。因其支持 IPv6 的特性，我们选择它而非其他常见网络工具包（如 net-tools）。

- Kbd

该软件包生成键位映射表文件、为非美式键盘提供键盘实用工具，以及多种控制台字体。

- Kmod

该软件包提供管理 Linux 内核模块所需的程序。

- Less

该软件包包含一个出色的文本文件查看器，可在浏览文件时上下滚动。许多软件包使用它来分页输出内容。

- Libcap

该软件包实现了 Linux 内核中 POSIX 1003.1e 能力特性的用户空间接口。

- Libelf

elfutils 项目提供了处理 ELF 文件和 DWARF 数据的库与工具。虽然该软件包中的多数工具在其他软件包中也可获取，但构建 Linux 内核时若采用默认（且最高效）配置则必须使用此库。

- Libffi

该软件包实现了可移植的高级编程接口，支持多种调用约定。某些程序在编译时可能无法确定需要传递给函数的参数。例如，解释器可能在运行时才获知调用特定函数所需的参数数量和类型。此类程序可利用 Libffi 在解释程序与编译代码之间建立桥梁。

- Libpipeline

Libpipeline 软件包提供了一个库，用于以灵活便捷的方式操作子进程管道。该库是 Man-DB 软件包的必备依赖项。

- Libtool

该软件包包含 GNU 通用库支持脚本，它将使用共享库的复杂性封装成统一且可移植的接口。该工具是其他 LFS 软件包中测试套件的必要组件。

- Libxcrypt

该软件包提供了多种程序（尤其是 Shadow）所需的 libcrypt 库，用于密码哈希处理。它取代了 Glibc 中过时的 libcrypt 实现。

- Linux 内核

该软件包是操作系统本身，即 GNU/Linux 环境中的 Linux 核心组件。

- M4

该软件包提供了一个通用的文本宏处理器，可作为其他程序的构建工具使用。

- Make

该软件包包含一个用于指导软件包构建的程序。LFS 中几乎每个软件包都需要它。

- Man-DB

该软件包包含用于查找和查看手册页的程序。选择它而非 man 软件包的原因是它具有更出色的国际化支持能力。它提供了 man 程序。

- Man-pages

该软件包提供了基础 Linux 手册页的实际内容。

- Meson

该软件包提供了一个用于自动化构建软件的工具。Meson 的主要目标是尽量减少开发人员在配置构建系统上花费的时间。构建 Systemd 以及许多 BLFS 软件包时都需要它。

- MPC

该软件包提供复数运算功能。GCC 编译器需要依赖此软件包。

- MPFR

该软件包包含用于多精度算术运算的函数，是 GCC 的必需组件。

- Ninja

该软件包提供了一套专注于速度的小型构建系统，其设计目标是通过高级构建系统生成输入文件，并以最快速度执行构建任务。该软件包是 Meson 的必需组件。

- Ncurses

该软件包包含用于独立于终端的字符屏幕处理的库。它常用于为菜单系统提供光标控制功能，是 LFS 中多个软件包所需的依赖项。

- Openssl

该软件包提供与密码学相关的管理工具和库，为包括 Linux 内核在内的其他软件包提供加密功能支持。

- 补丁

该软件包包含一个通过应用通常由 diff 程序创建的补丁文件来修改或创建文件的程序。多个 LFS 软件包的构建过程需要它。

- Perl

该软件包是运行时语言 PERL 的解释器。多个 LFS 软件包的安装和测试套件需要它。

- Pkgconf

该软件包包含一个帮助配置开发库编译器与链接器标志的程序。该程序可作为 pkg-config 的替代品使用，许多软件包的构建系统都需要此功能。相比原版 Pkg-config 软件包，它的维护更活跃且运行速度略快。

- Procs-NG

此软件包包含进程监控程序。这些程序对系统管理非常有用，同时也被 LFS 启动脚本所使用。

- Psmisc

该软件包提供用于显示运行进程信息的程序，这些程序对系统管理非常有用。

- Python 3

该软件包提供一种解释型编程语言，其设计理念强调代码可读性。

- Readline

该软件包提供了一套实现命令行编辑和历史功能的库，被 Bash 所使用。

- Sed

该软件包支持无需打开文本编辑器即可进行文本编辑，同时也是许多 LFS 软件包配置脚本的必需组件。

- 影子密码

该软件包包含用于安全处理密码的程序。

- 系统日志守护进程

该软件包提供记录系统消息的程序，例如当异常事件发生时由内核或守护进程发出的消息。

- SysVinit

该软件包提供 init 程序，这是运行中的 Linux 系统上所有其他进程的父进程。

- Udev

该软件包是设备管理器。当设备被添加至系统或从系统中移除时，它能动态控制/dev 目录下设备节点的所有权、权限、名称及符号链接。

- Tar

该软件包提供几乎所有 LFS 所用软件包的归档与解压功能。

- Tcl

该软件包包含用于多个测试套件的工具命令语言(Tcl)。

- Texinfo

该软件包提供用于读取、写入和转换 info 页面的程序，被众多 LFS 软件包的安装过程所使用。

- Util-linux

该软件包包含各类实用工具程序，其中包括用于处理文件系统、控制台、分区和消息的实用工具。

- Vim

该软件包提供了一个编辑器。选择它的原因在于其与经典 vi 编辑器的兼容性以及强大的功能集。对许多用户而言，编辑器是非常个人化的选择。如有需要，您可以选择替换为其他任意编辑器。

- Wheel

该软件包提供了一个 Python 模块，是 Python wheel 打包标准的参考实现。

- XML::解析器

该软件包是一个与 Expat 交互的 Perl 模块。

- XZ 工具集

该软件包包含用于压缩和解压缩文件的程序。它提供了当前普遍可用的最高压缩率，对于解压 XZ 或 LZMA 格式的软件包非常有用。

- Zlib 压缩库

该软件包包含某些程序使用的压缩和解压缩例程。

- Zstd 压缩工具

该软件包提供了一些程序使用的压缩与解压例程，能够实现高压缩比，并提供广泛的压缩速度与压缩率权衡方案。

排版规范

为便于理解，本书通篇采用若干排版约定。本节列举了《Linux From Scratch》中常见的排版格式示例。

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

此类文本格式要求严格按原样输入，除非上下文另有说明。在解释性章节中，这种格式也用于标识所引用的具体命令。

某些情况下，逻辑行会通过行尾的反斜杠延伸至两个或多个物理行。

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

请注意反斜杠后必须直接换行。若出现空格或制表符等其他空白字符，将导致错误结果。

```
install-info: 未知选项 '--dir-file=/mnt/lfs/usr/info/dir'
```

这种固定宽度的文本格式用于显示屏幕输出，通常是执行命令后的结果。该格式也用于显示文件名，例如/etc/ld.so.conf。

注意

请将浏览器配置为使用良好的等宽字体（字号 9pt）显示固定宽度文本，以便清晰区分 Il1 或 O0 等字符的形态差异。

强调

本书中此类文本形式用于多种用途，其主要目的是强调重点内容或条目。

<https://www.linuxfromscratch.org/>

该格式用于 LFS 社区内部及外部页面的超链接，包含操作指南、下载地址及网站链接。

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

EOF

该格式用于创建配置文件时。第一条命令指示系统创建文件\$LFS/etc/group，内容将来自随后输入的行，直到遇到文件结束符(EOF)为止。因此，通常直接按原文输入整个部分即可。

<替换文本>

此格式用于封装无需按原样输入或进行复制粘贴操作的文本。

[可选文本]

此格式用于封装可选文本。

passwd(5)

这种格式用于引用特定的手册页（man page）。括号内的数字表示手册中的特定章节。例如，passwd 有两个手册页。根据 LFS 安装说明，这两个手册页

将分别位于/usr/share/man/man1/passwd.1 和/usr/share/man/man5/passwd.5。当书中使用 passwd(5)时

特指/usr/share/man/man5/passwd.5。直接运行 man passwd 会显示第一个匹配"passwd"的手册页，即/usr/share/man/man1/passwd.1。因此要查看指定章节，需要运行 man 5 passwd。注意大多数手册页在不同章节中不会有重复名称，通常直接 man <程序名>即可。在 LFS 手册中，这些手册页引用也是超链接，点击可直接打开 Arch Linux 手册页渲染的 HTML 版本。

结构

本书分为以下几个部分。

第一部分 - 简介

第一部分阐述了进行 LFS 安装时需要注意的几个重要事项。本部分还提供了关于本书的元信息。

第二部分 - 构建前的准备工作

第二部分描述如何为构建过程做准备——包括创建分区、下载软件包以及编译临时工具。

第三部分 - 构建 LFS 交叉工具链与临时工具

第三部分提供构建最终 LFS 系统所需工具的具体指导。

第四部分 - 构建 LFS 系统

第四部分将引导读者逐步构建 LFS 系统——逐个编译安装所有软件包、设置启动脚本以及安装内核。最终构建完成的 Linux 系统将成为基础平台，可根据需要在此基础上扩展安装其他软件。本书末尾附有便捷的参考列表，详细记录了所有已安装的程序、库文件及重要文件。

第五部分 - 附录

第五部分提供与本书相关的参考信息，包括术语缩写表、致谢声明、软件包依赖关系、LFS 启动脚本清单、本书发行许可协议，以及完整的软件包/程序/库文件/脚本索引。

勘误与安全通告

用于构建 LFS 系统的软件会持续更新和优化。在 LFS 手册发布后，可能会出现安全警告和错误修复。在开始构建前，请访问 <https://www.linuxfromscratch.org/lfs/errata/12.3/> 以检查本版 LFS 中的软件包版本或操作说明是否需要修改——无论是修复安全漏洞还是修正其他错误。在构建 LFS 系统时，您应当记录所有显示的变更，并将其应用到手册的相关章节中。

此外，Linux From Scratch 编辑团队会维护手册发布后发现的安全漏洞列表。在开始构建前，请访问 <https://www.linuxfromscratch.org/lfs/advisories/> 查阅该列表。构建 LFS 系统时，您应当根据通告建议对手册相关章节进行相应修改。如果您计划将 LFS 系统用作实际桌面或服务器系统，即使系统已完全构建完成，也应持续关注安全通告并修复所有安全漏洞。

第一部分 简介

第 1 章 简介

1.1 如何构建 LFS 系统

LFS 系统将通过已安装的 Linux 发行版（如 Debian、OpenMandriva、Fedora 或 openSUSE）进行构建。这个现有的 Linux 系统（称为宿主系统）将作为起点，提供必要的程序（包括编译器、链接器和 shell）来构建新系统。在发行版安装过程中选择“开发”选项以包含这些工具。

注意

安装 Linux 发行版有多种方式，但默认设置通常不适合构建 LFS 系统。关于如何设置商业发行版的建议，请参阅：<https://www.linuxfromscratch.org/hints/downloads/files/partitioning-for-lfs.txt>

除了在您的机器上安装一个独立的发行版之外，您也可以考虑使用商业发行版的 LiveCD。

本书第 2 章将指导您如何创建一个新的 Linux 原生分区和文件系统，用于编译和安装新的 LFS 系统。第 3 章详细列出了构建 LFS 系统所需的软件包和补丁，并说明如何将它们存储在新文件系统中。第 4 章讨论了如何配置合适的工作环境。请仔细阅读第 4 章，因为它阐述了在开始第 5 章及后续章节之前您需要了解的若干重要事项。

第 5 章将介绍如何使用交叉编译技术安装初始工具链（包括 binutils、gcc 和 glibc），从而将新工具与宿主系统隔离开来。

第 6 章将展示如何利用刚刚构建的交叉工具链来编译基础工具程序。

第七章将进入“chroot”环境，我们会使用新构建的工具来创建 LFS 系统所需的所有剩余工具。

这种将新系统与宿主发行版隔离的做法看似有些过度。关于为何要这样做的完整技术说明，请参阅工具链技术注释部分。

第八章将构建完整的 LFS 系统。chroot 环境提供的另一个优势是，在构建 LFS 的同时，您仍可以继续使用宿主系统。在等待软件包编译完成时，您可以照常使用计算机。

为了完成安装，第九章将进行基本系统配置，第十章将创建内核和引导加载程序。第十一章包含如何继续深入 LFS 体验的信息。完成本章所述步骤后，计算机就可以启动进入全新的 LFS 系统了。

简而言之，这就是整个构建流程。后续章节将详细阐述每个步骤。那些现在看似复杂的环节都会得到清晰解释，当您开启 LFS 之旅时，一切都会变得井然有序。

1.2. 新版更新内容

以下是自 LFS 上一版本发布以来更新的软件包列表。

升级至：

- Bash-5.2.37
- Bc-7.0.3
- Binutils-2.44
- Coreutils-9.6
- Diffutils-3.11
- E2fsprogs-1.47.2
- Expat-2.6.4
- File-5.46
- Flit-core-3.11.0
- Gawk-5.3.1
- Gettext-0.24
- Glibc-2.41
- Iana-Etc-20250123
- Inetutils-2.6
- IPRoute2-6.13.0
- Jinja2-3.1.5
- Kbd-2.7.1
- Kmod-34
- Less-668
- Libcap-2.73
- Elfutils-0.192 中的 Libelf 库
- Libffi-3.4.7
- Libpipeline-1.5.8
- Libtool-2.5.4
- Libxcrypt-4.4.38
- Linux-6.13.4
- Man-DB-2.13.0
- Man-pages-6.12
- MarkupSafe-3.0.2
- Meson-1.7.0
- OpenSSL-3.4.1
- Perl-5.40.1
- Procps-ng-4.0.5
- Python-3.13.2
- Setuptools-75.8.1
- Shadow-4.17.3
- Sysklogd-2.7.0

- Systemd-257.3
- SysVinit-3.14
- Tcl-8.6.16
- Texinfo-7.2
- 时区数据-2025a
- Systemd-257.3 中的 Udev
- 工具集-linux-2.40.4
- Vim-9.1.1166
- Wheel-0.45.1
- Xz-5.6.4
- Zstd-1.5.7

新增:

.

已移除:

.

1.3. 更新日志 本文档为 2025 年 3 月 5 日发布的《Linux 从零开始》第 12.3 版。若您获取的版本已超过六个月，可能有更新的版本可供使用。请通过 <https://www.linuxfromscratch.org/mirrors.html> 访问镜像站点查询最新版本。

linuxfromscratch.org/mirrors.html 访问镜像站点查询最新版本。

以下列出的是自上一版本发布以来的变更内容。

更新日志条目:

- 2025-03-05
 - [bdubbs] - 发布 LFS-12.3 版本。
- 2025-03-02
 - [bdubbs] - 更新至 vim-9.1.1166 (安全更新)。修复问题 #5666。
- 2025-02-27
 - [bdubbs] - 更新至 zstd-1.5.7。修复问题 #5652。
 - [bdubbs] - 更新至 systemd-257.3。修复问题 #5612。
 - [bdubbs] - 更新至 shadow-4.17.3。修复问题 #5660。
 - [bdubbs] - 更新至 setuptools-75.8.1。修复问题 #5662。
 - [bdubbs] - 更新至 linux-6.13.4。修复问题 #5647。
 - [bdubbs] - 更新至 kmod-34。修复问题 #5657。
 - [bdubbs] - 更新至 inetutils-2.6。修复问题 #5656。
 - [bdubbs] - 更新至 gettext-0.24。修复问题 #5661。
 - [bdubbs] - 更新至 flit_core-3.11.0。修复问题 #5654。

- 2025-02-24
 - [bdubbs] - 更新至 man-pages-6.12。修复问题 #5658。
- 2025-02-19
 - [xry111] - 更新至 vim-9.1.1122 (安全更新)。处理问题 #4500。
 - [xry111] - 更新至 man-pages-6.11。修复问题 #5646。
- 2025-02-13
 - [bdubbs] - 更新至 vim-9.1.1106。处理问题 #4500。
 - [bdubbs] - 更新至 diffutils-3.11。修复问题 #5639。
 - [bdubbs] - 更新至 libffi-3.4.7。修复问题 #5642。
 - [bdubbs] - 更新至 linux-6.13.2。修复问题 #5643。
 - [bdubbs] - 更新至 Python3-3.13.2。修复问题 #5640。
 - [bdubbs] - 更新至 sysvinit-3.14。修复问题 #5641。
- 2025-02-02
 - [bdubbs] - 更新至 vim-9.1.1071。处理问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20250123。处理问题 #5006。
 - [bdubbs] - 更新至 binutils-2.44.0。修复问题 #5634。
 - [bdubbs] - 更新至 coreutils-9.6。修复问题 #5628。
 - [bdubbs] - 更新至 e2fsprogs-1.47.2。修复问题 #5637。
 - [bdubbs] - 更新至 glibc-2.41。修复问题 #5638。
 - [bdubbs] - 更新至 iproute2-6.13.0。修复问题 #5631。
 - [bdubbs] - 更新至 libxcrypt-4.4.38。修复问题 #5626。
 - [bdubbs] - 更新至 linux-6.13.1。修复问题 #5629。
 - [bdubbs] - 更新至 man-pages-6.10。修复问题 #5632。
 - [bdubbs] - 更新至 meson-1.7.0。修复问题 #5636。
 - [bdubbs] - 更新至 perl-5.40.1。修复问题 #5630。
 - [bdubbs] - 更新至 tcl8.6.16。修复问题 #5635。
 - [bdubbs] - 更新至 tzdata2025a。修复问题 #5627。
 - [bdubbs] - 更新至 xz-5.6.4。修复问题 #5633。
- 2025-01-15
 - [bdubbs] - 更新至 vim-9.1.1016。修复问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20250108。修复问题 #5006。
 - [bdubbs] - 更新至 util-linux-2.40.4。修复问题 #5624。
 - [bdubbs] - 更新至 sysvinit-3.13。修复问题 #5621。
 - [bdubbs] - 更新至 sysklogd-2.7.0。修复问题 #5623。
 - [bdubbs] - 更新至 shadow-4.17.2。修复问题 #5625。
 - [bdubbs] - 更新至 setuptools-75.8.0。修复问题 #5622。

- [bdubbs] - 更新至 linux-6.12.9。修复问题 #5620。
- [bdubbs] - 更新至 gettext-0.23.1。修复问题 #5619。
- 2025-01-01
 - [renodr] - 更新至 libxcrypt-4.4.37。修复问题 #5618。
 - [bdubbs] - 更新至 iana-etc-20241220。处理问题 #5006。
 - [bdubbs] - 更新至 texinfo-7.2。修复问题 #5616。
 - [bdubbs] - 更新至 sysvinit-3.12。修复问题 #5615。
 - [bdubbs] - 更新至 shadow-4.17.1。修复问题 #5617。
 - [bdubbs] - 更新至 procps-ng-4.0.5。修复问题 #5611。
 - [bdubbs] - 更新至 meson-1.6.1。修复问题 #5610。
 - [bdubbs] - 更新至 linux-6.12.7。修复问题 #5613。
 - [bdubbs] - 更新至 kbd-2.7.1。修复问题 #5608。
 - [bdubbs] - 更新至 jinja2-3.1.5 (安全更新)。修复问题 #5614。
- 2024-12-15
 - [bdubbs] - 更新至 vim-9.1.0927。修复问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20241206。修复问题 #5006。
 - [bdubbs] - 更新至 systemd-257。修复问题 #5559。
 - [bdubbs] - 更新至 Python-3.13.1 (安全更新)。修复问题 #5605。
 - [bdubbs] - 更新至 libcap-2.73。修复问题 #5604。
 - [bdubbs] - 更新至 linux-6.12.5。修复问题 #5607。
 - [bdubbs] - 更新至 kbd-2.7。修复问题 #5608。
 - [bdubbs] - 更新至 gettext-0.23。修复问题 #5603。
- 2024-12-01
 - [bdubbs] - 更新至 iana-etc-20241122。修复问题 #5006。
 - [bdubbs] - 更新至 file-5.46。修复问题 #5601。
 - [bdubbs] - 更新至 iproute2-6.12.0。修复问题 #5597。
 - [bdubbs] - 更新至 libtool-2.5.4。修复问题 #5598。
 - [bdubbs] - 更新至 linux-6.12.1。修复问题 #5586。
 - [bdubbs] - 更新至 setuptools-75.6.0 (Python 模块)。修复问题 #5599。
 - [bdubbs] - 更新至 wheel-0.45.1 (Python 模块)。修复问题 #5600。
- 2024-11-15
 - [bdubbs] - 更新至 vim-9.1.0866。处理问题 #4500。
 - [bdubbs] - 更新至 iana-etc-20241024。解决 #5006 问题。
 - [bdubbs] - 更新至 wheel-0.45.0 (Python 模块)。修复 #5593 问题。
 - [bdubbs] - 更新至 setuptools-75.5.0 (Python 模块)。修复 #5595 问题。
 - [bdubbs] - 更新至 linux-6.11.8。修复 #5582 问题。

- [bdubbs] - 更新至 libcap-2.72。修复问题 #5594。
- 2024-11-08
 - [bdubbs] - 添加 binutils-2.43.1-upstream_fix-1.patch 补丁。修复问题 #5591。
 - [bdubbs] - 更新至 flit_core-3.10.1。修复问题 #5589。
 - [bdubbs] - 更新至 expat-2.6.4。修复问题 #5590。
- 2024-10-25
 - [bdubbs] - 更新至 linux-6.11.6。修复 #5588。
 - [bdubbs] - 更新至 libcap-2.71。修复 #5584。
 - [bdubbs] - 更新至 setuptools-75.3.0。修复 #5585。
 - [bdubbs] - 更新至 flit_core-3.10.0。修复 #5587。
- 2024-10-25
 - [bdubbs] - 更新至 iana-etc-20241015。处理问题#5006。
 - [bdubbs] - 更新至 vim-9.1.0813。处理问题#4500。
 - [bdubbs] - 更新至 xz-5.6.3。修复问题#5572。
 - [bdubbs] - 更新至 sysvinit-3.11。修复问题#5581。
 - [bdubbs] - 更新至 setuptools-75.2.0。修复问题 #5577。
 - [bdubbs] - 更新至 Python3-3.13.0。修复问题 #5575。
 - [bdubbs] - 更新至 openssl-3.4.0。修复问题 #5582。
 - [bdubbs] - 更新至 meson-1.6.0。修复问题 #5580。
 - [bdubbs] - 更新至 markupsafe-3.0.2。修复问题 #5576。
 - [bdubbs] - 更新至 linux-6.11.5。修复问题 #5574。
 - [bdubbs] - 更新至 less-668。修复问题 #5578。
 - [bdubbs] - 更新至 elfutils-0.192。修复问题 #5579。
- 2024-10-03
 - [bdubbs] - 回退至 tcl8.6.15 版本。
- 2024-10-01
 - [bdubbs] - 更新至 Python3-3.12.7 版本。修复问题 #5571。
 - [bdubbs] - 更新至 tcl9.0.0 版本。修复问题 #5570。
 - [bdubbs] - 更新至 linux-6.11.1 版本。修复问题 #5556。
 - [bdubbs] - 更新至 libtool-2.5.3。修复问题 #5569。
 - [bdubbs] - 更新至 iproute2-6.11.0。修复问题 #5561。
 - [bdubbs] - 更新至 bash-5.2.37。修复问题 #5567。
 - [bdubbs] - 更新至 bc-7.0.3。修复问题 #5568。
- 2024-09-20
 - [bdubbs] - 更新至 vim-9.1.0738 版本。解决 #4500 问题。
 - [bdubbs] - 更新至 texinfo-7.1.1 版本。修复 #5558 问题。

- [bdubbs] - 更新至 tcl8.6.15。修复问题 #5562。
- [bdubbs] - 更新至 sysklogd-2.6.2。修复问题 #5557。
- [bdubbs] - 更新至 setuptools-75.1.0 版本。修复问题#5560。
- [bdubbs] - 更新至 meson-1.5.2 版本。修复问题#5566。
- [bdubbs] - 更新至 iana-etc-20240912。修复问题#5006。
- [bdubbs] - 更新至 gawk-5.3.1。修复问题#5564。
- [bdubbs] - 更新至 bc-7.0.2。修复问题#5563。
- 2024-09-07
 - [bdubbs] - 更新至 tzdata-2024b。修复问题#5554。
 - [bdubbs] - 更新至 systemd-256.5。修复问题 #5551。
 - [bdubbs] - 更新至 setuptools-74.1.2。修复问题 #5546。
 - [bdubbs] - 更新至 python3-3.12.6。修复问题 #5555。
 - [bdubbs] - 更新至 openssl-3.3.2。修复问题 #5552。
 - [bdubbs] - 更新至 man-db-2.13.0。修复问题 #5550。
 - [bdubbs] - 更新至 linux-6.10.8。修复问题 #5545。
 - [bdubbs] - 更新至 libpipeline-1.5.8。修复问题 #5548。
 - [bdubbs] - 更新至 expat-2.6.3。修复问题 #5553。
 - [bdubbs] - 更新至 bc-7.0.1 版本。修复问题 #5547。
- 2024-09-01
 - [bdubbs] - LFS-12.2 版本已发布。

1.4. 资源

1.4.1. 常见问题解答 在构建 LFS 系统过程中，若遇到任何错误、疑问或认为书中存在排版错误，请首先查阅位于 <https://www.linuxfromscratch.org/FAQ.html> 的常见问题解答（FAQ）列表。

org/faq/。

1.4.2. 邮件列表 [linuxfromscratch.org](https://www.linuxfromscratch.org/) 服务器托管了多个用于 LFS 项目开发的邮件列表。这些列表包括主要的开发和支持列表等。如果您在 FAQ 页面找不到问题的答案，下一步可以访问 <https://www.linuxfromscratch.org/search.html> 搜索邮件列表。

有关不同列表的信息、订阅方式、存档位置及其他详情，请访问 <https://www.linuxfromscratch.org/mail.html>。

1.4.3. IRC 聊天 LFS 社区的若干成员通过互联网中继聊天(IRC)提供帮助。在使用此支持渠道前，请确认您的问题未在 LFS 常见问题解答或邮件列表存档中得到解答。您可以在 [irc.libera.chat](irc://irc.libera.chat) 找到 IRC 网络，支持频道名为#lfs-support。

1.4.4. 镜像站点 LFS 项目在全球设有多个镜像站点，以便更便捷地访问网站和下载所需软件包。请访问 LFS 官网 <https://www.linuxfromscratch.org/mirrors.html> 获取最新镜像列表。

1.4.5. 联系方式 请将所有问题与建议发送至 LFS 邮件列表（参见上文）。

1.5. 帮助说明

若您在按照 LFS 指南构建某个软件包时遇到问题，我们强烈建议您先通过 1.4 节“资源”中列出的 LFS 支持渠道进行讨论，而非直接向上游支持渠道提交问题。这种做法通常效率极低，因为上游维护者往往不熟悉 LFS 的构建流程。即便您确实遇到了上游问题，LFS 社区仍能帮助提取上游维护者所需的关键信息，并提交规范的问题报告。

若您必须直接通过上游支持渠道提问，至少应当注意：许多上游项目将支持渠道与缺陷追踪系统分开管理。在这些项目中，将提问作为“缺陷”报告提交会被视为无效，并可能干扰上游开发者的工作。

若在使用本书过程中遇到问题或疑问，请先查阅常见问题解答页面：

<https://www.linuxfromscratch.org/faq/#generalfaq>。该页面通常已涵盖多数常见问题。若您的问题未获解答，请尝试定位问题根源。以下提示文档可提供故障排查指导：

<https://www.linuxfromscratch.org/hints/downloads/files/errors.txt>。

如果在 FAQ 中找不到您遇到的问题，请访问 <https://www.linuxfromscratch.org/search> 搜索邮件列表。

html.

我们还有一个优秀的 LFS 社区，愿意通过邮件列表和 IRC 提供帮助（参见本书 1.4 节“资源”部分）。然而，我们每天都会收到多个支持请求，其中许多问题本可以通过查阅 FAQ 或先搜索邮件列表轻松解决。因此，为了让我们能提供最佳协助，您应该先自行进行一些研究。这样我们才能专注于处理更特殊的 support 需求。如果您的搜索未能找到解决方案，请在求助时包含以下所有相关信息。

1.5.1. 需要说明的事项 除了简要描述遇到的问题外，任何求助请求都应包含以下关键

事物

- 所使用的书籍版本（此处为 12.3）
- 用于构建 LFS 的主机发行版及其版本
- 主机系统需求脚本的输出内容
- 遇到问题的软件包或相关章节
- 确切的错误信息，或对问题的清晰描述

- 记录你是否完全按照本书操作

注意

偏离本书指导并不意味着我们不会提供帮助。毕竟，LFS 的核心在于个人定制。事先说明对既定流程的任何改动，将有助于我们评估和确定问题的潜在原因。

1.5.2. 配置脚本问题 若运行 `configure` 脚本时出现错误，请检查 `config.log` 文件。该文件可能包含配置过程中未显示在屏幕上的错误信息。如需寻求帮助，请附上相关行内容。

1.5.3. 编译问题 屏幕输出和各种文件内容对于诊断编译问题都很有价值。

配置脚本和 `make` 运行的屏幕输出可能有所帮助。不必包含全部输出，但需包含所有相关信息。以下是 `make` 屏幕输出中应包含的信息类型示例。

```
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 signature.o variable.o
vpath.o default.o remote-stub.o version.o getopt.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-recurse] Error 1 make[1]: Leaving
directory `/lfs/tmp/make-3.79.1' make: *** [all-recurse-am] Error 2
```

这种情况下，多数人只需包含底部部分：

```
make [2]: *** [make] 错误 1
```

仅凭这条信息不足以诊断问题，因为它仅表明出现了错误，而未指明具体错误原因。完整的上下文（如上文示例所示）应当被保存，因为它包含了执行的命令以及所有相关的错误信息。

关于如何在互联网上有效求助，有一篇出色的在线文章可供参考：<http://catb.org/~esr/faqs/smarty-questions.html>。

阅读该文档并遵循其中的建议，这将显著提高您获得所需帮助的可能性。

第二部分 构建前的准备工作

第二章 准备宿主系统

2.1. 简介

本章将检查构建 LFS 所需的宿主工具，并在必要时进行安装。随后我们将准备用于存放 LFS 系统的分区，包括创建分区本身、在其上建立文件系统并挂载该分区。

2.2. 宿主系统要求

2.2.1. 硬件要求 LFS 编辑团队建议系统 CPU 至少具备四核处理器，内存至少 8GB。不满足这些要求的旧系统仍可运行，但软件包编译时间将显著长于文档所述时长。

2.2.2. 软件要求 宿主系统应安装以下最低版本要求的软件。这对大多数现代 Linux 发行版不成问题。需注意许多发行版会将软件头文件放在单独的包中，通常命名为`-devel` 或 `-dev`。若发行版提供此类包，请务必安装。

所列软件包的早期版本可能可用，但未经测试验证。

- Bash-3.2 (`/bin/sh` 应是 `bash` 的符号链接或硬链接)
- Binutils-2.13.1 (不建议使用高于 2.44 的版本，因未经测试)
- Bison-2.7 (`/usr/bin/yacc` 应链接至 `bison` 或设置为调用 `bison` 的小脚本)
- Coreutils-8.1
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-4.0.1 (`/usr/bin/awk` 应链接至 `gawk`)
- GCC-5.2 (包含 C++ 编译器 `g++`，不建议使用高于 14.2.0 的版本，因未经测试)。必须同时安装 C 和 C++ 标准库 (含头文件)，以便 C++ 编译器能构建宿主程序
- Grep-2.5.1a
- Gzip-1.3.12
- Linux 内核-5.4

设定该内核版本要求的原因是：我们在第五章和第八章构建 glibc 时指定了该版本，因此不会启用针对旧版内核的兼容方案，这样编译出的 glibc 会略微更快且更小。截至 2024 年 12 月，5.4 版本仍是内核开发者维护的最旧发行版。某些早于 5.4 版本的内核可能仍由第三方团队维护，但它们不被视为官方上游

内核版本；详情请参阅 <https://kernel.org/category/releases.html>。

如果主机内核版本早于 5.4，您需要将其替换为更新的版本。有两种方法可以实现这一点。首先，查看您的 Linux 供应商是否提供 5.4 或更高版本的内核包。如果有，您可以选择安装它。如果供应商未提供合适的内核包，或者您不想安装供应商提供的内核，您可以自行编译内核。编译内核及配置引导加载程序（假设主机使用 GRUB）的相关说明位于第 10 章。

我们要求宿主内核支持 UNIX 98 伪终端(PTY)。所有搭载 Linux 5.4 或更新内核的桌面/服务器发行版都应默认启用该功能。若您正在构建自定义宿主内核，请确保在内核配置中将 `UNIX98_PTYS` 参数设置为 `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

重要提示

请注意，上述符号链接是使用本书所含指令构建 LFS 系统所必需的。指向其他软件（如 dash、mawk 等）的符号链接或许能工作，但未经 LFS 开发团队测试或支持，可能需要偏离指令或对某些软件包应用额外补丁。

要检查宿主系统是否具备所有合适版本及程序编译能力，请运行以下命令：

```
cat > version-check.sh << "EOF"
#!/bin/bash # 列出关键开发工具版本号的脚本

# 若工具安装在其他目录，请在此处调整 PATH 变量 # 并同步修改~lfs/.bashrc 文件（参见 4.4 节）
```

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

bail() { echo "致命错误: $1"; exit 1; } grep --version > /dev/null 2> /dev/null || bail "grep
无法正常工作" sed '' /dev/null || bail "sed 无法正常工作" sort /dev/null || bail "sort 无法正
常工作"
```

版本检查()

```
{ if ! type -p $2 &>/dev/null then

echo "错误: 找不到 $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 "通
过: %-9s %-6s >= $3\n" "$1" "$v"; return 0; else printf "错误: %-9s 版本过旧 (需要$3 或更高版
本)\n" "$1"; return 1; fi }
```

内核版本
{

```
kver=$(uname -r | grep -E -o '[0-9\.\.]+') if printf '%s\n' $1 $kver | sort --version-sort --check &>/dev/null
then printf "OK: Linux 内核版本 $kver >= $1\n"; return 0; else printf "错误: Linux 内核版本 ($kver) 过低 (需要 $1
或更高版本)\n" "$kver"; return 1; fi }
```

```
# 首先检查 Coreutils, 因为--version-sort 需要 Coreutils >= 7.0
版本检查 Coreutils sort 8.1 || 终止 "Coreutils 版本过低, 终止"
版本检查 Bash bash 3.2
版本检查 Binutils ld 2.13.1
版本检查 Bison bison 2.7
版本检查 Diffutils diff 2.8.1
版本检查 Findutils find 4.2.31
版本检查 Gawk gawk 4.0.1
版本检查 GCC gcc 5.2
版本检查 "GCC (C++)" g++ 5.2
版本检查 Grep grep 2.5.1a
版本检查 Gzip gzip 1.3.12
版本检查 M4 m4 1.4.10
版本检查 Make make 4.0
版本检查 Patch patch 2.5.4
版本检查 Perl perl 5.8.8
版本检查 Python python3 3.4
版本检查 Sed sed 4.1.5
版本检查 Tar tar 1.22
版本检查 Texinfo texi2any 5.0
版本检查 Xz xz 5.0.0
内核版本检查 5.4
```

```
if mount | grep -q 'devpts on /dev/pts' && [ -e /dev/ptmx ] then echo "OK: Linux 内核支
持 UNIX 98 PTY"; else echo "错误: Linux 内核不支持 UNIX 98 PTY"; fi
```

```
别名检查() { if $1 --version 2>&1 | grep -qi $2 then printf "通过:
%-4s 是 $2\n" "$1"; else printf "错误: %-4s 不是 $2\n" "$1"; fi }
echo "别名检查:" 别名检查 awk GNU 别名检查 yacc Bison 别名检查 sh
Bash
```

```
echo "编译器检查:" if printf "int main() {}" | g++ -x
c++ then echo "通过: g++ 工作正常"; else echo "错误:
g++ 无法工作"; fi rm -f a.out
```

```
if [ "$(nproc)" = "" ]; then echo "错误: nproc 不可用或其输出为空" else echo "通过: nproc
报告当前可用逻辑核心数为 $(nproc)" fi
```

EOF

bash version-check.sh

2.3. 分阶段构建 LFS

LFS 设计为单次会话中完成构建。这意味着操作指南假设系统在整个过程中不会关机。但这并不要求必须一次性完成所有构建步骤。关键在于：当在不同阶段恢复 LFS 构建时，某些操作必须在重启后重复执行。

2.3.1. 第 1-4 章

这些章节需要在宿主系统上执行命令。重启后继续时，必须确保：

- 以 root 用户身份执行 2.4 节之后的操作时，必须为 root 用户设置 LFS 环境变量。

2.3.2. 第 5-6 章

- 必须挂载 `/mnt/lfs` 分区。
- 这两章必须作为用户 lfs 完成。在执行这些章节的任何任务前，必须执行 `su - lfs` 命令。如果不这样做，可能会将软件包安装到宿主机上，甚至可能导致宿主机无法使用。
- 《通用编译指南》中的步骤至关重要。如果对软件包是否正确安装存在任何疑问，请确保删除之前解压的压缩包，然后重新解压该软件包，并完整执行该章节的所有指令。

2.3.3. 第 7-10 章

- 必须挂载/mnt/lfs 分区。
- 从"准备虚拟内核文件系统"到"进入 Chroot 环境"的若干操作，必须以 root 用户身份执行，并为 root 用户设置 LFS 环境变量。
- 进入 chroot 时，必须为 root 设置 LFS 环境变量。进入 chroot 环境后就不再使用 LFS 变量。
- 必须挂载虚拟文件系统。这可以在进入 chroot 之前或之后完成，方法是切换到主机虚拟终端，以 root 身份运行 7.3.1 节"挂载并填充/dev"和 7.3.2 节"挂载虚拟内核文件系统"中的命令。

2.4. 创建新分区

与大多数其他操作系统类似，LFS 通常需要安装在独立分区上。构建 LFS 系统的推荐方法是使用现有的空闲分区，或者如果您有足够的未分配空间，可以新建一个分区。

最小系统需要约 10GB 的分区空间，这足以存储所有源代码包并进行编译。但如果 LFS 系统计划作为主 Linux 系统使用，可能还需要安装额外软件，这将需要更多空间。30GB 的分区容量能合理满足后续扩展需求——LFS 系统本身并不会占用如此大的空间，该容量要求主要是为了确保临时存储空间充足，并支持完成 LFS 后添加额外功能。此外，编译软件包时可能暂时需要大量磁盘空间，这些空间在软件包安装完成后可被回收利用。

由于编译过程并不总是有足够的随机存取内存（RAM）可用，使用一个小型磁盘分区作为交换空间是个不错的选择。内核会利用该空间存储不常使用的数据，从而为活跃进程释放更多内存。LFS 系统的交换分区可以与宿主系统共用同一个，这种情况下就无需再创建新的交换分区。

启动磁盘分区工具（如 cfdisk 或 fdisk），通过命令行参数指定要创建新分区的硬盘——例如主磁盘驱动器为/dev/sda。根据需要创建一个 Linux 原生分区和一个交换分区。若您尚不熟悉这些工具的使用方法，请参阅 cfdisk(8)或 fdisk(8)手册页。

注意

对于有经验的用户，也可以采用其他分区方案。新的 LFS 系统可以安装在软件 RAID 阵列或 LVM 逻辑卷上。但请注意，部分方案需要 initramfs 支持，这属于高级主题。不建议首次构建 LFS 系统的用户采用这些分区方法。

请记录新分区的设备标识（例如 sda5），本书将称其为 LFS 分区。同时请记下交换分区的设备标识，后续配置/etc/fstab 文件时需要用到这些名称。

2.4.1. 其他分区注意事项

关于系统分区的建议请求经常出现在 LFS 邮件列表中。这是一个高度主观的话题。

大多数发行版的默认设置是使用整个驱动器，仅保留一个小的交换分区。这对 LFS 来说并非最佳选择，原因如下：它会降低灵活性，使得跨多个发行版或 LFS 构建共享数据更加困难，增加备份耗时，并可能因文件系统结构分配不当而浪费磁盘空间。

2.4.1.1. 根分区

对于大多数系统而言，20GB 的 LFS 根分区（不要与/root 目录混淆）是一个理想的折中方案。它既提供了足够的空间来构建 LFS 和大部分 BLFS，又足够紧凑以便轻松创建多个分区进行实验。

2.4.1.2. 交换分区

多数发行版会自动创建交换分区。通常建议交换分区大小为物理内存的两倍左右，但实际很少需要这么大。若磁盘空间有限，可将交换分区控制在 2GB 以内，并监控磁盘交换量。

若想使用 Linux 的休眠功能（挂起到磁盘），系统会在关机前将内存内容写入交换分区。此时交换分区的大小至少应与系统安装的内存容量相当。

交换行为绝非好事。对于机械硬盘，通常只需监听磁盘活动声响并观察系统对命令的响应速度，就能判断是否发生了交换。而使用固态硬盘时虽听不见交换声响，但可通过运行 top 或 free 程序查看交换空间使用量。应尽可能避免将 SSD 用作交换分区。当出现交换现象时，首先应检查是否执行了不合理命令（例如尝试编辑 5GB 的大文件）。若交换成为常态，最佳解决方案是为系统扩充内存容量。

2.4.1.3. GRUB BIOS 分区

若启动盘采用 GUID 分区表(GPT)进行分区，则需创建一个通常为 1MB 的小分区（若该分区不存在）。此分区无需格式化，但必须可供 GRUB 在安装引导加载程序时使用。使用 fdisk 工具时该分区通常标记为'BIOS Boot'，使用 gdisk 命令时其代码应为 EF02。

注意

GRUB BIOS 分区必须位于 BIOS 用于启动系统的驱动器上，该驱动器不一定是包含 LFS 根分区的驱动器。系统中的磁盘可能采用不同类型的分区表。GRUB BIOS 分区的必要性仅取决于启动盘的分区表类型。

2.4.1.4. 便利性分区 设计磁盘布局时，还有其他几种非必需但值得考虑的分区。以下列表虽非详尽无遗，但可作为参考指南。

- `/boot` – 强烈推荐。该分区用于存储内核及其他启动信息。为避免大容量磁盘可能引发的启动问题，建议将其设为第一块磁盘的首个物理分区。200MB 的分区容量已足够使用。
- `/boot/efi` – EFI 系统分区，采用 UEFI 方式启动系统时必需。详情请参阅 BLFS 页面说明。
- `/home` – 强烈推荐。用于在多发行版或 LFS 构建之间共享主目录及用户个性化配置。该分区通常需要较大容量，具体大小取决于可用磁盘空间。
- `/usr` – 在 LFS 中，`/bin`、`/lib` 和 `/sbin` 都是指向 `/usr` 中对应目录的符号链接。因此 `/usr` 包含了系统运行所需的所有二进制文件。对于 LFS 系统，通常不需要为 `/usr` 单独分区。若坚持创建，则应确保分区容量足以容纳系统中所有程序和库文件。在此配置下，根分区可以非常小（可能仅需 1GB），因此适合瘦客户端或无盘工作站（此时 `/usr` 通过远程服务器挂载）。但需注意，这种独立 `/usr` 分区配置需要 initramfs（LFS 未涵盖该内容）才能启动系统。
- `/opt` – 该目录对 BLFS 最为实用，可安装 KDE 或 Texlive 等大型软件包而无需将文件嵌入 `/usr` 层级结构。若使用此目录，通常 5 到 10GB 空间即可满足需求。
- `/tmp` – 单独划分 `/tmp` 分区较为罕见，但对配置瘦客户端很有帮助。若使用该分区，容量一般不超过几 GB 即可。若内存充足，可将 `tmpfs` 挂载到 `/tmp` 以加速临时文件访问。
- `/usr/src` —— 该分区非常实用，可为存储 BLFS 源码文件提供位置，并支持在多个 LFS 构建之间共享这些文件。它也可用作构建 BLFS 软件包的场所。30-50GB 的合理大分区能提供充足空间。

任何需要在系统启动时自动挂载的独立分区，都必须在 `/etc/fstab` 文件中进行指定。关于如何指定分区的详细信息将在第 10.2 节“创建 `/etc/fstab` 文件”中讨论。

2.5. 在分区上创建文件系统

分区仅是磁盘驱动器上由分区表设定的边界所限定的扇区范围。在操作系统能使用分区存储文件之前，必须将分区格式化为包含文件系统的形式——通常由卷标、目录块、数据块及按需定位特定文件的索引方案构成。文件系统还协助操作系统跟踪分区剩余空间，在创建新文件或扩展现有文件时预留所需扇区，并回收删除文件时产生的空闲数据段。它还可能提供数据冗余和错误恢复支持。

LFS 可以使用 Linux 内核支持的任何文件系统，但最常用的类型是 ext3 和 ext4。选择合适的文件系统可能比较复杂，这取决于文件特性和分区大小。例如：

ext2

适用于更新频率较低的小型分区，例如 /boot。

ext3 是 ext2 的升级版，增加了日志功能，可在非正常关机时帮助恢复分区状态。

它通常被用作通用文件系统。

ext4 是 ext 系列文件系统的最新版本。它提供了多项新功能，包括纳秒级时间戳、超大文件（最大 16TB）的创建与使用，以及速度提升。

其他文件系统如 FAT32、NTFS、JFS 和 XFS 适用于特定用途。更多关于这些文件系统的信息可访问：

https://en.wikipedia.org/wiki/Comparison_of_file_systems

LFS 假定根文件系统(/)采用 ext4 类型。要在 LFS 分区上创建 ext4 文件系统，请执行以下命令：

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

将替换为 LFS 分区的名称。

如果正在使用现有的交换分区，则无需格式化。如果是新建的交换分区，则需要用以下命令进行初始化：

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

将替换为交换分区的名称。

2.6. 设置\$LFS 变量与 umask 值

在本书中，我们会多次使用 LFS 环境变量。请确保在整个 LFS 构建过程中始终定义该变量。该变量应设置为构建 LFS 系统的目录名称——我们将以 /mnt/lfs 为例，但您可以选择任意目录名称。如果在独立分区上构建 LFS，该目录将作为分区的挂载点。请选择一个目录

位置并通过以下命令设置变量：

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

设置此变量的好处在于，可以原样输入诸如 `mkdir -v \$LFS/tools` 这样的命令。当 shell 处理命令行时，会自动将 "\$LFS" 替换为 "/mnt/lfs"（或该变量被设置的其他值）。

现在将文件模式创建掩码（umask）设置为 022，以防宿主发行版使用不同的默认值：

```
umask 022
```

将 umask 设为 022 能确保新创建的文件和目录仅对其所有者可写，但对所有用户可读且可搜索（仅针对目录）（假设 open(2) 系统调用使用默认模式，新文件最终将具有 644 权限模式，目录具有 755 权限模式）。过于宽松的默认设置会在 LFS 系统中留下安全漏洞，而过于严格的默认设置则可能导致构建或使用 LFS 系统时出现奇怪问题。

警告

无论何时离开并重新进入当前工作环境（例如使用 `su` 切换到 root 或其他用户时），切勿忘记检查 LFS 变量是否已设置且 `umask` 值是否为 022。可通过以下命令检查 LFS 变量是否正确设置：

```
echo $LFS
```

请确保输出显示的是 LFS 系统的构建路径，若遵循了提供的示例，则应为 `/mnt/lfs`。

使用以下命令检查 `umask` 设置是否正确：

```
umask
```

输出结果可能是 0022 或 022（前导零的数量取决于宿主发行版）。

如果这两个命令的任何输出不正确，请使用本页前面给出的命令将 `$LFS` 设置为正确的目录名称，并将 `umask` 设为 022。

注意

确保 LFS 变量和 `umask` 始终正确设置的一种方法是，在您个人主目录下的 `.bash_profile` 文件以及 `/root/.bash_profile` 文件中编辑并添加上文提到的 `export` 和 `umask` 命令。此外，所有需要 LFS 变量的用户在 `/etc/passwd` 文件中指定的 shell 必须是 `bash`，以确保 `.bash_profile` 文件能作为登录流程的一部分被加载。

另一个需要考虑的因素是登录主机系统的方式。如果通过图形显示管理器登录，当启动虚拟终端时，用户的 `.bash_profile` 通常不会被加载。这种情况下，需要将相关命令添加到用户和 root 的 `.bashrc` 文件中。另外，某些发行版会使用“if”条件测试，对于非交互式 `bash` 调用不会执行 `.bashrc` 中剩余的指令。请务必在这些命令放置在非交互式使用的条件测试之前。

2.7. 挂载新分区

既然文件系统已经创建完成，现在必须挂载该分区以便主机系统能够访问它。本书假设该文件系统将挂载到前文所述 LFS 环境变量指定的目录下。

严格来说，人们无法“挂载分区”，实际挂载的是该分区内嵌的文件系统。但由于单个分区不能包含多个文件系统，人们常将分区与其关联的文件系统视为同一事物。

使用以下命令创建挂载点并挂载 LFS 文件系统：

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

请将替换为 LFS 分区的名称。

如果你为 LFS 使用多个分区（例如一个用于`/`，另一个用于`/home`），请按以下方式挂载它们：

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

将和替换为适当的分区名称。

将\$LFS 目录（即新创建的 LFS 系统文件系统中的根目录）的所有者和权限模式设置为 root 和 755，以防主机发行版为 mkfs 配置了不同的默认值：

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

确保这个新分区没有以过于严格的权限挂载（例如 nosuid 或 nodev 选项）。运行不带任何参数的 mount 命令，查看已挂载的 LFS 分区设置了哪些选项。

如果设置了 nosuid 和/或 nodev 选项，必须重新挂载该分区。

警告

上述说明假设您在 LFS 构建过程中不会重启计算机。如果关闭系统，您需要在每次重新开始构建过程时重新挂载 LFS 分区，或者修改宿主系统的/etc/fstab 文件使其在重启时自动挂载。例如，您可以在/etc/fstab 文件中添加如下行：

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

如果使用了额外的可选分区，请确保也添加它们。

如果正在使用交换分区，请确保通过 swapon 命令启用它：

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

将 替换为交换分区的名称。

现在新的 LFS 分区已准备就绪，是时候下载软件包了。

第 3 章 软件包与补丁

3.1. 简介

本章列出了构建基础 Linux 系统需要下载的软件包清单。所列版本号均经过验证可正常使用，本书内容基于这些版本编写。我们强烈建议不要使用其他版本，因为针对某个版本的构建指令可能不适用于其他版本（除非该版本由 LFS 勘误或安全公告特别指定）。最新版本的软件包可能存在需要规避的问题，这些规避方案将在本书的开发版本中逐步完善并稳定。

对于某些软件包，其发布压缩包和对应版本的（Git 或 SVN）代码库快照压缩包可能会使用相似甚至完全相同的文件名。但发布压缩包除了包含对应代码库快照的内容外，还可能包含一些虽未存储在代码库中却至关重要的文件（例如由 autoconf 生成的 configure 脚本）。本书在可能的情况下均使用发布压缩包。若使用代码库快照替代本书指定的发布压缩包，将会引发问题。

下载地址可能并非始终可用。若某个下载地址自本书发布后已变更，可通过谷歌搜索引擎

[\(https://www.google.com/\)](https://www.google.com/) 查找大多数软件包。若搜索未果，请尝试访问

<https://www.linuxfromscratch.org/lfs/mirrors.html#files> 获取其他下载方式。

下载的软件包和补丁需要存放在整个构建过程中便于访问的位置。同时还需要一个工作目录来解压源码并进行构建。`$LFS/sources` 目录既可以用来存放压缩包和补丁文件，也可以作为工作目录使用。通过使用该目录，所需的文件都将位于 LFS 分区上，并在构建过程的所有阶段均可访问。

请在开始下载前，以 root 用户身份执行以下命令创建该目录：

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

将该目录设置为可写并启用粘滞位。“粘滞位”意味着即使多个用户对目录拥有写权限，在启用粘滞位的目录中只有文件所有者才能删除该文件。以下命令将启用写权限和粘滞位模式：

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

有几种方法可以获取构建 LFS 所需的所有软件包和补丁：

- 如后两节所述，这些文件可以单独下载。
- 对于稳定版本的书籍，可以从 <https://www.linuxfromscratch.org/mirrors.html#files> 列出的镜像站点下载包含所有必需文件的压缩包。
- 可以通过如下所述的 wget 命令和 wget-list 文件下载所需文件。

要使用 wget-list-sysv 作为 wget 命令的输入来下载所有软件包和补丁，请执行：

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

此外，从 LFS-7.0 版本开始，提供了一个单独的 md5sums 校验文件，用于在继续操作前验证所有软件包是否完整无误。请将该文件放入\$LFS/sources 目录并运行：

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

在使用上述任意方法获取所需文件后，均可进行此项校验。

如果以非 root 用户身份下载软件包和补丁，这些文件将归属于该用户。文件系统通过 UID 记录所有者，而宿主发行版中普通用户的 UID 在 LFS 系统中未被分配。因此这些文件在最终的 LFS 系统中将归属于一个未命名的 UID。如果您不打算在 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更改为 root 以避免此问题：

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

3.2. 所有软件包说明

下载软件包前请阅读安全公告，以确定是否需要使用某些软件包的更新版本以避免安全漏洞。

上游源代码可能会移除旧版本发布，特别是当这些版本存在安全漏洞时。如果下方某个 URL 无法访问，您应首先查阅安全公告，确认是否应使用修复了漏洞的新版本。若无需更新版本，可尝试从镜像站点下载被移除的软件包。请注意：即使能从镜像获取因漏洞被移除的旧版本，但在构建系统时使用已知存在漏洞的版本绝非明智之举。

请下载或通过其他方式获取以下软件包：

- Acl (2.3.2) - 363 KB:

主页: <https://savannah.nongnu.org/projects/acl> 下载:
<https://download.savannah.gnu.org/releases/acl/acl-2.3.2.tar.xz>
MD5 校验值: 590765dee95907dbc3c856f7255bd669

- 属性工具包(2.5.2) - 484 KB:

主页: <https://savannah.nongnu.org/projects/attr> 下载:
<https://download.savannah.gnu.org/releases/attr/attr-2.5.2.tar.gz>
MD5 校验值: 227043ec2f6ca03c0948df5517f9c927

- Autoconf (2.72 版) - 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 校验值: 1be79f7106ab6767f18391c5e22be701

- Automake (1.17 版) - 1,614 KB:

主页: <https://www.gnu.org/software/automake/> 下载地址:
<https://ftp.gnu.org/gnu/automake/automake-1.17.tar.xz>
MD5 校验值: 7ab3a02318fee6f5bd42adfc369abf10

- Bash (5.2.37 版) - 10,868 KB:

主页: <https://www.gnu.org/software/bash/>
下载地址: <https://ftp.gnu.org/gnu/bash/bash-5.2.tar.gz>
MD5 校验值: f21ff65de72ca329c1779684a972

- Bc (7.0.3) - 464 KB:

主页: <https://git.gavinhoward.com/gavin/bc>
下载地址: <https://github.com/gavinhoward/bc/releases/download/7.0.3/bc-7.0.3.tar.xz>
MD5 校验值: 4db5a0eb4fdbb3f6813be4b6b3da74

- Binutils (2.44 版) - 26,647 KB:

主页: <https://www.gnu.org/software/binutils/> 下载地址:
<https://sourceware.org/pub/binutils/releases/binutils-2.44.tar.xz>
MD5 校验值: 49912ce774666a30806141f106124294

- Bison (3.8.2) - 2,752 KB:

官网: <https://www.gnu.org/software/bison/> 下载地
址: <https://ftp.gnu.org/gnu/bison/bison-3.8.2.tar.xz>
MD5 校验值: c28f119f405a2304ff0a7ccdcc629713

- Bzip2 压缩工具 (1.0.8 版) - 792 KB:

下载地址: <https://www.sourceware.org/pub/bzip2/bzip2-1.0.8.tar.gz>
MD5 校验值: 67e051268d0c475ea773822f7500d0e5

- Check (0.15.2) - 760 KB:

主页: <https://libcheck.github.io/check> 下载地址:
<https://github.com/libcheck/check/releases/download/0.15.2/check-0.15.2.tar.gz>
MD5 校验值: 50fcacfecde5a380415b12e9c574e0b2

- Coreutils (9.6) - 5,991 KB:

主页: <https://www.gnu.org/software/coreutils/> 下载:
<https://ftp.gnu.org/gnu/coreutils/coreutils-9.6.tar.xz>
MD5 校验值: 0ed6cc983fe02973bc98803155cc1733

- DejaGNU (1.6.3) - 608 KB:

主页: <https://www.gnu.org/software/dejagnu/> 下载:
<https://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.3.tar.gz>
MD5 校验值: 68c5208c58236eba447d7d6d1326b821

- Diffutils (3.11 版) - 1,881 KB:

主页: <https://www.gnu.org/software/diffutils/> 下载地址:
<https://ftp.gnu.org/gnu/diffutils/diffutils-3.11.tar.xz>
MD5 校验值: 75ab2bb7b5ac0e3e10cece85bd1780c2

- E2fsprogs (1.47.2) - 9,763 KB:

主页: <https://e2fsprogs.sourceforge.net/> 下载地址:
<https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.47.2/e2fsprogs-1.47.2.tar.gz>
MD5 校验值: 752e5a3ce19aea060d8a203f2fae9baa

- Elfutils (0.192) - 11,635 KB:

主页: <https://sourceware.org/elfutils/> 下载地址:
<https://sourceware.org/ftp/elfutils/0.192/elfutils-0.192.tar.bz2>
MD5 校验值: a6bb1efc147302cfcc15b5c2b827f186a

- Expat (2.6.4) - 476 KB:

主页: <https://libexpat.github.io/> 下载地址:
<https://prdownloads.sourceforge.net/expat/expat-2.6.4.tar.xz>
MD5 校验值: 101fe3e320a2800f36af8cf4045b45c7

- Expect 软件包(5.45.4 版) - 618 KB:

主页: <https://core.tcl.tk/expect/> 下载地址:
<https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz>
MD5 校验值: 00fce8de158422f5cccd2666512329bd2

· 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 校验值: 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 校验值: 870cf71c07d37ebe56f9f4aaf4ad872

· 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 校验值: 2882e3179748cc9f9c23ec593d6adc8d

· Flit-core (3.11.0) - 51 KB:

主页: <https://pypi.org/project/flit-core/> 下载:

https://pypi.org/packages/source/f/flit-core/flit_core-3.11.0.tar.gz

MD5 校验值: 6d677b1acef1769c4c7156c7508e0dbd

· Gawk (5.3.1) - 3,428 KB:

主页: <https://www.gnu.org/software/gawk/> 下载:

<https://ftp.gnu.org/gnu/gawk/gawk-5.3.1.tar.xz>

MD5 校验值: 4e9292a06b43694500e0620851762eec

· GCC (14.2.0) - 90,144 KB:

主页: <https://gcc.gnu.org/> 下载地址:

<https://ftp.gnu.org/gnu/gcc/gcc-14.2.0/gcc-14.2.0.tar.xz>

MD5 校验值: 2268420ba02dc01821960e274711bde0

· GDBM (1.24 版) - 1,168 KB:

主页: <https://www.gnu.org/software/gdbm/> 下载地

址: <https://ftp.gnu.org/gnu/gdbm/gdbm-1.24.tar.gz>

MD5 校验值: c780815649e52317be48331c1773e987

· Gettext (0.24 版) - 8,120 KB:

主页: <https://www.gnu.org/software/gettext/> 下载地址:

<https://ftp.gnu.org/gnu/gettext/gettext-0.24.tar.xz>

MD5 校验值: 87aea3013802a3c60fa3feb5c7164069

· Glibc (2.41) - 18,892 KB:

主页: <https://www.gnu.org/software/libc/> 下载地址:

<https://ftp.gnu.org/gnu/glibc/glibc-2.41.tar.xz>

MD5 校验值: 19862601af60f73ac69e067d3e9267d4

注意

Glibc 开发者维护着一个 Git 分支，其中包含针对 Glibc-2.41 值得采用的补丁，但这些补丁不幸在 Glibc-2.41 发布后才开发完成。若该分支中添加了任何安全修复补丁，LFS 编辑团队将发布安全公告；对于其他新增补丁则不会采取特别措施。您可以自行审查这些补丁，若认为某些补丁重要，可自行将其整合使用。

· 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 校验值: 956dc04e864001a9c22429f761f2c283

· Gperf (3.1) - 1,188 KB:

主页: <https://www.gnu.org/software/gperf/> 下载地址: <https://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz>

MD5 校验值: 9e251c0a618ad0824b51117d5d9db87e

· Grep (3.11) - 1,664 KB:

主页: <https://www.gnu.org/software/grep/> 下载地址: <https://ftp.gnu.org/gnu/grep/grep-3.11.tar.xz>

MD5 校验值: 7c9bbd74492131245f7cdb291fa142c0

· 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 校验值: 5e4f40315a22bb8a158748e7d5094c7d

· GRUB (2.12 版) - 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 校验值: 60c564b1bdc39d8e43b3aab4bc0fb140

· Gzip (1.13 版) - 819 KB:

主页: <https://www.gnu.org/software/gzip/> 下载地址: <https://ftp.gnu.org/gnu/gzip/gzip-1.13.tar.xz>

MD5 校验值: d5c9fc9441288817a4a0be2da0249e29

· Iana-Etc (20250123) - 591 KB:

主页: <https://www.iana.org/protocols> 下载地址: <https://github.com/Mic92/iana-etc/releases/download/20250123/iana-etc-20250123.tar.gz>

MD5 校验值: f8a0ebdc19a5004cf42d8bdcf614fa5d

· Inetutils 工具包(2.6 版) - 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 校验值: 401d7d07682a193960bcddecaf0d03de94

· 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 校验值: 12e517cac2b57a0121cda351570f1e63

· IPRoute2 (6.13.0) - 906 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/iproute2-6.13.0.tar.xz>

- **Jinja2 (3.1.5) - 239 KB:**

主页: <https://jinja.palletsprojects.com/en/3.1.x/> 下载:
<https://pypi.org/packages/source/J/Jinja2/jinja2-3.1.5.tar.gz>

MD5 校验值: 083d64f070f6f1b5f75971ae60240785

- **Kbd (2.7.1) - 1,438 KB:**

主页: <https://kbd-project.org/> 下载:
<https://www.kernel.org/pub/linux/utils/kbd/kbd-2.7.1.tar.xz>

MD5 校验值: f15673d9f748e58f82fa50cff0d0fd20

- **Kmod (34) - 331 KB:**

主页: <https://github.com/kmod-project/kmod> 下载地址:
<https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-34.tar.xz>

MD5 校验值: 3e6c5c9ad9c7367ab9c3cc4f08dfde62

- **Less (668) - 635 KB:**

主页: <https://www.greenwoodsoftware.com/less/> 下载:
<https://www.greenwoodsoftware.com/less/less-668.tar.gz>

MD5 校验值: d72760386c5f80702890340d2f66c302

- **LFS 启动脚本 (20240825) - 33 KB:**

下载地址: <https://www.linuxfromscratch.org/lfs/downloads/12.3/lfs-bootscripts-20240825.tar.xz>
MD5 校验值: 7b078c594a77e0f9cd53a0027471c3bc

- **Libcap (2.73 版) - 191 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-2.73.tar.xz>

MD5 校验值: 0e186df9de9b1e925593a96684fe2e32

- **Libffi (3.4.7 版) - 1,362 KB:**

主页: <https://sourceware.org/libffi/> 下载地址:
<https://github.com/libffi/libffi/releases/download/v3.4.7/libffi-3.4.7.tar.gz>

MD5 校验值: 696a1d483a1174ce8a477575546a5284

- **Libpipeline (1.5.8) - 1046 KB:**

主页: <https://libpipeline.nongnu.org/> 下载:
<https://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.8.tar.gz>

MD5 校验值: 17ac6969b2015386bcb5d278a08a40b5

- **Libtool (2.5.4) - 1,033 KB:**

主页: <https://www.gnu.org/software/libtool/> 下载:
<https://ftp.gnu.org/gnu/libtool/libtool-2.5.4.tar.xz>

MD5 校验值: 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 校验值: 1796a5d20098e9dd9e3f576803c83000

- Linux (6.13.4) - 145,015 KB:

主页: <https://www.kernel.org/> 下载地址:

<https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.13.4.tar.xz>

MD5 校验值: 13b9e6c29105a34db4647190a43d1810

注意

Linux 内核更新非常频繁，多数情况是由于发现安全漏洞所致。除非勘误页另有说明，否则建议使用最新的稳定内核版本。对于网速较慢或带宽费用较高的用户，若需更新 Linux 内核，可分别下载该软件包的基础版本和补丁文件。这种方式在次要版本内进行补丁级别升级时，可能节省后续更新时间或成本。

- 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 校验值: dead9f5f1966d9ae56e1e32761e4e675

- M4 (1.4.19) - 1,617 KB:

主页: <https://www.gnu.org/software/m4/> 下载地

址: <https://ftp.gnu.org/gnu/m4/m4-1.4.19.tar.xz>

MD5 校验值: 0d90823e1426f1da2fd872df0311298d

- 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 校验值: c8469a3713cbbe04d955d4ae4be23eeb

- Man-DB (2.13.0) - 2,023 KB:

主页: <https://www.nongnu.org/man-db/> 下载地址:

<https://download.savannah.gnu.org/releases/man-db/man-db-2.13.0.tar.xz>

MD5 校验值: 97ab5f9f32914eef2062d867381d8cee

- Man-pages (6.12) - 1,838 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-6.12.tar.xz>

MD5 校验值: 44de430a598605eaba3e36dd43f24298

- 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 校验值: cb0071711b573b155cc8f86e1de72167

- Meson (1.7.0) - 2,241 KB:

主页: <https://mesonbuild.com> 下载地址:

<https://github.com/mesonbuild/meson/releases/download/1.7.0/meson-1.7.0.tar.gz>

MD5 校验值: c20f3e5ebbb007352d22f4fd6ceb925c

- 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 校验值: 5c9bc658c9fd0f940e8e3e0f09530c62

· MPFR (4.2.1) - 1,459 KB:

主页: <https://www.mpfr.org/> 下载地址:

<https://ftp.gnu.org/gnu/mpfr/mpfr-4.2.1.tar.xz>

MD5 校验值: 523c50c6318dde6f9dc523bc0244690a

· Ncurses (6.5) - 2,156 KB:

主页: <https://www.gnu.org/software/ncurses/> 下载:

<https://invisible-mirror.net/archives/ncurses/ncurses-6.5.tar.gz>

MD5 校验值: ac2d2629296f04c8537ca706b6977687

· Ninja (1.12.1) - 235 KB:

主页: <https://ninja-build.org/> 下载: <https://github.com/ninja-build/ninja/archive/v1.12.1/ninja-1.12.1.tar.gz>

MD5 校验值: 6288992b05e593a391599692e2f7e490

· OpenSSL (3.4.1 版) - 17,917 KB:

主页: <https://www.openssl-library.org/> 下载地址:

<https://github.com/openssl/openssl/releases/download/openssl-3.4.1/openssl-3.4.1.tar.gz>

MD5 校验值: fb7a747ac6793a7ad7118eaba45db379

· 补丁工具 (2.7.6) - 766 KB:

主页: <https://savannah.gnu.org/projects/patch/> 下载地

址: <https://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz>

MD5 校验值: 78ad9937e4caadcba1526ef1853730d5

· Perl 语言 (5.40.1) - 13,605 KB:

官方网站: <https://www.perl.org/> 下载地址:

<https://www.cpan.org/src/5.0/perl-5.40.1.tar.xz>

MD5 校验值: bab3547a5cdf2302ee0396419d74a42e

· Pkgconf (2.3.0) - 309 KB:

官方网站: <https://github.com/pkgconf/pkgconf> 下载地址:

<https://distfiles.ariadne.space/pkgconf/pkgconf-2.3.0.tar.xz>

MD5 校验值: 833363e77b5bed0131c7bc4cc6f7747b

· Procps 软件包(4.0.5 版) - 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 校验值: 90803e64f51f192f3325d25c3335d057

· Psmisc (23.7 版) - 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 校验值: 53eae841735189a896d614cba440eb10

· Python (3.13.2 版) - 22,091 KB:

主页: <https://www.python.org/> 下载地址:

<https://www.python.org/ftp/python/3.13.2/Python-3.13.2.tar.xz>

MD5 校验值: 4c2d9202ab4db02c9d0999b14655dfe5

· Python 官方文档(3.13.2 版) - 10,102 KB:

下载地址: <https://www.python.org/ftp/python/doc/3.13.2/python-3.13.2-docs-html.tar.bz2>

MD5 校验值: d6aede88f480a018d26b3206f21654ae

- Readline (8.2.13) - 2,974 KB:

主页: <https://tiswww.case.edu/php/chet/readline/rltop.html>

下载: <https://ftp.gnu.org/gnu/readline/readline-8.2.13.tar.gz>

MD5 校验值: 05080bf3801e6874bb115cd6700b708f

- 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 校验值: 6aac9b2dbafcd5b7a67a8a9bcb8036c3

- Setuptools (75.8.1) - 1,313 KB:

主页: <https://pypi.org/project/setuptools/> 下载地址:

<https://pypi.org/packages/source/s/setuptools/setuptools-75.8.1.tar.gz>

MD5 校验值: 7dc3d3f529b76b10e35326e25c676b30

- Shadow 软件包 (4.17.3 版) - 2,274 KB:

主页: <https://github.com/shadow-maint/shadow/> 下载地址: <https://github.com/shadow-maint/shadow/releases/download/4.17.3/shadow-4.17.3.tar.xz>

MD5 校验值: 0da190e53ecee76237e4c8f3f39531ed

- Sysklogd (2.7.0) - 465 KB:

主页: <https://www.infodrom.org/projects/sysklogd/> 下载地址:

<https://github.com/troglbit/sysklogd/releases/download/v2.7.0/sysklogd-2.7.0.tar.gz>

MD5 校验值: 611c0fa5c138eb7a532f3c13bdf11ebc

- Systemd (257.3) - 15,847 KB:

主页: <https://www.freedesktop.org/wiki/Software/systemd/> 下载地址:

<https://github.com/systemd/systemd/archive/v257.3/systemd-257.3.tar.gz>

MD5 校验值: 8e4fc90c7ead651fa5c50bd1b34abc2

- Systemd 手册页 (257.3 版) - 733 KB:

主页: <https://www.freedesktop.org/wiki/Software/systemd/> 下载地址:

<https://anduin.linuxfromscratch.org/LFS/systemd-man-pages-257.3.tar.xz>

MD5 校验值: 9b77c3b066723d490cb10aed4fb05696

注意

Linux From Scratch 团队使用 systemd 源码自行生成了 man 手册页的压缩包，此举旨在避免引入不必要的依赖项。

- SysVinit (3.14 版) - 236 KB:

主页: <https://savannah.nongnu.org/projects/sysvinit> 下载地址:

<https://github.com/slicer69/sysvinit/releases/download/3.14/sysvinit-3.14.tar.xz>

MD5 校验值: bc6890b975d19dc9db42d0c7364dd092

- Tar (1.35 版) - 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 校验值: a2d8042658cf8ea939e6d911eaf4152

- **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 校验值: eaef5d0a27239fb840f04af8ec608242

- **Tcl 文档(8.6.16 版) - 1,169 KB:**

下载地址: <https://downloads.sourceforge.net/tcl/tcl8.6.16-html.tar.gz>

MD5 校验值: 750c221bcb6f8737a6791c1fbe98b684

- **Texinfo (7.2 版) - 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 校验值: 11939a7624572814912a18e76c8d8972

- **时区数据 (2025a) - 453 KB:**

主页: <https://www.iana.org/time-zones> 下载: <https://www.iana.org/time-zones/repository/releases/tzdata2025a.tar.gz>

MD5 校验值: 404229390c06b7440f5e48d12c1a3251

- **Udev-lfs 压缩包 (udev-lfs-20230818) - 10 KB:**

下载链接: <https://anduin.linuxfromscratch.org/LFS/udev-lfs-20230818.tar.xz>

MD5 校验值: acd4360d8a5c3ef320b9db88d275dae6

- **Util-linux 工具集 (2.40.4 版) - 8,641 KB:**

项目主页: <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/v2.40/util-linux-2.40.4.tar.xz>

MD5 校验值: b1c8315d8ccbeb0ec36d10350304

- **Vim 编辑器 (9.1.1166 版) - 18,077 KB:**

官网: <https://www.vim.org> 下载地址:

<https://github.com/vim/vim/archive/v9.1.1166/vim-9.1.1166.tar.gz>

MD5 校验值: 718d43ce957ab7c81071793de176c2eb

注意

vim 版本每日更新。获取最新版本请访问: <https://github.com/vim/vim/tags>

- **Wheel (0.45.1) - 106 KB:**

主页: <https://pypi.org/project/wheel/> 下载:

<https://pypi.org/packages/source/w/wheel/wheel-0.45.1.tar.gz>

MD5 校验值: dddc505d0573d03576c7c6c5a4fe0641

- **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 校验值: 89a8e82cf2ad948b349c0a69c494463

- **Xz 压缩工具 (5.6.4) - 1,310 KB:**

主页: <https://tukaani.org/xz> 下载地址: <https://github.com/tukaani-project/xz/releases/download/v5.6.4/xz-5.6.4.tar.xz>

MD5 校验值: 4b1cf07d45ec7eb90a01dd3c00311a3e

- 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 校验值: 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 校验值: 780fc1896922b1bc52a4e90980cdda48

这些软件包总大小: 约 527 MB

3.3. 所需补丁

除了软件包外, 还需要若干补丁程序。这些补丁用于修正软件包中本应由维护者修复的错误。补丁还会对软件包进行小幅修改, 使其更易于使用

构建 LFS 系统需要以下补丁:

- Bzip2 文档安装补丁 - 1.6 KB:

下载地址: https://www.linuxfromscratch.org/patches/lfs/12.3/bzip2-1.0.8-install_docs-1.patch
MD5 校验值: 6a5ac7e89b791aae556de0f745916f7f

- Coreutils 国际化修复补丁 - 164 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/coreutils-9.6-i18n-1.patch>
MD5 校验值: 6aee45dd3e05b7658971c321d92f44b7

- Expect GCC14 补丁 - 7.8 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/expect-5.45.4-gcc14-1.patch>
MD5 校验值: 0b8b5ac411d011263ad40b0664c669f0

- Glibc FHS 补丁 - 2.8 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/glibc-2.41-fhs-1.patch>
MD5 校验值: 9a5997c3452909b1769918c759eff8a2

- 键盘退格/删除键修复补丁 - 12 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/kbd-2.7.1-backspace-1.patch>
MD5 校验值: f75cca16a38da6caa7d52151f7136895

- SysVinit 综合补丁 - 2.5 KB:

下载地址: <https://www.linuxfromscratch.org/patches/lfs/12.3/sysvinit-3.14-consolidated-1.patch>
MD5 校验值: 3af8fd8e13cad481eeeeaa48be4247445

这些补丁的总大小: 约 190.7 KB

除了上述必需的补丁外, LFS 社区还创建了若干可选补丁。这些可选补丁能解决小问题或启用默认未开启的功能。欢迎浏览位于 <https://www.linuxfromscratch.org/patches/downloads/> 的补丁数据库, 根据系统需求获取额外补丁。

第 4 章 最终准备工作

4.1. 简介

本章将执行若干额外任务来为构建临时系统做准备。我们将在\$LFS 中创建一组目录（用于安装临时工具）、添加非特权用户，并为该用户创建合适的构建环境。同时会解释用于衡量 LFS 软件包构建耗时的“SBU”时间单位，并提供有关软件包测试套件的信息。

4.2. 在 LFS 文件系统中创建有限目录结构

本节开始向 LFS 文件系统填充构成最终 Linux 系统的各个组件。

第一步是创建一个有限的目录层次结构，以便将第 6 章编译的程序（以及第 5 章的 glibc 和 libstdc++）安装到它们的最终位置。这样做是为了在第 8 章构建最终版本时覆盖这些临时程序。

以 root 身份执行以下命令创建所需的目录结构：

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

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

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

第六章中的程序将使用交叉编译器进行编译（更多细节可参阅工具链技术说明章节）。这个交叉编译器会被安装在一个特殊目录中，以与其他程序隔离。仍以 root 身份执行以下命令创建该目录：

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

注意

LFS 开发团队特意决定不使用/usr/lib64 目录。我们采取了多项措施确保工具链不会使用该目录。如果该目录因任何原因出现（可能是由于您未遵循操作说明，或在完成 LFS 后安装了创建该目录的二进制包），可能会导致系统故障。您应始终确保该目录不存在。

4.3. 添加 LFS 用户

以 root 用户身份登录时，任何微小失误都可能导致系统损坏。因此，接下来两章的软件包将作为非特权用户进行构建。虽然可以使用您现有的用户名，但为了便于建立纯净的工作环境，我们将创建一个名为 lfs 的新用户（同时创建同名用户组 lfs），并在安装过程中以该用户身份执行命令。

以 root 身份执行以下命令添加新用户：

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

命令行参数含义如下：

-s /bin/bash

这将使 bash 成为用户 lfs 的默认 shell。

```
-g lfs
```

此选项将用户 lfs 添加到 lfs 组中。

-m

这将为 lfs 用户创建主目录。

```
-k /dev/null
```

该参数通过将输入位置改为特殊的空设备，防止可能从骨架目录（默认为/etc/skel）复制文件。

lfs

这是新用户的名称。

如果您希望以 lfs 身份登录，或从非 root 用户切换至 lfs（与以 root 身份登录时切换至 lfs 用户不同，后者无需为 lfs 用户设置密码），则需要为 lfs 设置密码。请以 root 用户身份执行以下命令来设置密码：

```
passwd lfs
```

通过将\$LFS 下所有目录的所有权赋予 lfs 用户，授予其完全访问权限：

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

注意

在某些宿主系统中，以下 su 命令可能无法正常完成，并会将 lfs 用户的登录会话挂起到后台。若提示符“lfs:~\$”未立即出现，输入 fg 命令可解决此问题。

接下来，启动一个以用户 lfs 身份运行的 shell。这可以通过在虚拟控制台以 lfs 身份登录实现，或者使用以下替代/切换用户命令：

```
su - lfs
```

这里的“-”指示 su 启动一个登录 shell，而非非登录 shell。这两种 shell 类型的区别在 bash(1) 和 info bash 中有详细说明。

4.4. 设置环境 通过为 bash shell 创建两个新的启动文件来建立良好的工作环境。在以用户 lfs 身份登录时，

执行以下命令创建新的.bash_profile 文件：

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

当以 lfs 用户身份登录，或使用带“-”选项的 su 命令切换到 lfs 用户时，初始 shell 是登录 shell，它会先读取主机的/etc/profile 文件（可能包含某些设置和环境变量），然后读取.bash_profile。.bash_profile 文件中的 exec env - i.../bin/bash 命令会用全新的空环境替换当前运行的 shell，仅保留 HOME、TERM 和 PS1 变量。这确保不会从主机系统泄漏任何不必要且可能有害的环境变量到构建环境中。

新启动的 shell 实例是一个非登录 shell，它不会读取并执行/etc/profile 或.bash_profile 文件的内容，而是会读取并执行.bashrc 文件。现在创建.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 中设置项的含义

set +h

执行 set +h 命令会关闭 bash 的哈希功能。哈希通常是个实用特性——bash 通过哈希表记录可执行文件的完整路径，避免重复搜索 PATH 来定位相同程序。但新工具安装后需要立即投入使用。关闭哈希功能会强制 shell 在每次运行程序时重新搜索 PATH。这样一来，一旦 \$LFS/tools/bin 中的新编译工具就绪，shell 就能立即找到它们，而不会记住宿主发行版在 /usr/bin 或 /bin 中提供的旧版本程序。

umask 022

如我们在第 2.6 节“设置\$LFS 变量与 umask”中所述，此处设置 umask 值。

LFS=/mnt/lfs

LFS 变量应设置为选定的挂载点。

LC_ALL=POSIX

LC_ALL 变量控制某些程序的本地化行为，使其消息遵循指定国家的惯例。将 LC_ALL 设置为“POSIX”或“C”（两者等效）可确保在交叉编译环境中一切按预期工作。

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

LFS_TGT 变量用于设置一个非默认但兼容的机器描述，在构建交叉编译器、链接器以及交叉编译临时工具链时使用。更多技术细节可参阅工具链技术说明。

PATH=/usr/bin

许多现代 Linux 发行版已将/bin 和/usr/bin 目录合并。这种情况下，第六章环境中的标准 PATH 变量应设置为/usr/bin/。若未合并目录，则需添加以下路径：

bin 到路径。

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

如果/bin 不是符号链接，则必须将其添加到 PATH 变量中。

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

将\$LFS/tools/bin 置于标准 PATH 之前，使得第五章开头安装的交叉编译器在安装完成后立即被 shell 识别。这一措施配合关闭哈希功能，能有效降低误用宿主系统编译器而非交叉编译器的风险。

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

在第五章和第六章中，若未设置该变量，配置脚本可能会尝试从宿主系统的/usr/share/config.site 加载特定发行版的配置项。通过覆盖此变量可防止宿主系统的潜在污染。

```
export ...
```

虽然前面的命令已经设置了一些变量，但为了让它们在所有子 shell 中可见，我们需要将这些变量导出。

重要提示

一些商业发行版在初始化 bash 时会添加一个未记录的/etc/bash.bashrc 实例。这个文件可能会以影响关键 LFS 软件包构建的方式修改 lfs 用户的环境。为确保 lfs 用户环境的纯净性，请检查是否存在/etc/

`bash.bashrc` 文件，如果存在，请将其移走。以 root 用户身份运行以下命令：

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

当不再需要 lfs 用户时（在第 7 章开始时），您可以安全地恢复/etc/bash.bashrc（如果需要）。

请注意，我们将在第 8.36 节“Bash-5.2.37”中构建的 LFS Bash 软件包并未配置为加载或执行/etc/bash.bashrc，因此该文件在完整的 LFS 系统上是无用的。

对于配备多处理器（或多核心）的现代系统，通过告知 make 程序可用的处理器数量（通过命令行选项或环境变量），执行“并行编译”可以显著缩短软件包的编译时间。例如，英特尔酷睿 i9-13900K 处理器拥有 8 个 P（性能）核心和 16 个 E（能效）核心，每个 P 核心可同时运行两个线程，因此 Linux 内核会将每个 P 核心模拟为两个逻辑核心。最终系统将显示总计 32 个逻辑核心。最直接的利用方式就是允许 make 命令同时启动最多 32 个编译任务，这可以通过向 make 传递`-j32` 参数实现：

```
make -j32
```

或者设置 `MAKEFLAGS` 环境变量，其内容会被 make 自动识别为命令行参数：

```
export MAKEFLAGS=-j32
```

重要提示

切勿在未指定数字的情况下向 make 传递`-j` 选项，或在 `MAKEFLAGS` 中设置此类选项。否则将导致 make 启动无限数量的编译任务，引发系统稳定性问题。

要为第五章和第六章的软件包编译启用所有可用逻辑核心，请立即在`.bashrc` 中设置 `MAKEFLAGS`：

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

若不想使用全部逻辑核心，请将 `$(nproc)` 替换为你希望使用的逻辑核心数量。

最后，为确保环境完全准备好构建临时工具，强制 bash shell 读取新的用户配置文件：

```
source ~/.bash_profile
```

4.5. 关于 SBU 的说明

许多用户都希望预先了解编译安装每个软件包大致所需的时间。

由于 Linux From Scratch 可以在多种不同配置的系统上构建，因此无法提供绝对准确的时间预估。最大的软件包（gcc）在最快的系统上可能仅需 5 分钟，但在性能较弱的系统上可能耗时数日！为此我们将采用标准构建单位（SBU）作为衡量标准，而非提供具体时间。

SBU 的计算方法如下：第五章中首个编译的软件包是 binutils。使用单核处理器完成其编译所需的时间，即被定义为一个标准构建单位（SBU）。其他所有软件包的编译时长都将以此时间单位为基准进行表述。

举例来说，假设某个软件包的编译时间为 4.5 SBU。这意味着如果您的系统编译安装第一遍 binutils 耗时 4 分钟，那么构建该示例软件包将需要约 18 分钟。

幸运的是，大多数编译时间都短于 1 个 SBU。

SBU 并非完全精确，因为它取决于诸多因素（包括宿主系统的 GCC 版本）。此处提供 SBU 数值仅用于预估软件包安装时长，某些情况下实际耗时可能出现数十分钟的偏差。

在某些较新的系统中，主板能够控制系统时钟速度。这可以通过诸如 powerprofilesctl 等命令进行控制。LFS 系统暂不支持此功能，但宿主发行版可能提供该功能。待 LFS 系统构建完成后，可参照 BLFS 的 power-profiles-daemon 页面所述流程添加该功能。在测量任何软件包的构建时间前，建议将系统电源方案设置为最高性能模式（同时功耗也最大）。否则测得的 SBU 值可能不准确，因为系统在构建 binutils-pass1 或其他软件包时可能表现不同。需注意即使对两个软件包采用相同电源方案，仍可能出现显著误差，因为若系统在开始构建流程时处于空闲状态，其响应速度可能较慢。将电源方案设为“performance”模式可最大限度避免此问题。显然，这样做也能加快 LFS 系统的构建速度。

若系统支持 powerprofilesctl 命令，请执行 `powerprofilesctl set performance` 以启用性能模式。某些发行版使用 `tuned-adm` 命令替代 powerprofilesctl 进行配置管理，在此类系统上应执行 `tuned-adm profile throughput-performance` 来启用吞吐性能模式。

注意

当采用多处理器并行编译时，本书中的 SBU（标准编译单位）数值波动将比常规情况更为显著。某些情况下 make 步骤可能直接失败。由于不同进程的输出信息会交错显示，构建过程的日志分析也将更加困难。若遇到编译问题，请切换回单处理器模式以便准确分析错误信息。

本书所列所有软件包的编译时间（除基于单核编译的 binutils-pass1 外）均基于四核并行编译（-j4 参数）。第八章中的时间数据若无特别说明，均包含执行软件包回归测试所耗时间。

4.6. 关于测试套件

大多数软件包都提供测试套件。对新构建的软件包运行测试套件是个好主意，因为它能提供“健全性检查”，表明所有内容都正确编译。通过全套检查的测试套件通常能证明该软件包按开发者预期的方式运行。但这并不保证该软件包完全没有错误。

某些测试套件比其他套件更为重要。例如，核心工具链软件包（GCC、binutils 和 glibc）的测试套件至关重要，因为它们在系统正常运行中扮演核心角色。GCC 和 glibc 的测试套件可能需要很长时间才能完成，特别是在较慢的硬件上，但强烈建议运行这些测试。

注意

在第五章和第六章运行测试套件没有意义；由于测试程序是用交叉编译器编译的，它们很可能无法在构建主机上运行。

运行 binutils 和 GCC 测试套件时常见的问题是伪终端（PTYs）耗尽，这可能导致大量测试失败。该问题可能由多种原因引起，但最可能的原因是主机系统未正确设置 devpts 文件系统。此问题在 <https://www.linuxfromscratch.org/lfs/faq.html#no-ptys> 中有更详细的讨论。

有时软件包的测试套件会因开发者已知且认为非关键性的原因而失败。请查阅位于 <https://www.linuxfromscratch.org/lfs/build-logs/12.3/> 的日志文件，以确认这些失败是否在预期范围内。该网站适用于本书中所有测试套件的验证。

第三部分 构建 LFS 交叉工具链与临时工具

重要预备材料

引言

本部分分为三个阶段：首先构建交叉编译器及其相关库；其次利用这套交叉工具链构建若干实用程序，使其与宿主系统保持隔离；最后进入 chroot 环境（进一步增强宿主隔离）并构建最终系统所需的剩余工具。

重要提示

从这里开始，才是真正构建新系统的起点。请严格按照书中所示步骤操作，务必格外谨慎。您应当尝试理解每条命令的作用，无论多么渴望完成构建，都应避免盲目照搬输入命令。遇到不理解的内容时，请查阅相关文档。同时建议使用 tee 工具将终端输出保存至文件，以便记录您的操作和命令输出结果，这在出现问题时能简化调试过程。

下一节将介绍构建过程的技术要点，随后的章节则包含极其重要的通用操作指南。

工具链技术说明

本节阐述整体构建方法背后的技术原理与细节。不必强求立即理解本节所有内容，实际完成构建后，多数概念会变得清晰易懂。您可以在构建过程中的任何阶段重新研读本章节。

第 5 章和第 6 章的总体目标是建立一个临时区域，其中包含一组已知可靠且与主机系统隔离的工具。通过使用 chroot 命令，后续章节的编译工作将被隔离在该环境中，确保目标 LFS 系统的构建过程干净且无故障。整个构建流程的设计既降低了新读者的操作风险，又同时提供了最佳的学习价值。

该构建流程基于交叉编译技术。交叉编译通常用于在与构建机器不同的目标机器上构建编译器及其相关工具链。虽然对 LFS 而言这并非绝对必要（因为新系统运行的机器与构建机器相同），但交叉编译具有一个显著优势：任何经过交叉编译的内容都不会依赖于主机环境。

关于交叉编译

注意

LFS 手册并非（也不包含）构建跨平台（或本机）工具链的通用教程。除非您确实理解自己在做什么，否则不要将书中的命令用于构建 LFS 以外的跨平台工具链目的。

交叉编译涉及一些值得单独章节阐述的概念。虽然初次阅读时可跳过本节，但后续回顾将帮助您更全面地理解整个过程。

让我们先定义本文语境中使用的一些术语。

构建机 (build) 指我们编译程序的机器。注意该机器也被称为"宿主机 (host)"。而运行机 (host) 则是构建好的程序将运行的机器/系统。需注意此处"host"的用法与其他章节不同。目标机 (target) 仅针对编译器而言，指编译器生成代码所针对的机器，它可能与构建机和运行机都不同。

例如，让我们设想以下场景（有时被称为"加拿大交叉编译"）。我们只有一台慢速机器上的编译器，称之为机器 A，其编译器为 ccA。同时我们有一台快速机器（B），但该机器没有编译器，而我们需要为第三台慢速机器（C）生成代码。我们将分三个阶段构建针对机器 C 的编译器。

阶段 构建主机 目标平台 操作

1	A	A	B 构建 交叉编译 编译器 正在使用 cc1 在 ccA 上 机器 A.
2	A	B	C 语言构建 交叉 编译器 使用 cc2 开启 cc1 机器 A.
3	B	C	C 语言构建 编译器 使用 C 语言 开启 cc2 机器 B.

随后，机器 C 所有的所有程序都可以在快速机器 B 上使用 cc2 进行编译。需要注意的是，除非机器 B 能够运行为机器 C 生成的程序，否则在机器 C 自身运行之前，无法测试这些新构建的程序。例如，要在 ccC 上运行测试套件，我们可能需要添加第四个阶段：

阶段 构建主机 目标 操作

4	C	C	C 重建 并测试 使用 ccC 在 ccC 上 机器 C.
---	---	---	--

在上面的例子中，只有 cc1 和 cc2 是交叉编译器，也就是说它们生成的代码运行在与编译器本身不同的机器上。其他编译器 ccA 和 ccC 生成的代码则运行在它们所在的机器上，这类编译器被称为本地编译器。

LFS 交叉编译的实现方案

注意

本书中所有交叉编译的软件包都采用基于 autoconf 的构建系统。这种构建系统接受以 cpu-厂商-内核-操作系统形式表示的系统类型，称为系统三元组。由于厂商字段通常无关紧要，autoconf 允许省略该字段。

敏锐的读者可能会疑惑为什么“三元组”却指代四个组成部分的名称。内核字段和操作系统字段最初是合并为一个“系统”字段的。这种三字段形式至今仍适用于某些系统，例如 x86_64-unknown-freebsd。但两个系统可能共享相同内核却差异巨大，无法使用相同的三元组来描述。例如运行在手机上的 Android 系统与运行在 ARM64 服务器上的 Ubuntu 系统就截然不同，尽管它们都使用相同类型的 CPU (ARM64) 和相同内核 (Linux)。

若没有模拟层，您无法在手机上运行服务器可执行文件，反之亦然。因此“system”字段被拆分为 kernel 和 os 字段，以明确区分这些系统。在我们的示例中，Android 系统被标记为 aarch64-unknown-linux-android，而 Ubuntu 系统则被

标记为 aarch64-unknown-linux-gnu。

“三元组”这个术语仍保留在专业词汇中。确定您系统三元组的简单方法是运行许多软件包源码中附带的 config.guess 脚本。解压 binutils 源码包，运行 ./config.guess 脚本并记录输出结果。例如，对于 32 位 Intel 处理器，输出将是 i686-pc-linux-gnu；在 64 位系统上则会显示 x86_64-pc-linux-gnu。在大多数 Linux 系统上，更简单的 gcc -dumpmachine 命令也能提供类似信息。

您还应当了解平台动态链接器的名称，它常被称为动态加载器（注意不要与 binutils 工具集中的标准链接器 ld 混淆）。由 glibc 软件包提供的动态链接器负责查找并加载程序所需的共享库，为程序运行做好准备，然后启动程序。32 位 Intel 架构机器的动态链接器名为 ld-linux.so.2；而 64 位版本则称为 ld-linux-x86-64。

在 64 位系统上为 so.2。确定动态链接器名称的可靠方法是检查宿主系统中的任意二进制文件，运行命令：readelf -1 <二进制文件名> | grep interpreter 并记录输出结果。涵盖所有平台的权威参考资料可在 Glibc 维基页面找到。

为了在 LFS 中模拟交叉编译，我们通过修改 LFS_TGT 变量中的“vendor”字段为“lfs”来微调主机三元组名称。在构建交叉链接器和交叉编译器时，我们还使用了--with-sysroot 选项，以告知它们所需主机文件的查找位置。这确保了第 6 章构建的其他程序都不会链接到构建机器上的库文件。整个流程仅需两个强制阶段，外加一个可选的测试阶段。

阶段 构建主机 目标 操作

1	个人电脑	个人电脑	LFS 构建	
			交叉编译	
			编译器	
			cc1 使用中	
			cc-pc 启用	
			个人电脑	

阶段 构建主机 目标 操作

2	个人电脑 Linux 从零开始构建		
		编译器 cc-lfs 使用 cc1 编译器 在个人电脑上	
3	lfs	lfs	LFS 重建
			并测试 cc-lfs 使用 CC- LFS 上的 LFS

在前述表格中，"on pc"表示命令在使用已安装发行版的机器上运行。"On lfs"表示命令在 chroot 环境中运行。这还不是故事的结尾。C 语言不仅仅是一个编译器；它还定义了一个标准库。本书使用的是名为 glibc 的 GNU C 库（另有替代方案"musl"）。这个库必须为 LFS 系统编译，即使用交叉编译器 cc1。但编译器本身使用一个内部库，为汇编指令集不支持的函数提供复杂的子程序。这个内部库名为 libgcc，它必须链接到 glibc 库才能完全发挥作用。此外，C++的标准库 (libstdc++) 也必须与 glibc 链接。解决这个先有鸡还是先有蛋问题的方法是：首先构建一个功能受限的基于 cc1 的 libgcc（缺少线程和异常处理等功能），然后用这个受限的编译器构建 glibc（glibc 本身不受限），同时构建 libstdc++。最后这个库将缺少 libgcc 的部分功能。

前文所述的核心在于，cc1 编译器无法使用功能降级的 libgcc 构建出完全可用的 libstdc++ 库，但在第二阶段构建 C/C++ 库时，cc1 却是唯一可用的编译器。我们不立即使用第二阶段构建的 cc-lfs 编译器来构建这些库，主要基于两个原因。

- 一般而言，cc-lfs 无法在 pc（宿主系统）上运行。尽管 pc 和 lfs 的三元组彼此兼容，但针对 lfs 编译的可执行文件必须依赖 glibc-2.41；而宿主系统可能使用不同的 libc 实现（例如 musl），或是旧版 glibc（例如 glibc-2.13）。
- 即便 cc-lfs 能在 pc 上运行，由于它是原生编译器，在 pc 上使用仍存在链接到宿主系统库的风险。

因此，当我们构建 gcc 第二阶段时，我们会指示构建系统使用 cc1 重新构建 libgcc 和 libstdc++，但我们将新构建的 libstdc++ 链接到新重建的 libgcc，而非旧的、性能下降的版本。这使得重新构建的 libstdc++ 功能完整。

在第 8 章（或称为“阶段 3”）中，会构建 LFS 系统所需的所有软件包。即使某个软件包在前面的章节中已经安装到 LFS 系统中，我们仍会重新构建它。重新构建这些软件包的主要目的是确保其稳定性：如果我们在一个已完成的 LFS 系统上重新安装某个 LFS 软件包，重新安装的内容应与第 8 章首次安装时的内容相同。第 6 章或第 7 章中安装的临时软件包无法满足这一要求，因为其中一些软件包在构建时没有包含可选依赖项，而且由于交叉编译的原因，autoconf 在第 6 章无法执行某些特性检查，导致临时软件包缺少可选功能或使用了次优的代码例程。此外，重新构建软件包的一个次要原因是运行测试套件。

其他程序细节

交叉编译器将被安装在一个独立的 \$LFS/tools 目录中，因为它不会成为最终系统的一部分。

首先安装 Binutils，因为无论是 gcc 还是 glibc 的配置过程都会对汇编器和链接器进行各种功能测试，以确定启用或禁用哪些软件特性。这一点的重要性可能远超最初的认知。错误配置的 gcc 或 glibc 可能导致工具链存在难以察觉的缺陷，而这些缺陷的影响可能要到整个发行版构建接近尾声时才会显现。测试套件的失败通常能在进行过多额外工作之前凸显这类错误。

Binutils 将其汇编器和链接器安装在两个位置：\$LFS/tools/bin 和 \$LFS/tools/\$LFS_TGT/bin。其中一处的工具会硬链接到另一处。链接器的一个重要特性是其库搜索顺序。通过向 ld 传递--verbose 标志可以获取详细信息。例如，执行 \$LFS_TGT-ld --verbose | grep SEARCH 将显示当前搜索路径及其顺序。（注意此示例仅在以 lfs 用户身份登录时才能直接运行。若后续返回查看本页，需将 \$LFS_TGT-ld 替换为 ld）

接下来安装的软件包是 gcc。运行 configure 时可能看到的示例如下：

```
检查使用何种汇编器... /mnt/lfs/tools/i686-lfs-linux-gnu/bin/as 检查使用何种链接器...
/mnt/lfs/tools/i686-lfs-linux-gnu/bin/ld
```

基于上述原因，这一点至关重要。同时也表明 gcc 的配置脚本不会通过搜索 PATH 目录来确定使用哪些工具。然而在实际运行 gcc 时，并不一定会采用相同的搜索路径。要查看 gcc 将使用哪个标准链接器，可执行命令：

\$LFS_TGT-gcc -print-prog-name=ld。（注意：若后续再次使用该命令，需移除 \$LFS_TGT- 前缀。）

在编译程序时通过向 gcc 传递 -v 命令行选项可获取详细信息。例如执行 \$LFS_TGT-gcc -v example.c（若后续操作则无需加 \$LFS_TGT- 前缀），将显示预处理、编译和汇编阶段的详细信息，包括 gcc 搜索头文件的路径及其顺序。

接下来是经过净化的 Linux API 头文件。这些头文件使得标准 C 库（glibc）能够与 Linux 内核提供的功能进行交互。

接下来是 glibc。构建 glibc 时最重要的考量因素是编译器、二进制工具和内核头文件。编译器和二进制工具通常不是问题，因为 glibc 始终会使用与 -- 相关的那些工具。

传递给其配置脚本的 host 参数；例如，在我们的案例中，编译器将是 \$LFS_TGT-gcc，而 readelf 工具将是 \$LFS_TGT-readelf。内核头文件可能会稍微复杂一些。因此，我们采取零风险策略，使用可用的配置开关来强制正确选择。运行配置脚本后，请检查

构建目录中的 config.make 文件包含了所有重要细节。这些内容凸显了 glibc 软件包的一个重要特性——其构建机制高度自给自足，通常不依赖工具链的默认设置。

如上所述，接下来将编译标准 C++ 库，随后在第 6 章中交叉编译其他程序以解决构建时的循环依赖问题。所有这些软件包的安装步骤都使用 DESTDIR 变量强制安装到 LFS 文件系统中。

在第六章结束时，原生 LFS 编译器已安装完毕。首先构建 binutils-pass2，与其他程序存放在相同的 DESTDIR 目录中，随后构建 gcc 的第二阶段编译，省略部分非关键库文件。由于 gcc 配置脚本中存在特殊逻辑，当宿主系统与目标系统相同时（但不同于构建系统），CC_FOR_TARGET 变量最终会被设为 cc。因此需要在配置选项中显式声明 CC_FOR_TARGET=\$LFS_TGT-gcc。

进入第 7 章的 chroot 环境后，将安装工具链正常运行所需的临时程序。从此刻起，核心工具链便实现了自给自足与自主托管。

在第八章中，将构建、测试并安装一个功能完备系统所需的所有软件包的最终版本。

通用编译说明

注意

在 LFS 的开发周期中，手册中的指令常会因软件包更新或利用新特性而进行调整。混用不同版本 LFS 手册的指令可能导致难以察觉的故障。这类问题通常源于复用为旧版 LFS 创建的脚本，我们强烈不建议这种做法。若因任何原因必须复用旧版脚本，请务必仔细对照当前 LFS 手册版本更新脚本内容。

以下是构建每个软件包时需要注意的事项：

- 部分软件包在编译前需要打补丁，但仅限于需要规避特定问题时才进行补丁操作。这些补丁通常在本章及后续章节都会用到，但有时同一个软件包会多次编译，此时补丁可能不会立即使用。因此，若发现某些下载补丁的操作说明缺失，无需担心。应用补丁时可能还会遇到关于偏移量或模糊匹配的警告信息，这些警告可忽略不计，补丁实际上已成功应用。
- 在编译大多数软件包时，屏幕上会滚动显示一些警告信息。这些属于正常现象，可安全忽略。警告通常涉及 C 或 C++ 语法中已弃用但仍有效的用法。由于 C 语言标准更新频繁，部分软件包尚未同步调整。这不会造成严重问题，但确实会触发警告提示。
- 最后一次检查 LFS 环境变量是否设置正确：

```
echo $LFS
```

确保输出显示 LFS 分区挂载点的路径，在我们的示例中为 /mnt/lfs。

- 最后，必须强调两个重要事项：

重要提示

编译说明基于以下前提：宿主系统（包括符号链接）已按要求正确配置：

- bash 是当前使用的 shell 程序。
- 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，或者指向一个执行 bison 的小脚本。

重要提示

以下是构建过程的概要说明。

1. 将所有源码包和补丁文件存放在 chroot 环境可访问的目录中，例如 /mnt/lfs/sources/。
2. 切换到 /mnt/lfs/sources/ 目录。3. 对每个软件包执行以下操作：
 - a. 使用 tar 程序解压待构建的软件包。注意在第五章和第六章中，解压时必须确保当前用户是 lfs 用户。

提取源代码时请仅使用 tar 命令，切勿使用其他方法。特别要注意的是，使用 cp -R 命令将源代码树复制到其他位置可能会破坏源代码树中的时间戳，从而导致构建失败。

- b. 切换到解压软件包时创建的目录。
- c. 按照软件包构建说明进行操作。
- d. 构建完成后切换回源代码目录。e. 除非另有说明，否则请删除已解压的源代码目录。

第 5 章 构建交叉工具链

5.1. 简介

本章将展示如何构建交叉编译器及其相关工具。虽然这里采用的是模拟交叉编译的方式，但其原理与真实的交叉工具链完全相同。

本章编译的程序将安装在\$LFS/tools 目录下，以便与后续章节安装的文件区分开来。而库文件则会安装到它们的最终位置，因为这些库属于我们要构建的目标系统。

5.2. Binutils-2.44 - 第 1 遍编译

Binutils 软件包包含链接器、汇编器以及其他用于处理目标文件的工具。

预计编译时间：1 SBU

所需磁盘空间： 677 MB

5.2.1. 交叉编译 Binutils 的安装说明

请返回重新阅读标题为"通用编译说明"章节中的注意事项。理解标有"重要"的注释能为您后续避免许多问题。

必须首先编译 Binutils 软件包，因为 Glibc 和 GCC 都会对现有链接器和汇编器进行多项测试，以确定启用它们自身的哪些功能特性。

Binutils 文档建议在专用构建目录中进行编译：

```
mkdir -v build
进入目录 构建
```

注意

为了使本书后续列出的 SBU(标准构建单位)值具有参考意义，请测量从配置开始直至首次安装完成所耗费的时间。为方便测量，建议将

像这样在 time 命令中执行多个命令：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：

--prefix=\$LFS/tools

该选项指示配置脚本准备将 Binutils 程序安装到\$LFS/tools 目录中。

--with-sysroot=\$LFS

对于交叉编译，这会告知构建系统在需要时到\$LFS 中寻找目标系统的库文件。

--target=\$LFS_TGT

由于 LFS_TGT 变量中的机器描述与 config.guess 脚本返回的值略有不同，该选项将指示配置脚本调整 binutils 的构建系统以构建交叉链接器。

--disable-nls

这将禁用国际化功能，因为临时工具不需要 i18n 支持。

--enable-gprofng=no

这将禁用 gprofng 的构建，因为临时工具不需要该组件。

--disable-werror

该选项可防止在宿主编译器出现警告时中断构建过程。

--enable-new-dtags

该选项使链接器使用"runpath"标签而非传统的"rpath"标签，将库搜索路径嵌入可执行文件和共享库中。这既便于调试动态链接的可执行程序，又能规避某些软件包测试套件中的潜在问题。

--enable-default-hash-style-gnu

默认情况下，链接器会为共享库和动态链接可执行文件同时生成 GNU 风格哈希表和经典 ELF 哈希表。这些哈希表仅供动态链接器执行符号查找使用。在 LFS 系统中，动态链接器（由 Glibc 软件包提供）始终会使用查询速度更快的 GNU 风格哈希表，因此经典 ELF 哈希表完全无用。该选项使链接器默认仅生成 GNU 风格哈希表，这样既能避免软件包构建时生成经典 ELF 哈希表的时间浪费，也能节省存储空间。

继续编译该软件包：

make

安装该软件包：

make install

该软件包的详细信息请参阅第 8.20.2 节"Binutils 的内容"。

5.3. GCC-14.2.0 - 第 1 遍编译

GCC 软件包包含 GNU 编译器集合，其中涵盖 C 和 C++ 编译器。

预计编译时间：3.2 SBU

所需磁盘空间：4.8 GB

5.3.1. 交叉工具链 GCC 的安装 GCC 需要 GMP、MPFR 和 MPC 软件包。由于宿主系统可能未包含这些软件包，它们将与 GCC 一同编译。请将这些软件包解压至 GCC 源码目录，并重命名生成的目录

以便 GCC 编译流程自动识别它们：

注意

关于本章节存在一些常见误解。其操作流程与其他章节完全相同，正如先前所述（软件包构建说明）。首先从源码目录解压 gcc-14.2.0 的压缩包，然后进入解压生成的目录。完成这些步骤后，方可继续执行以下指令。

```
tar -xf ./mpfr-4.2.1.tar.xz mv -v
mpfr-4.2.1 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
```

在 x86_64 架构主机上，将 64 位库的默认目录名设置为 "lib"：

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

GCC 文档建议在专用构建目录中编译 GCC：

```
mkdir -v build
进入目录 构建
```

为 GCC 编译做好准备：

```
./configure \
--target=$LFS_TGT \
--prefix=$LFS/tools \
--with-glibc-
version=2.41 \
--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=2.41

此选项指定将在目标系统上使用的 Glibc 版本。它与宿主发行版的 libc 无关，因为由 pass1 GCC 编译的所有内容都将在 chroot 环境中运行，该环境与宿主发行版的 libc 隔离。

--with-newlib

由于可用的 C 语言库尚未就绪，这一步骤确保在构建 libgcc 时定义了 inhibit_libc 常量。此举能防止编译任何需要 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。就我们的需求而言，这些头文件并非必需。此选项可阻止 GCC 搜索这些头文件。

--enable-default-pie 和 --enable-default-ssp

这些选项使 GCC 默认以具备某些安全加固特性的方式编译程序（关于 PIE 和 SSP 的详细信息可参阅第 8 章的说明）。虽然在此阶段并非绝对必要（因为编译器仅生成临时可执行文件），但让临时软件包尽可能接近最终形态是更规范的做法。

--disable-shared

该选项强制 GCC 以静态方式链接其内部库。由于共享库需要 Glibc 支持，而目标系统尚未安装 Glibc，因此必须采用此设置。

--disable-multilib

在 x86_64 架构上，LFS 不支持 multilib 配置。该选项对 x86 架构无害。

--disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --
--disable-libvtv, --disable-libstdcxx

这些选项分别用于禁用线程支持、libatomic 库、libgomp 库、libquadmath 库、libssp 库、libvtv 库以及 C++ 标准库。在构建交叉编译器时，这些功能可能导致编译失败，且对于交叉编译临时 libc 的任务并非必需。

--enable-languages=c, c++

此选项确保仅构建 C 和 C++ 编译器。这是目前唯一需要的语言。

运行以下命令编译 GCC:

```
make
```

安装该软件包:

```
make install
```

本次构建的 GCC 安装了几个内部系统头文件。通常其中之一的 limits.h 会包含对应的系统 limits.h 头文件（即 \$LFS/usr/include/limits.h）。但在本次 GCC 构建时，\$LFS/usr/include/limits.h 尚不存在，因此刚安装的内部头文件是一个不完整的独立文件，未包含系统头文件的扩展功能。这对于构建 Glibc 已足够，但后续将需要完整的内部头文件。使用与 GCC 构建系统在常规情况下相同的命令创建完整版本的内部头文件：

注意

以下命令展示了使用两种方法（反引号和\$()结构）进行嵌套命令替换的示例。虽然可以使用相同方法重写这两个替换，但这里特意混合使用以展示其组合方式。通常推荐使用\$()方法。

```
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参见第 8.29.2 节“GCC 的组成内容”。

5.4. Linux-6.13.4 API 头文件

Linux API 头文件(位于 `linux-6.13.4.tar.xz`)向 Glibc 暴露内核应用程序接口。

预计编译时间：少于 0.1 SBU

所需磁盘空间：1.6 GB

5.4.1. Linux API 头文件安装

Linux 内核需要向系统的 C 库(LFS 中使用 Glibc)暴露应用程序接口(API)。这是通过清理 Linux 内核源码包中的各种 C 头文件来实现的。

确保软件包中没有残留的旧文件：

```
make mrproper
```

现在从源码中提取用户可见的内核头文件。由于需要 rsync (可能未安装)，无法使用推荐的"headers_install"编译目标。头文件会先存放在`./usr`目录，随后复制到指定位置。

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

5.4.2. Linux API 头文件安装内容 已安装的头文件：

已安装的目录：
`/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/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/xen`

简要描述

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

5.5. Glibc-2.41 Glibc 软件包包含主要的 C 语言库。该库提供了内存分配、目录搜索、文件开关、读写操作、字符串处理、模式匹配、算术运算等基础例程。

预计构建时间：1.4 SBU

所需磁盘空间：850 MB

5.5.1. Glibc 的安装

首先，为符合 LSB 标准创建一个符号链接。此外，对于 x86_64 架构，还需创建必需的兼容性符号链接。

确保动态库加载器正常运行：

```
case $(uname -m) in
i?86) ln -sfv ld-linux.so.2 $LFS/lib/ld-1sb.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-1sb-x86-64.so.3
;;
esac
```

注意

上述命令是正确的。ln 命令存在多种语法版本，因此在报告看似错误的情况下，请务必查阅 info coreutils ln 和 ln(1) 手册。

部分 Glibc 程序会使用不符合 FHS 标准的 /var/db 目录存储运行时数据。应用以下补丁可使这些程序将运行时数据存储在符合 FHS 标准的位置：

```
patch -Np1 -i ../glibc-2.41-fhs-1.patch
```

Glibc 文档建议在专用构建目录中编译 Glibc：

```
mkdir -v build
进入目录 构建
```

确保将 ldconfig 和 sln 工具安装到 /usr/sbin 目录下：

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

接下来，准备编译 Glibc：

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

配置选项的含义：

```
--host=$LFS_TGT, --build=$(./scripts/config.guess)
```

这些开关的组合作用是让 Glibc 的构建系统配置为交叉编译模式，使用 \$LFS/tools 目录中的交叉链接器和交叉编译器。

```
--enable-kernel=5.4
```

该参数指示 Glibc 编译支持 5.4 及以上版本 Linux 内核的库文件，同时禁用对旧版内核的兼容性处理。

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

这指示 Glibc 根据最近安装到\$LFS/usr/include 目录的头文件进行编译，使其能准确了解内核特性并据此进行优化。

```
libc_cv_slibdir=/usr/lib
```

这确保库文件会被安装到/usr/lib 目录，而非 64 位机器默认的/lib64 目录。

```
--disable-nscd
```

不构建已不再使用的名称服务缓存守护进程。

在此阶段可能会出现以下警告：

```
configure: 警告: *** 这些辅助程序缺失或版本不兼容: msgfmt
*** 部分功能将被禁用。
```

*** 请查阅 INSTALL 文件获取所需版本信息。

缺失或不兼容的 msgfmt 程序通常不会造成严重影响。该 msgfmt 程序属于 Gettext 软件包，应由宿主发行版提供。

注意

有报告称此软件包在进行"并行编译"时可能失败。若出现此情况，请使用-j1 选项重新运行 make 命令。

编译该软件包：

编译

安装该软件包：

警告

如果 LFS 环境未正确设置，且不顾建议仍以 root 身份进行构建，下一条命令将会把新编译的 Glibc 安装到宿主系统，这几乎必然导致系统无法使用。因此在执行后续命令前，请务必再次确认环境配置正确且未使用 root 权限。

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

make install 选项的含义：

```
DESTDIR=$LFS
```

DESTDIR 这个 make 变量被几乎所有软件包用来定义安装位置。若未设置该变量，默认会安装到根目录(/)下。此处我们指定将软件包安装至\$LFS 目录，该目录将在第 7.4 节"进入 Chroot 环境"中成为根目录。

修复 ldd 脚本中对可执行加载器的硬编码路径：

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

注意

此时必须暂停操作，确保新工具链的基本功能（编译和链接）能按预期工作。为进行完整性检查，请运行以下命令：

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

若一切正常，应无报错信息，且最后一条命令的输出格式应为：

```
[请求的程序解释器: /lib64/ld-linux-x86-64.so.2]
```

请注意，对于 32 位机器，解释器名称应为/lib/ld-linux.so.2。

如果输出与上述不符，或完全没有输出，则表明存在问题。请检查并回溯操作步骤，找出问题所在并进行修正。必须解决此问题后才能继续后续操作。

确认一切正常后，清理测试文件：

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

注意

下一章中构建软件包将作为工具链是否正确构建的额外验证。如果某些软件包（特别是 Binutils 第二遍或 GCC 第二遍）构建失败，则表明之前的 Binutils、GCC 或 Glibc 安装环节出现了问题。

该软件包的详细信息见第 8.5.3 节“Glibc 的内容”。

5.6. GCC-14.2.0 中的 Libstdc++ Libstdc++ 是标准 C++ 库。编译 C++ 代码时需要它（GCC 的部分代码是用 C++ 编写的），但在构建 gcc-pass1 时我们不得不推迟其安装，因为 Libstdc++ 依赖于 Glibc，而目标目录中当时尚未安装 Glibc。

预计构建时间：0.2 SBU

所需磁盘空间：850 MB

5.6.1. 目标 Libstdc++ 的安装说明

Libstdc++ 是 GCC 源码的一部分。您需要先解压 GCC 的压缩包并进入 gcc-14.2.0 目录

为 Libstdc++ 创建一个单独的构建目录并进入：

```
mkdir -v build
cd      构建
```

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++/14.2.0
```

配置选项的含义：

--host=...

指定应使用我们刚刚构建的交叉编译器，而非 /usr/bin 目录下的编译器。

--disable-libstdcxx-pch

此选项可防止安装预编译的头文件，当前阶段并不需要这些文件。

--with-gxx-include-dir=/tools/\$LFS_TGT/include/c++/14.2.0

该参数指定了头文件的安装目录。由于 Libstdc++ 是 LFS 的标准 C++ 库，此目录应与 C++ 编译器 (\$LFS_TGT-g++) 搜索标准 C++ 头文件的位置相匹配。在常规构建中，这些信息会从顶层目录自动传递给 Libstdc++ 的配置选项。而在我们的场景中，必须显式指定这些信息。C++ 编译器会在头文件搜索路径前添加系统根路径 \$LFS（在构建 GCC-pass1 时指定），因此实际会搜索 \$LFS/tools/\$LFS_TGT/include/c++/14.2.0 目录。下方 make install 命令中的 DESTDIR 变量与此选项共同作用，确保头文件被安装到正确位置。

运行以下命令编译 Libstdc++：

编译

安装该库：

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

删除这些 libtool 归档文件，因为它们对交叉编译有害：

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

该软件包的详细信息参见第 8.29.2 节“GCC 的组成内容”。

第 6 章 交叉编译临时工具

6.1. 简介

本章将展示如何使用刚刚构建的交叉工具链来交叉编译基础工具程序。这些工具会被安装到最终位置，但暂时还无法使用。基础任务仍需依赖宿主系统的工具。不过，在链接时会使用已安装的库文件。

进入"chroot"环境后，下一章即可使用这些工具程序。但本章构建的所有软件包都必须在此之前完成编译。因此现阶段我们仍无法完全脱离宿主系统。

再次提醒：若 LFS 环境设置不当，同时以 root 身份进行构建，可能导致计算机无法正常使用。本章所有操作必须按照 4.4 节"设置环境"中的描述，以 lfs 用户身份在配置好的环境下完成。

6.2. M4-1.4.19 M4 软件包包含一个宏处理器。

预计编译时间：0.1 SBU 所需磁盘空间：32 MB

6.2.1. M4 的安装 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位于第 8.13.2 节 "M4 的内容" 中。

6.3. Ncurses-6.5 Ncurses 软件包包含用于终端无关字符屏幕处理的函数库。

预计编译时间：0.4 SBU 所需磁盘空间：53 MB

6.3.1. 安装 Ncurses 首先执行以下命令在构建主机上编译“tic”程序：

```
mkdir build
pushd build ../configure
AWK=gawk make -C include make -C
progs tic 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
```

新增配置选项的含义：

--with-manpage-format=normal

该选项防止 Ncurses 安装压缩版手册页（当宿主发行版自身使用压缩版手册页时可能出现此情况）。

--with-shared

该选项使 Ncurses 构建并安装共享 C 库。

--without-normal

该选项阻止 Ncurses 构建并安装静态 C 库。

--without-debug

该选项会阻止 Ncurses 构建和安装调试库。

--with-cxx-shared

这使得 Ncurses 构建并安装共享的 C++ 绑定库，同时避免构建和安装静态的 C++ 绑定库。

--without-ada

该选项确保 Ncurses 不会构建对 Ada 编译器的支持（主机系统可能装有该编译器，但进入 chroot 环境后将不可用）。

--disable-stripping

此选项阻止构建系统使用宿主机的 strip 程序。在交叉编译的程序上使用宿主工具可能导致失败。

AWK=gawk

该选项可防止构建系统使用宿主机的 mawk 程序。某些版本的 mawk 可能导致此软件包构建失败。

编译该软件包：

make

安装该软件包：

```
make DESTDIR=$LFS TIC_PATH=$(pwd)/build/progs/tic install ln -sv
libncursesw.so $LFS/usr/lib/libcurses.so sed -e 's/^#if.*XOPEN.*$/#if 1/' \
-i $LFS/usr/include/curses.h
```

安装选项的含义：

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

我们需要传递在构建机器上新构建的 tic 程序路径，这样终端数据库才能无误地创建。

ln -sv libncursesw.so \$LFS/usr/lib/libcurses.so

我们即将构建的几个软件包需要 libcurses.so 库。我们创建这个符号链接来使用 libncursesw.so 作为替代。

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

头文件 curses.h 包含了各种 Ncurses 数据结构的定义。通过不同的预处理器宏定义，可以使用两种不同的数据结构定义集：8 位定义与 libcurses.so 兼容，而宽字符定义则与 libncursesw.so 兼容。由于我们正在使用 libncursesw.so 作为 libcurses.so 的替代品，请编辑头文件使其始终使用与 libncursesw.so 兼容的宽字符数据结构定义。

该软件包的详细信息参见第 8.30.2 节" Ncurses 内容"。

6.4. Bash-5.2.37 Bash 软件包包含 Bourne-Again Shell。

预计构建时间：0.2 SBU 所需磁盘空间：68 MB

6.4.1. Bash 的安装 准备编译 Bash：

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

配置选项的含义：

--without-bash-malloc

该选项会禁用 Bash 的内存分配 (malloc) 功能，该功能已知会导致段错误。关闭此选项后，Bash 将使用更稳定的 Glibc 内存分配函数。

编译该软件包：

```
make
```

安装该软件包：

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

为使用 sh 作为 shell 的程序创建链接：

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

该软件包的详细信息参见第 8.36.2 节 "Bash 软件包内容"。

6.5. Coreutils-9.6

Coreutils 软件包包含每个操作系统都需要的基础工具程序。

预计编译时间: 0.3 SBU

所需磁盘空间: 181 MB

6.5.1. Coreutils 的安装准备

为编译 Coreutils 做好准备:

```
./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

该选项允许构建并安装 hostname 二进制程序——该程序默认被禁用，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预期的位置。虽然在这个临时环境中并非必要，但我们必须这样做，因为某些程序会硬编码可执行文件路径:

```
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见第 8.58.2 节"Coreutils 内容说明"。

6.6. Diffutils-3.11

Diffutils 软件包包含用于显示文件或目录差异的程序。

预计构建时间: 0.1 SBU

所需磁盘空间: 35 MB

6.6.1. Diffutils 的安装

准备编译 Diffutils:

```
./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见第 8.60.2 节 "Diffutils 的内容"。

6.7. File-5.46 File 软件包包含一个用于确定给定文件类型的实用工具。

预计构建时间: 0.1 SBU

所需磁盘空间: 42 MB

6.7.1. 安装 File 命令

构建主机上的 file 命令需要与我们正在构建的版本相同, 以便创建

签名文件。运行以下命令创建文件命令的临时副本:

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

```
make
弹出目录
```

新配置选项的含义:

--disable-*

配置脚本会尝试使用来自宿主发行版的某些软件包 (如果对应的库文件存在)。如果库文件存在但对应的头文件缺失, 则可能导致编译失败。这些选项可防止使用宿主系统中这些不必要的功能。

准备编译文件:

```
./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 libtool 归档文件, 因其对交叉编译有害:

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

该软件包的详细信息见第 8.11.2 节"文件内容"。

6.8. Findutils-4.10.0 Findutils 软件包提供查找文件的程序集。这些程序能够遍历目录树中的所有文件，并支持创建、维护及查询文件数据库（通常比递归查找更快，但若数据库未及时更新则可能不可靠）。Findutils 还提供 xargs 程序，可对搜索选中的每个文件执行指定命令。

预计编译时间：0.2 SBU 所需磁盘空间：48 MB

6.8.1. Findutils 的安装 为编译 Findutils 做准备：

```
./configure --prefix=/usr \  
           --statedir=/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参见第 8.62.2 节"Findutils 软件包内容"。

6.9. Gawk-5.3.1 Gawk 软件包包含用于处理文本文件的程序。

预计编译时间：0.1 SBU 所需磁盘空间：
间：47 MB

6.9.1. Gawk 的安装 首先，确保不会安装一些不必要的文件：

```
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见第 8.61.2 节 “Gawk 内容”。

6.10. Grep-3.11 Grep 软件包包含用于搜索文件内容的程序。

预计构建时间: 0.1 SBU 所需磁盘空间: 27 MB

6.10.1. Grep 的安装 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
```

关于此软件包的详细信息请参阅第 8.35.2 节"Grep 内容说明"。

6.11. Gzip-1.13 Gzip 软件包包含用于文件压缩和解压缩的程序。

预计编译时间：0.1 SBU 所需磁盘空
间：11 MB

6.11.1. Gzip 的安装 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见第 8.65.2 节"Gzip 内容"。

6.12. Make-4.4.1 Make 软件包包含一个用于控制从源文件生成可执行文件及其他非源代码文件的程序。

预计编译时间：少于 0.1 SBU

所需磁盘空间：15 MB

6.12.1. Make 的安装准备 为编译准备 Make：

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

新配置选项的含义：

--without-guile

尽管我们正在进行交叉编译，但如果配置程序在构建主机上发现 guile，它会尝试使用该组件。这将导致编译失败，因此该开关用于阻止使用 guile。

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见第 8.69.2 节"Make 的内容"。

6.13. Patch-2.7.6 Patch 软件包包含一个通过应用"补丁"文件（通常由 diff 程序生成）来修改或创建文件的程序。

预计编译时间：0.1 SBU 所需磁盘空间：
间：12 MB

6.13.1. Patch 的安装 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参见第 8.70.2 节"补丁内容"。

6.14. Sed-4.9 Sed 软件包包含一个流编辑器。

预计编译时间：0.1 SBU 所需磁盘空间：
间：21 MB

6.14.1. Sed 的安装 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见第 8.31.2 节 "Sed 的内容"。

6.15. Tar-1.35 Tar 软件包提供了创建 tar 归档文件的功能，并能执行多种其他类型的归档操作。Tar 可用于从已创建的归档文件中提取文件、存储额外文件，或更新/列出已存储的文件。

预计编译时间：0.1 SBU 所需磁盘空

间：42 MB

6.15.1. Tar 的安装 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请参阅第 8.71.2 节“Tar 包内容”。

6.16. Xz-5.6.4 Xz 软件包包含用于压缩和解压文件的程序。它支持 lzma 和更新的 xz 压缩格式。使用 xz 压缩文本文件能获得比传统 gzip 或 bzip2 命令更好的压缩率。

预计编译时间：0.1 SBU

所需磁盘空间：21 MB

6.16.1. Xz 的安装 准备编译 Xz:

```
./configure --prefix=/usr           \
--host=$LFS_TGT \ --build=$(build-
aux/config.guess) \ --disable-static \ --
docdir=/usr/share/doc/xz-5.6.4
```

编译该软件包：

编译

安装软件包：

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

移除 libtool 归档文件，因其对交叉编译有害：

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

该软件包的详细信息参见第 8.8.2 节 "Xz 软件包内容"。

6.17. Binutils-2.44 - 第 2 遍构建 Binutils 软件包

包含链接器、汇编器以及其他处理目标文件的工具。

预计编译时间：0.4 SBU 所需磁盘空间：539 MB

6.17.1. Binutils 的安装

Binutils 构建系统依赖自带的 libtool 副本来链接内部静态库，但软件包自带的 liberty 和 zlib 副本并未使用 libtool。这种不一致性可能导致生成的二进制文件错误地链接

到宿主发行版的库文件。通过以下方式规避该问题：

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

再次创建独立的构建目录：

创建构建目录并显示详细信息：
进入 构建目录

为编译准备 Binutils 工具链：

```
./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
```

新配置选项的含义：

--enable-shared

构建 libbfd 为共享库。

--enable-64-bit-bfd

启用 64 位支持（适用于字长较小的主机）。64 位系统可能不需要此选项，但启用也无妨。

编译该软件包：

编译

安装软件包：

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

移除可能对交叉编译有害的 libtool 归档文件，并删除不必要的静态库：

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

该软件包的详细信息参见第 8.20.2 节“Binutils 的内容”。

6.18. GCC-14.2.0 - 第二次编译

GCC 软件包包含 GNU 编译器集合，其中涵盖 C 和 C++ 编译器。

预计编译时间：4.1 SBU 所需磁盘空间：5.5 GB

6.18.1. GCC 的安装

与首次编译 GCC 时相同，需要 GMP、MPFR 和 MPC 软件包。解压这些压缩包并将它们

移动到指定目录：

解压并重命名 mpfr 库：

```
tar -xf ./mpfr-4.2.1.tar.xz
mv -v mpfr-4.2.1 mpfr
```

解压并重命名 gmp 库：

```
tar -xf ./gmp-6.3.0.tar.xz
^ ^ ^
```

若在 x86_64 架构上构建，需将 64 位库的默认目录名改为“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，以支持使用 POSIX 线程构建这些库：

```
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 之前，请记得取消所有会覆盖默认优化标志的环境变量设置。

现在准备编译 GCC：

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

新增配置选项的含义：

--with-build-sysroot=\$LFS

通常使用--host 参数能确保使用交叉编译器构建 GCC，该编译器知道应在\$LFS 中查找头文件和库。但 GCC 的构建系统会使用其他工具，这些工具并不知晓该路径。此开关的作用是让这些工具能在\$LFS 而非宿主机上找到所需文件。

--target=\$LFS_TGT

我们正在进行 GCC 的交叉编译，因此无法用当前阶段编译的 GCC 二进制文件（这些二进制文件无法在宿主机上运行）来构建目标库（libgcc 和 libstdc++）。默认情况下，GCC 构建系统会尝试使用宿主机的 C 和 C++ 编译器作为临时解决方案。但官方不支持使用不同版本的 GCC 来构建目标库，因此使用宿主机编译器可能导致构建失败。此参数确保这些库由 GCC 第一阶段构建完成。

LDFLAGS_FOR_TARGET=...

允许 libstdc++ 使用当前阶段构建的 libgcc，而非 gcc-pass1 阶段构建的旧版本。由于旧版本在构建时未启用 libc 支持，因此无法正确支持 C++ 异常处理机制。

--disable-libsanitizer

禁用 GCC 的 sanitizer 运行时库。临时安装阶段不需要这些组件。在 gcc-pass1 阶段通过--disable-libstdcxx 参数已隐含禁用，现在我们可以显式传递该参数。

编译软件包：

make

安装软件包：

make DESTDIR=\$LFS install

作为收尾工作，创建一个实用符号链接。许多程序和脚本会调用 cc 而非 gcc，这种设计是为了保持程序的通用性，使其能在各种 UNIX 系统上运行——毕竟并非所有系统都安装了 GNU C 编译器。通过保留 cc 调用方式，系统管理员可以自由决定安装哪种 C 编译器：

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

该软件包的详细信息参见第 8.29.2 节“GCC 内容说明”。

第 7 章 进入 Chroot 环境并构建额外的临时工具

7.1 简介 本章将展示如何构建临时系统中最后缺失的部分：用于构建各类软件包所需的工具。既然所有循环依赖已解决，现在可以使用完全隔离于宿主操作系统（除正在运行的内核外）的“chroot”环境进行构建。

为确保隔离环境的正常运行，必须与运行中的内核建立通信机制。这是通过所谓的虚拟内核文件系统实现的，这些文件系统将在进入 chroot 环境前完成挂载。您可以通过执行 `findmnt` 命令来验证它们是否已挂载。

在进入第 7.4 节“进入 Chroot 环境”之前，所有命令都必须在设置好 LFS 变量的情况下以 root 身份运行。进入 chroot 后，所有命令同样以 root 权限执行——幸运的是，此时将无法访问您构建 LFS 的计算机操作系统。但仍需谨慎操作，因为错误的命令很容易摧毁整个 LFS 系统。

7.2. 变更所有权注意事项

本书后续章节的所有命令都必须以 root 用户身份执行，而不再使用 lfs 用户。同时，请再次确认 root 环境中已正确设置\$LFS 变量。

目前，\$LFS 下的整个目录层次结构由用户 lfs 拥有，而该用户仅存在于宿主系统中。如果 \$LFS 下的目录和文件保持现状，它们将由一个没有对应账户的用户 ID 拥有。这存在安全隐患，因为后续创建的用户账户可能会获得相同的用户 ID，从而拥有 \$LFS 下的所有文件，导致这些文件面临潜在的恶意操作风险。

为解决此问题，请运行以下命令将 \$LFS/* 目录的所有权更改为 root 用户：

```
chown --from lfs -R root:root $LFS/{usr,lib,var,etc,bin,sbin,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

运行在用户空间的应用程序需要通过内核创建的各类文件系统与内核本身通信。这些文件系统是虚拟的：它们不占用磁盘空间，其内容驻留在内存中。必须将这些文件系统挂载到 \$LFS 目录树中，以便应用程序在 chroot 环境中能够访问它们。

首先创建这些虚拟文件系统将要挂载的目录：

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

7.3.1. 挂载并填充/dev 目录

在 LFS 系统正常启动时，内核会自动将 devtmpfs 文件系统挂载到 /dev 目录；内核在启动过程中或首次检测/访问设备时，会在该虚拟文件系统上创建设备节点。udev 守护进程可能会修改内核创建的设备节点的所有者或权限，

并创建新的设备节点或符号链接，以简化发行版维护者和系统管理员的工作。（详见第 9.3.2.2 节“设备节点创建”）。如果主机内核支持 devtmpfs，我们可以直接挂载一个在\$LFS/dev 挂载 devtmpfs，并依赖内核来填充它。

但某些主机内核缺乏 devtmpfs 支持；这些主机发行版使用不同方法来创建/dev 目录内容。因此填充\$LFS/dev 目录的唯一与主机无关的方法，是通过绑定挂载主机系统的/dev 目录。绑定挂载是一种特殊挂载类型，它使得目录子树或文件在其他位置可见。使用以下命令实现此操作。

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

7.3.2. 挂载虚拟内核文件系统 现在挂载剩余的虚拟内核文件系统：

```
挂载 -vt devpts devpts -o gid=5,mode=0620 $LFS/dev/pts
挂载 -vt proc proc $LFS/proc
挂载 -vt sysfs sysfs $LFS/sys
挂载 -vt tmpfs tmpfs $LFS/run
```

devpts 挂载选项的含义：

gid=5

这确保所有由 devpts 创建设备节点的所属组 ID 均为 5。这是我们后续将为终端*我们*使用组 ID 而非组名，因为宿主系统可能为其终端设备组分配了不同的 ID。

模式=0620

这确保所有由 devpts 创建的设备节点都具有 0620 模式（用户可读写，组可写）。结合上述选项，可保证 devpts 创建的设备节点符合 grantpt() 函数要求，意味着无需安装 Glibc 的 pt_chown 辅助程序（默认不安装）。

在某些宿主系统中，/dev/shm 是一个指向目录（通常是/run/shm）的符号链接。由于之前已经挂载了/run tmpfs，因此在这种情况下只需创建一个具有正确权限的目录即可。

而在其他宿主系统中，/dev/shm 是 tmpfs 的挂载点。此时上述/dev 的挂载操作仅会在 chroot 环境中创建/dev/shm 目录。这种情况下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 现在系统中已包含构建剩余所需工具的所有必要软件包，是时候进入 chroot 环境来完成临时工具的安装了。该环境也将用于最终系统的安装。以 root 用户身份运行以下命令进入当前已配置好的环境：

仅作为临时工具使用：

```
chroot "$LFS" /usr/bin/env -i \
    HOME=/root \
    TERM="$TERM" \ PS1='(1fs chroot) \u:\w\$'
    \ PATH=/usr/bin:/usr/sbin \
    MAKEFLAGS="-j$(nproc)" \
    TESTSUITEFLAGS="-j$(nproc)" \ /bin/bash
--login
```

若不想使用全部可用的逻辑核心, 请将\$(nproc)替换为您希望在本章及后续章节构建软件包时使用的逻辑核心数量。第 8 章中部分软件包 (特别是 Autoconf、Libtool 和 Tar) 的测试套件不受 MAKEFLAGS 影响, 它们使用 TESTSUITEFLAGS 环境变量替代。

我们在此同样设置该变量, 以便使用多核心运行这些测试套件。

传递给 env 命令的-i 选项将清除 chroot 环境中的所有变量。此后, 仅重新设置 HOME、TERM、PS1 和 PATH 变量。TERM=\$TERM 结构会将 chroot 内的 TERM 变量设置为与外部 chroot 相同的值。该变量是 vim 和 less 等程序正常运行所必需的。如需设置其他变量 (如 CFLAGS 或 CXXFLAGS), 此处是理想的设置位置。

从现在开始, 不再需要使用 LFS 变量, 因为所有工作都将限制在 LFS 文件系统中; chroot 命令会启动 Bash shell, 并将根目录(/)设置为\$LFS。

请注意/tools/bin 不在 PATH 环境变量中。这意味着交叉工具链将不再被使用。

还要注意的是 bash 提示符会显示 "I have no name!", 这是正常现象, 因为/etc/passwd 文件尚未创建。

注意

非常重要的一点是: 本章剩余部分及后续章节的所有命令都必须在 chroot 环境中运行。如果因任何原因 (例如重启) 退出了该环境, 请确保按照 7.3.1 节 "挂载并填充/dev" 和 7.3.2 节 "挂载虚拟内核文件系统" 的说明重新挂载虚拟内核文件系统, 并在继续安装前再次进入 chroot 环境。

7.5. 创建目录结构 现在是在 LFS 文件系统中建立完整目录结构的时机。

注意

本节提及的部分目录可能已通过前期明确指令或安装某些软件包时创建。为保持完整性, 此处再次列出。

通过执行以下命令, 创建前几章有限需求之外的其他根级目录:

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

通过执行以下命令在根目录下创建所需的子目录结构：

```
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
```

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

默认情况下，新建目录的权限模式为 755，但这并非适用于所有场景。上述命令中做了两处调整：一处针对 root 用户的主目录，另一处针对临时文件目录。

首次权限变更确保并非任何人都能进入/root 目录——就像普通用户对其个人主目录的设置一样。第二次权限变更确保所有用户都能向/tmp 和 /var/tmp 目录允许用户删除自己的文件，但不能删除其他用户的文件。后者被所谓的“粘滞位”（sticky bit）所禁止，即 1777 权限掩码中的最高位（1）。

7.5.1. FHS 合规性说明 本目录树基于文件系统层次结构标准(FHS)（参见 <https://refspecs.linuxfoundation.org/fhs.shtml>）。FHS 还规定了可选目录的存在，如/usr/local/games 和

/usr/share/games。在 LFS 中，我们只创建真正必要的目录。不过，如果您愿意，可以自由创建更多目录。

警告

FHS 标准并未强制要求存在/usr/lib64 目录，LFS 编辑团队决定不使用该目录。为确保 LFS 和 BLFS 中的指令能正确执行，必须确保该目录不存在。建议您定期检查该目录是否被误创建，因为该目录容易被无意创建，而这可能导致系统故障。

7.6. 创建必要文件与符号链接 历史上，Linux 系统通过/etc/mtab 文件维护已挂载文件系统列表。现代内核改为在内部维护该列表，并通过/proc 文件系统向用户展示。为兼容需要读取/etc/mtab 的工具程序，需创建

以下符号链接：

创建符号链接将/proc/self/mounts 指向/etc/mtab：
ln -sv /proc/self/mounts /etc/mtab

创建一个基础的/etc/hosts 文件，供部分测试套件及 Perl 某个配置文件引用：

使用以下内容生成/etc/hosts 文件：

```
#!/etc/hosts
localhost $(hostname)
::1 本地主机
EOF
```

为了让 root 用户能够登录且系统能识别"root"名称，必须在 /etc/passwd 和 /etc/group 文件中存在相关条目。

通过运行以下命令创建/etc/passwd 文件：

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/bin/bash
bin:x: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
uuidd:x:80:80:UUID Generation Daemon User:/dev/null:/usr/bin/false
nobody:x:65534:65534:Unprivileged User:/dev/null:/usr/bin/false
EOF
```

root 用户的实际密码将在稍后设置。

通过运行以下命令创建/etc/group 文件：

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:daemon
sys:x:2:
kmem:x:3:
磁带设备组:x:4:
终端设备组:x:5:
守护进程组:x:6:
软盘设备组:x:7:
磁盘:x:8:
打印:x:9:
拨号:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
cdrom:x:15:
adm:x:16:
messagebus:x:18:
input:x:24:
mail:x:34:
kvm:x:61:
uuidd:x:80:
wheel:x:97:
users:x:999:
nogroup:x:65534:
EOF
```

所创建的组并不属于任何标准——这些组部分是根据第 9 章中 Udev 配置的需求决定的，部分则是采用了多个现有 Linux 发行版的通用惯例。此外，某些测试套件依赖于特定的用户或组。Linux 标准规范 (LSB，详见 <https://refspecs.linuxfoundation.org/lsb.shtml>) 仅建议：除了 GID 为 0 的 root 组外，还应存在 GID 为 1 的 bin 组。GID 为 5 通常用于 tty 组，而数字 5 也被用于 /etc/fstab 中的 devpts 文件系统。所有其他组名和 GID 都可以由系统管理员自由选择，因为编写良好的程序并不依赖于 GID 编号，而是使用组的名称。

内核使用 ID 65534 来为 NFS 和独立的用户命名空间处理未映射的用户和组（这些用户和组存在于 NFS 服务器或父用户命名空间中，但在本地机器或独立命名空间中“不存在”）。我们分配 nobody 和 nogroup 以避免出现未命名的 ID。但其他发行版可能对此 ID 的处理方式不同，因此任何可移植程序都不应依赖此分配。

第八章中的某些测试需要一个普通用户。我们在此添加该用户，并在该章节末尾删除此账户。

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

要消除"I have no name!"提示，请启动一个新的 shell。由于/etc/passwd 和/etc/group 文件已创建，用户名和组名解析现在可以正常工作：

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

登录程序、agetty 和 init 程序（以及其他程序）使用多个日志文件来记录诸如谁在何时登录系统等信息。但如果这些日志文件不存在，这些程序将不会写入信息。请初始化日志文件并设置适当权限：

```
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
```

/var/log/wtmp 文件记录所有登录和注销活动。/var/log/lastlog 文件记录每位用户最近一次登录时间。/var/log/faillog 文件记录失败的登录尝试。/var/log/btmp 文件记录不良登录尝试。

注意

/run/utmp 文件记录当前登录的用户信息，该文件由启动脚本动态创建。

注意

utmp、wtmp、btmp 和 lastlog 文件使用 32 位整数存储时间戳，在 2038 年后将完全失效。许多软件包已停止使用这些文件，其他软件包也将逐步弃用。

这些文件很可能已被视为过时，建议不再使用。

7.7. Gettext-0.24 Gettext 软件包包含国际化和本地化工具。这些工具允许程序在编译时启用 NLS（本地语言支持），使其能够以用户母语输出信息。

预计构建时间：1.3 SBU

所需磁盘空间：349 MB

7.7.1. Gettext 的安装 对于我们临时的工具集，只需要安装 Gettext 中的三个程序。

准备编译 Gettext：

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

配置选项的含义：

```
--disable-shared
```

此时我们无需安装任何共享的 Gettext 库文件，因此无需构建它们。

编译该软件包：

```
make
```

安装 msgfmt、msgmerge 和 xgettext 程序：

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

该软件包的详细信息位于第 8.33.2 节"Gettext 的内容"中。

7.8. Bison-3.8.2 Bison 软件包包含一个解析器生成器。

预计编译时间：0.2 SBU 所需磁盘空间：58 MB

7.8.1. Bison 的安装 准备编译 Bison：

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

新配置选项的含义：

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

该选项指示构建系统将 bison 文档安装到带版本号的目录中。

编译该软件包：

```
make
```

安装该软件包：

```
make install
```

该软件包的详细信息参见第 8.34.2 节 "Bison 的内容"。

7.9. Perl-5.40.1 Perl 软件包包含实用报表提取语言。

预计编译时间: 0.6 SBU

所需磁盘空间: 285 MB

7.9.1. Perl 的安装 准备编译 Perl:

```
sh Configure -des \
-D prefix=/usr \
-D vendorprefix=/usr \
-D useshrplib \
-D privlib=/usr/lib/perl5/5.40/core_perl \
-D archlib=/usr/lib/perl5/5.40/core_perl \
-D sitelib=/usr/lib/perl5/5.40/site_perl \
-D sitearch=/usr/lib/perl5/5.40/site_perl \
-D vendorlib=/usr/lib/perl5/5.40/vendor_perl \
-D vendorarch=/usr/lib/perl5/5.40/vendor_perl
```

Configure 选项的含义:

-des

这是三个选项的组合: -d 表示对所有项目使用默认值; -e 确保完成所有任务; -s 则抑制非必要输出。

-D vendorprefix=/usr

这确保 perl 知道如何告知软件包应该将它们的 Perl 模块安装到何处。

-D useshrplib

将某些 Perl 模块所需的 libperl 构建为共享库, 而非静态库。

-D privlib, -D archlib, -D sitelib, ...

这些设置定义了 Perl 查找已安装模块的位置。LFS 编辑团队选择将其置于基于 Perl 主次版本号(5.40)的目录结构中, 这样在升级到新版补丁级别(完整版本号中最后点分部分, 如 5.40.1)时无需重新安装所有模块。

编译该软件包:

make

安装该软件包:

make install

该软件包的详细信息见第 8.43.2 节"Perl 的内容"。

7.10. Python-3.13.2 Python 3 软件包包含 Python 开发环境。它适用于面向对象编程、编写脚本、大型程序原型设计以及开发完整应用程序。Python 是一种解释型计算机语言。

预计编译时间：0.5 SBU

所需磁盘空间：634 MB

7.10.1. Python 的安装说明

有两个以"python"前缀命名的软件包文件。需要解压的是 Python-3.13.2.tar.xz (注意首字母大写)。

为 Python 编译做准备：

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

配置选项的含义：

--enable-shared

此选项阻止安装静态库。

--without-ensurepip

此选项禁用 Python 包安装工具，当前阶段无需此功能。

编译该软件包：

make

注意

部分 Python 3 模块因依赖项尚未安装而暂时无法构建。对于 ssl 模块，系统会输出"Python 需要 OpenSSL 1.1.1 或更新版本"的提示信息，该提示可忽略。只需确保顶层 make 命令未执行失败即可。这些可选模块目前并非必需，将在第 8 章进行构建。

安装该软件包：

make install

该软件包的详细信息请参阅第 8.51.2 节"Python 3 的内容"。

7.11. Texinfo-7.2

Texinfo 软件包包含用于读取、编写和转换 info 页面的程序。

预计编译时间：0.2 SBU
所需磁盘空间：152 MB

7.11.1. Texinfo 的安装

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请参阅第 8.72.2 节 "Texinfo 内容"。

7.12. Util-linux-2.40.4

Util-linux 软件包包含各种实用工具程序。

预计编译时间: 0.2 SBU

所需磁盘空间: 182 MB

7.12.1. 安装 Util-linux

FHS 规范建议使用 /var/lib/hwclock 目录而非常规的 /etc 目录作为

`adjtime` 文创建该目录:

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

准备 Util-linux 进行编译:

```
./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-2.40.4
```

配置选项的含义:

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

该参数根据 FHS 标准设置记录硬件时钟信息的文件位置。虽然这个临时工具并不严格要求此项配置，但能避免在其他位置创建文件——这些文件在构建最终 util-linux 软件包时不会被覆盖或删除。

`--libdir=/usr/lib`

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

`--disable-*`

这些选项可避免因构建需要 LFS 未包含或尚未安装的依赖组件而出现警告信息。

`--without-python`

此选项禁用 Python 功能，避免构建不必要的绑定模块。

`runstatedir=/run`

此选项正确设置 `uuid` 和 `libuuid` 使用的套接字文件位置。

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息见第 8.79.2 节 "Util-linux 组件内容"。

7.13. 清理并保存临时系统

7.13.1. 清理工作

首先删除当前安装的文档文件，防止它们出现在最终系统中，同时

可节省约 35MB 空间：

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

其次，在现代 Linux 系统中，libtool 的.la 文件仅对 libltdl 有用。LFS 中的任何库都不会被 libltdl 加载，而且已知某些.la 文件可能导致 BLFS 软件包构建失败。现在删除这些文件：

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

当前系统大小约为 3GB，但/tools 目录已不再需要。该目录占用约 1GB 磁盘空间。现在将其删除：

```
rm -rf /tools
```

7.13.2. 备份

此时，基础程序与库已构建完成，您当前的 LFS 系统处于良好状态。

现在可以对系统进行备份以便后续复用。若后续章节出现致命错误，通常彻底删除所有内容并（更谨慎地）重新开始是最佳恢复方案。遗憾的是，所有临时文件也将被清除。为避免重复已完成的工作耗费额外时间，对当前 LFS 系统进行备份可能非常有用。

注意

本节剩余步骤均为可选操作。但请注意，从第 8 章开始安装软件包时，临时文件将被覆盖。因此按照下文所述对当前系统进行备份是明智之举。

以下步骤需在 chroot 环境外执行。这意味着您必须先退出 chroot 环境再继续操作。此举目的是访问 chroot 环境外部的文件系统位置来存储/读取备份归档文件，该文件不应存放在\$LFS 目录结构中。

若您已决定进行备份，请先退出 chroot 环境：

```
exit
```

重要提示

后续所有指令均需在宿主系统上以 root 身份执行。请特别注意即将运行的命令，此处的操作失误可能修改宿主系统。请注意环境变量 LFS 默认设置为用户 lfs 使用，但可能未对 root 用户设置。

当需要以 root 身份执行命令时，请确保已设置 LFS 变量。

相关内容已在第 2.6 节“设置\$LFS 变量与 Umask 值”中讨论过。

创建备份前，请先卸载虚拟文件系统：

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

请确保在存放备份归档的目录所在文件系统中，至少保留 1GB 可用磁盘空间（源代码包将包含在备份归档中）。请注意，以下指令指定了宿主系统 root 用户的主目录，该目录通常位于根文件系统上。如果您不希望将备份存储在此处，请将\$HOME 替换为您选择的目录。

root 的主目录。

运行以下命令创建备份存档：

注意

由于备份存档是压缩文件，即使在性能相当不错的系统上也需要较长时间（超过 10 分钟）。

```
cd $LFS tar -cJpf $HOME/lfs-temp-tools-12.3.tar.xz .
```

注意

如果继续第 8 章的操作，请勿忘记按照下方“重要提示”框中的说明重新进入 chroot 环境。

7.13.3. 恢复备份

若操作过程中出现错误需要重新开始，可使用此备份恢复系统以节省恢复时间。由于源代码位于\$LFS 目录下，它们也会被包含在备份归档文件中，因此无需重复下载。确认\$LFS 变量设置正确后，可通过

执行以下命令进行恢复：

警告

以下命令极其危险。若以 root 用户身份执行 rm -rf /*且未切换至\$LFS 目录，或未为 root 用户设置 LFS 环境变量，将彻底摧毁您的整个宿主系统。特此警告。

```
cd $LFS
rm -rf /* tar -xpf $HOME/lfs-temp-tools-12.3.tar.xz
```

请再次确认环境配置无误后，继续构建系统的其余部分。

重要提示

如果您离开 chroot 环境进行备份或使用恢复点重新开始构建，请记得检查虚拟文件系统是否仍然挂载（执行 findmnt | grep \$LFS 命令）。如果未挂载，请立即按照第 7.3 节“准备虚拟内核文件系统”所述重新挂载，并在继续操作前重新进入 chroot 环境（参见第 7.4 节“进入 Chroot 环境”）。

第四部分 构建 LFS 系统

第 8 章 安装基础系统软件

8.1. 简介 本章我们将正式开始构建 LFS 系统。

这些软件的安装过程较为直接。尽管在许多情况下安装说明可以更简短通用，但我们选择为每个软件包提供完整说明，以最大限度减少出错可能。理解 Linux 系统运作机制的关键，在于知晓每个软件包的用途及其存在必要性（无论对用户还是系统而言）。

我们不建议使用自定义优化选项。虽然这些优化可能让程序运行稍快，但它们也可能导致编译困难及程序运行时出现问题。如果某个软件包在使用自定义优化时拒绝编译，请尝试不使用优化进行编译，看看是否能解决问题。即便软件包在使用自定义优化时能够编译成功，由于代码与构建工具之间复杂的交互作用，仍存在编译错误的潜在风险。同时请注意，书中未指定的-march 和-mtune 选项值未经测试，可能导致工具链软件包（Binutils、GCC 和 Glibc）出现问题。自定义编译器优化带来的微小性能提升往往得不偿失。我们鼓励首次构建 LFS 的用户不要使用自定义优化选项。

另一方面，我们保留了软件包默认配置启用的优化选项。此外，我们有时会显式启用某些软件包提供的优化配置（这些配置默认未被启用）。软件包维护者已对这些配置进行过测试并认为其安全可靠，因此不太可能导致构建失败。通常默认配置已启用-O2 或-O3 优化级别，因此最终生成的系统即便不进行任何定制优化，仍能保持高速运行且稳定可靠。

在安装指导之前，每个安装页面都会提供软件包的相关信息，包括对其内容的简要说明、预计构建所需时间以及构建过程中占用的磁盘空间。安装指导之后会列出该软件包安装的程序和库文件（附简要说明）。

注意

SBU 数值与所需磁盘空间包含第八章所有适用软件包的测试套件数据。除非特别说明，所有操作的 SBU 值均基于四核 CPU (-j4) 计算得出。

8.1.1. 关于库文件

通常情况下，LFS 编辑团队不建议构建和安装静态库。在现代 Linux 系统中，大多数静态库已过时。此外，将静态库链接到程序中可能带来弊端。若需更新库以修复安全问题，所有使用该静态库的程序都必须重新链接新版库。由于静态库的使用往往不易察觉，相关程序（以及重新链接所需的操作流程）甚至可能无从追溯。

本章节的操作流程会移除或禁用大多数静态库的安装。通常通过向配置脚本传递 --disable-static 参数实现。某些情况下则需要采用替代方案。少数例外场景（尤其是 Glibc 和 GCC）中，静态库仍是软件包构建过程的核心功能组件。

更完整的库文件讨论请参阅 BLFS 手册中的《库文件：静态还是动态？》章节。

8.2. 软件包管理

软件包管理是《LFS 手册》中经常被要求增加的内容。软件包管理器能追踪文件的安装过程，使软件包的移除和升级更加便捷。优秀的软件包管理器还会特殊处理配置文件，以便在重新安装或升级软件包时保留用户配置。在您开始猜测之前，需要明确说明——本节既不会讨论也不会推荐任何特定的软件包管理器。本节实际提供的是对流行技术方案及其工作原理的综述。适合您的完美软件包管理器可能就在这些技术之中，也可能是两种或多种技术的组合。本节还会简要提及升级软件包时可能出现的问题。

LFS 或 BLFS 未提及包管理器的部分原因包括：

- 处理软件包管理会分散这些书籍的核心目标——教授如何构建 Linux 系统。
- 包管理存在多种解决方案，各具优缺点。要找到满足所有用户需求的单一方案实属不易。

关于软件包管理方面提供了一些提示文档。可访问 Hints 项目查看是否有符合需求的解决方案。

8.2.1. 升级注意事项

当新版本发布时，软件包管理器能简化升级流程。通常 LFS 和 BLFS 手册中的指导可用于升级至新版本。以下是升级软件包时需特别注意的事项，尤其针对运行中的系统。

- 若需升级 Linux 内核（例如从 5.10.17 升级至 5.10.18 或 5.11.1），则无需重新构建其他组件。得益于内核与用户空间之间定义良好的接口，系统仍可正常运行。具体而言，Linux API 头文件无需随内核一同升级，仅需重启系统即可使用新内核。
- 如需将 Glibc 升级至新版（如从 Glibc-2.36 升级至 Glibc-2.41），则需采取额外步骤以避免系统损坏。详情请参阅第 8.5 节《Glibc-2.41》的说明。
- 若某个包含共享库的软件包被更新，且库文件名发生变更，则所有动态链接该库的软件包都必须重新编译，以链接至新版库文件。（请注意：软件包版本与库文件名之间并无必然关联。）例如，假设软件包 foo-1.2.3 安装的共享库名为 libfoo.so.1。当您将其升级至新版 foo-1.2.4 时，该版本安装的共享库名为 libfoo.so.2。此时，所有动态链接 libfoo.so.1 的软件包都需要重新编译，改为链接 libfoo.so.2 才能使用新版库。

在依赖该库的所有软件包完成重新编译前，切勿移除旧版库文件。

- 若某个软件包（直接或间接）同时链接到共享库的旧名称和新名称（例如该软件包同时链接到 libfoo.so.2 和 libbar.so.1，而后者又链接到 libfoo.so.3），由于共享库的不同版本对某些符号名称存在不兼容的定义，可能导致软件包功能异常。这种情况通常发生在共享库提供方升级后，仅重新编译了部分而非全部链接到旧版共享库的软件包。为避免该问题，用户需要尽快重新编译所有链接到更新版本共享库（如从 libfoo.so.2 升级到 libfoo.so.3）的软件包。
- 如果某个包含共享库的软件包被更新，且库名称未改变但库文件的版本号降低（例如库仍名为 libfoo.so.1，但库文件名从 libfoo.so.1.25 变为 libfoo.so.1.24），则应从先前

已安装版本（本例中为 libfoo.so.1.25）。否则，执行 ldconfig 命令（无论是通过命令行手动调用还是由某些软件包安装触发）会将符号链接 libfoo.so.1 重置指向旧库文件——因为旧版本看似“更新”，其版本号数值更大。当您需要降级软件包，或库文件作者变更版本编号规则时，可能出现这种情况。

- 若包含共享库的软件包更新后，库名称未改变但修复了严重问题（特别是安全漏洞），所有链接该共享库的正在运行程序都应重启。更新完成后以 root 身份执行以下命令，可列出仍在使用旧版本库的进程（将 libfoo 替换为实际库名）：

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

若通过 OpenSSH 访问系统且其链接了更新后的库，必须重启 sshd 服务，然后注销并重新登录，再次运行上述命令以确认没有进程仍在使用已删除的库文件。

- 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代码或数据的进程可能会崩溃。正确更新程序或共享库而不导致进程崩溃的方法是先删除旧版本，再安装新版本。coreutils 提供的 install 命令已实现这一功能，大多数软件包都使用该命令来安装二进制文件和库文件。这意味着在大多数情况下您不会受此问题困扰。但某些软件包的安装过程（特别是 BLFS 中的 SpiderMonkey）会直接覆盖现有文件，这将导致崩溃。因此更安全的做法是在更新软件包前保存工作并关闭不必要的运行进程。

8.2.2. 软件包管理技术

以下是几种常见的软件包管理技术。在决定采用某种包管理器之前，请对各种技术（尤其是每种特定方案的缺点）进行充分调研。

8.2.2.1. 全凭记忆管理！没错，这也是一种软件包管理方式。有些用户不需要包管理器，因为他们对软件包了如指掌，清楚每个包会安装哪些文件。还有些用户不需要任何包管理方案，因为他们计划在每次修改软件包时都重新构建整个系统。

8.2.2.2. 独立目录安装法这是一种简单的包管理技术，无需专用程序来管理软件包。每个软件包都安装在独立目录中。例如，foo-1.1 软件包会被安装到/opt/foo-1.1 目录，并创建从/opt/foo 指向/opt/foo-1.1 的符号链接。当新版本 foo-1.2 发布时，它会被安装到/opt/foo-1.2 目录，并将原有符号链接重新指向新版本。

可能需要扩展的环境变量包括 PATH、MANPATH、INFOPATH、PKG_CONFIG_PATH、CPPFLAGS、LDFLAGS 以及配置文件 so.conf，以包含/opt/foo-x.y 目录下的相应子目录。

BLFS 手册采用此方案安装一些非常庞大的软件包，以便于后续升级。但若安装过多软件包，该方案将变得难以管理。某些软件包（如 Linux API 头文件和 Glibc）可能无法与此方案良好兼容。切勿在整个系统范围内使用此方案。

8.2.2.3. 符号链接式软件包管理

这是前一种软件包管理技术的变体。每个软件包的安装方式与前述方案相同。

不同之处在于，不是通过通用包名创建符号链接，而是将每个文件都符号链接到/usr 目录结构中。这样就无需扩展环境变量。虽然用户可以手动创建这些符号链接，但许多软件包管理器采用这种方法，并自动完成符号链接的创建。常见的此类工具包括 Stow、Epkg、Graft 和 Depot。

需要欺骗安装脚本，让软件包误以为它被安装在/usr 目录下，而实际上它被安装在/usr/pkg 层级结构中。以这种方式安装通常并非易事。例如，假设您正在安装 libfoo-1.1 软件包，以下指令可能无法正确安装该软件包：

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

安装过程可以完成，但依赖包可能不会如预期那样链接到 libfoo。如果你编译一个依赖 libfoo 的软件包，可能会发现它链接到了/usr/pkg/libfoo/1.1/lib/libfoo.so.1 而非/

正如你所预期的，usr/lib/libfoo.so.1 文件会被安装。正确的方法是使用 DESTDIR 变量来指定安装路径。具体操作如下：

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

大多数软件包都支持这种方法，但也有少数不支持。对于这些不兼容的软件包，你可能需要手动安装，或者发现将它们安装到 /opt 目录下更为简便。

8.2.2.4. 基于时间戳的方法

这种技术是在安装软件包之前对一个文件进行时间戳标记。安装完成后，只需使用带有适当选项的 find 命令，就能生成自时间戳文件创建以来所有被安装文件的日志。采用这种方法的包管理器是 install-log。

虽然这种方案具有简单易行的优点，但也存在两个缺陷。如果在安装过程中，文件被安装时的时间戳不是当前时间，这些文件将不会被包管理器追踪到。此外，该方案仅适用于逐个安装软件包的情况。若从两个不同控制台同时安装两个软件包，日志记录将不可靠。

8.2.2.5. 追踪安装脚本

该方法通过记录安装脚本执行的命令来实现。具体可采用两种技术：

可供使用：

可以设置 LD_PRELOAD 环境变量指向一个预加载库。在安装过程中，该库会通过附着到 cp、install、mv 等可执行文件上，追踪修改文件系统的系统调用，从而记录正在安装的软件包。使用此方法时，所有可执行文件都必须是动态链接且未设置 uid 或 sgid 位。预加载库可能导致安装过程中出现意外副作用，因此建议进行测试以确保包管理器不会破坏系统，并能正确记录所有相关文件。

另一种技术是使用 strace 工具，它会记录安装脚本执行期间所有的系统调用。

8.2.2.6. 创建软件包归档文件 该方案中，软件包安装过程会通过符号链接式软件包管理章节所述方式，将文件伪安装至独立目录树。安装完成后，系统会根据已安装文件创建软件包归档。该归档文件可用于在本地机器甚至其他机器上安装软件包。

商业发行版中的大多数软件包管理器均采用此方案。遵循此方案的软件包管理器包括 RPM（恰好是 Linux 标准基础规范要求的管理器）、pkg-utils、Debian 的 apt 以及 Gentoo 的 Portage 系统。关于如何为 LFS 系统采用此类软件包管理方式的提示文档，可访问 <https://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt> 获取。

创建包含依赖关系的软件包文件较为复杂，已超出 LFS 手册的讨论范围。

Slackware 采用基于 tar 的软件包归档系统。该系统有意不像更复杂的软件包管理器那样处理依赖关系。有关 Slackware 软件包管理的详细信息，请参阅 <https://www.slackbook.org/html/package-management.html>。

8.2.2.7. 基于用户的管理方案 这一 LFS 特有的方案由 Matthias Benkmann 设计，可通过 Hints 项目获取。该方案将每个软件包作为独立用户安装到标准路径中，通过检查用户 ID 即可轻松识别属于特定软件包的文件。这种方法的优缺点较为复杂，本节不便详述。

具体细节请参阅提示文档：https://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt。

8.2.3. 在多台系统上部署 LFS

LFS 系统的优势之一在于其文件不依赖于磁盘系统上的物理位置。

将 LFS 系统克隆到与基础系统架构相同的另一台计算机上非常简单：只需对包含根目录的 LFS 分区使用 tar 命令打包（基础 LFS 构建的未压缩大小约为 900MB），通过网络传输或 CD-ROM/U 盘将该文件复制到新系统后解压即可。之后需要修改部分配置文件，可能需要更新的配置文件包括：/etc/hosts、/etc/fstab、/etc/passwd、/

/etc/group、/etc/shadow、/etc/ld.so.conf、/etc/sysconfig/rc.site、/etc/sysconfig/network 以及 /etc/sysconfig/ifconfig.eth0

根据新系统硬件差异和原始内核配置情况，可能需要为新系统定制内核。

注意

有报告显示在相似但不完全相同的架构之间复制时可能出现问题。例如英特尔系统的指令集与 AMD 处理器并不完全相同，且某些处理器的较新版本可能提供旧版本不具备的指令。

最后，需要通过第 10.4 节“使用 GRUB 设置启动流程”使新系统可引导。

8.3. Man-pages-6.12 Man-pages 软件包包含超过 2400 个手册页。

预计编译时间：0.1 SBU 所需磁盘空间：52 MB

8.3.1. Man-pages 的安装 移除两个关于密码哈希函数的手册页。Libxcrypt 将提供这些手册页的更优版本：

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

运行以下命令安装 Man-pages：

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

选项含义说明：

-R

这会阻止 make 设置任何内置变量。man-pages 的构建系统与内置变量配合不佳，但目前除了通过命令行显式传递-R 参数外，没有其他方法禁用这些内置变量。

GIT=false

这能防止构建系统输出大量"git: command not found"警告信息。

8.3.2. Man-pages 安装文件内容 已安装文件：

各类手册页

简要描述

手册页

描述 C 语言函数、重要设备文件及关键配置文件

8.4. lana-Etc-20250123 lana-Etc 软件包提供网络服务和协议的相关数据。

预计构建时间：少于 0.1 SBU

所需磁盘空间：4.8 MB

8.4.1. 安装 lana-Etc 对于此软件包，我们只需将文件复制到指定位置：

```
cp services protocols /etc
```

8.4.2. lana-Etc 安装文件内容：

/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 互联网协议

/etc/services 提供互联网服务的友好文本名称与其底层分配的端口号和协议类型之间的映射关系

8.5. Glibc-2.41 Glibc 软件包包含主要的 C 语言库。该库提供了分配内存、搜索目录、打开和关闭文件、读写文件、字符串处理、模式匹配、算术运算等基本例程。

预计编译时间：12 SBU 所需磁盘空间：3.2 GB

8.5.1. Glibc 的安装 部分 Glibc 程序会使用不符合 FHS 规范的/var/db 目录存储运行时数据。应用以下补丁可使这些程序将运行时数据存储在符合 FHS 规范的位置：

```
patch -Np1 -i ../glibc-2.41-fhs-1.patch
```

Glibc 文档建议在专用构建目录中构建 Glibc：

```
mkdir -v build  
进入目录 构建
```

确保将 ldconfig 和 sln 工具安装到/usr/sbin 目录：

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

准备编译 Glibc：

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

配置选项的含义：

--disable-werror

该选项禁用传递给 GCC 的 -Werror 选项，这是运行测试套件所必需的。

--enable-kernel=5.4

此选项告知构建系统该 Glibc 可兼容低至 5.4 版本的内核。这意味着当系统调用在更高版本中引入而无法使用时，将生成相应的变通方案。

--enable-stack-protector=strong

该选项通过添加额外代码来检测缓冲区溢出（如栈粉碎攻击），从而增强系统安全性。请注意 Glibc 总是显式覆盖 GCC 的默认设置，因此即使我们已为 GCC 指定--enable-default-ssp 参数，仍需保留此选项。

--disable-nscd

不构建已不再使用的名称服务缓存守护进程。

libc_cv_slibdir=/usr/lib

此变量为所有系统设置正确的库路径。我们不希望系统使用 lib64 目录。

编译该软件包：

```
make
```

重要提示

在本节中，Glibc 的测试套件被视为关键环节。任何情况下都不应跳过此步骤。

通常会有少量测试无法通过。以下列出的测试失败通常可以安全忽略。

```
make check
```

您可能会看到一些测试失败。Glibc 测试套件在一定程度上依赖于宿主系统。在超过 6000 项测试中出现少量失败通常可以忽略。以下是近期 LFS 版本中最常见的已知问题列表：

- 已知在 LFS chroot 环境中 io/tst-lchmod 测试会失败。
- 部分测试（例如 nss/tst-nss-files-hosts-multi 和 nptl/tst-thread-affinity*）由于超时原因已知会失败（尤其在系统运行较慢和/或使用多线程并行编译运行测试套件时）。可通过以下命令识别这些测试：

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

可通过设置 TIMEOUTFACTOR=<倍数> make test t=<测试名> 来重新运行单个测试并延长超时时间。例如，设置 TIMEOUTFACTOR=10 make test t=nss/tst-nss-files-hosts-multi 将以原始超时时间的十倍重新运行 nss/tst-nss-files-hosts-multi 测试。

- 此外，某些测试可能在较旧的 CPU 型号（例如 elf/tst-cpu-features-cpuinfo）或主机内核版本（例如 stdlib/tst-arc4random-thread）下会失败。

虽然这属于无害提示，但 Glibc 的安装阶段会因缺少 /etc/ld.so.conf 文件而报错。可通过以下命令消除该警告：

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

修复 Makefile 以跳过因现代 Glibc 配置而失败的过时健全性检查：

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

重要提示

若在运行的 LFS 系统上升级 Glibc 至新的次要版本（例如从 Glibc-2.36 升级到 Glibc-2.41），需采取额外预防措施以避免系统崩溃：

- LFS 11.0（不含）之前的系统不支持升级 Glibc。若您仍在使用此类旧版 LFS 系统但需要新版 Glibc，请重新构建 LFS。
- 若在 LFS 12.0（不含）之前的系统上升级，请按照第 8.27 节“Libxcrypt-4.4.38”安装 Libxcrypt。除常规安装外，必须遵循 Libxcrypt 章节中的说明安装 libcrypt.so.1*（替换先前 Glibc 安装中的 libcrypt.so.1）。
- 若在 LFS 12.1（不含）之前的系统上升级，需移除 nscd 程序：

删除 /usr/sbin/nscd 文件
- 若内核版本低于 5.4（可通过 uname -r 命令查看当前版本）或您执意要升级内核，请按照第 10.3 节“Linux-6.13.4”的说明升级内核并重启系统。
- 若内核 API 头文件版本低于 5.4（可通过 cat /usr/include/linux/version.h 命令查看当前版本）或您执意要升级，请按照第 5.4 节“Linux-6.13.4 API 头文件”的说明进行升级（但需从 cp 命令中移除 \$LFS 参数）。
- 执行 DESTDIR 安装方式，并通过单条 install 命令同时升级系统中的 Glibc 共享库：

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

除非您完全理解自己在做什么，否则必须严格遵循上述步骤。任何意外偏离都可能导致系统完全无法使用。特此警告。

然后继续运行 make install 命令、针对/usr/bin/ldd 的 sed 命令以及安装区域设置的命令。完成后立即重启系统。

当系统成功重启后，如果您运行的 LFS 系统版本早于 12.0（不含），且 GCC 构建时未使用--disable-fixincludes 选项，请将两个 GCC 头文件移至更合适的位置，并删除 Glibc 头文件的陈旧“fixed”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
$(dirname $(gcc -print-libgcc-file-name))/include-fixed/*
```

安装该软件包：

```
/Makefile make install
```

修复 ldd 脚本中加载器路径的硬编码问题：

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

接下来安装能让系统以不同语言响应的区域设置。这些区域设置都不是必需的，但如果缺少某些设置，部分软件包的测试套件会跳过重要测试用例。

可以使用 `localeddef` 程序单独安装区域设置。例如，下面第二条 `localeddef` 命令将字符集无关的区域定义文件 /usr/share/i18n/locales/cs_CZ 与字符映射定义文件 /usr/share/i18n/charmaps=UTF-8.gz 结合，并将结果追加到 /usr/lib/locale-archive 文件中。以下指令将安装确保测试最佳覆盖率所需的最小区域设置集合：

```
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
SHIFT_JIS ja_JP.SJIS 2> /dev/null || true localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
localedef -i nl_NL@euro -f ISO-8859-15 nl_NL@euro localedef -i ru_RU -f KOI8-R
ru_RU.KOI8-R localedef -i ru_RU -f UTF-8 ru_RU.UTF-8 localedef -i se_NO -f UTF-8
se_NO.UTF-8
```

```
localedef -i ta_IN -f UTF-8 ta_IN.UTF-8 localedef -i tr_TR -f UTF-8
tr_TR.UTF-8 localedef -i zh_CN -f GB18030 zh_CN.GB18030 localedef -
i zh_HK -f BIG5-HKSCS zh_HK.BIG5-HKSCS localedef -i zh_TW -f UTF-8
zh_TW.UTF-8
```

此外，请为您所在国家/地区的语言和字符集安装对应的区域设置。

或者，也可以通过以下耗时命令一次性安装 glibc-2.41/localedata/SUPPORTED 文件中列出的所有区域设置（包含上述所有区域及更多选项）：

```
make locedata/install-locales
```

当需要时，可使用 `localeddef` 命令创建并安装未列在 glibc-2.41/localedata/SUPPORTED 文件中的区域设置。例如，本章后续部分测试将需要以下两个区域设置：

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

注意

Glibc 在解析国际化域名时现在使用 libidn2。这是一个运行时依赖项。如果需要此功能，安装 libidn2 的说明可在 BLFS libidn2 页面找到。

8.5.2. 配置 Glibc

8.5.2.1. 添加 nsswitch.conf 需要创建 /etc/nsswitch.conf 文件，因为 Glibc 的默认设置在网络环境中效果不佳。

通过运行以下命令创建新文件 /etc/nsswitch.conf:

```
cat > /etc/nsswitch.conf << "EOF"
# 开始 /etc/nsswitch.conf 文件内容

passwd: files
组: 文件
影子: 文件

主机: 文件 dns
网络: 文件

协议: 文件
服务: 文件
以太网地址: 文件
远程过程调用: 文件

# 结束 /etc/nsswitch.conf 文件
EOF
```

8.5.2.2. 添加时区数据 通过以下命令安装并设置时区数据:

```
tar -xf ../../tzdata2025a.tar.gz
```

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

```
为以下时区执行操作: etcetera southamerica northamerica europe africa antarctica \
asia australasia backward; 执行命令: zic -L /dev/null -d
$ZONEINFO ${tz} zic -L /dev/null -d $ZONEINFO/posix ${tz}
zic -L leapseconds -d $ZONEINFO/right ${tz} done
```

复制文件: cp -v zone.tab zone1970.tab iso3166.tab \$ZONEINFO 并执
行: zic -d \$ZONEINFO -p America/New_York 取消设置变量: unset
ZONEINFO tz

zic 命令的含义:

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

这将创建不含闰秒的 POSIX 时区。按照惯例，这些时区数据应同时存放在 zoneinfo 和 zoneinfo/posix 目录下。必须将 POSIX 时区放入 zoneinfo 目录，否则多个测试套件会报错。在嵌入式系统中，若存储空间紧张且不打算更新时区数据，可通过不使用 posix 目录节省 1.9MB 空间，但某些应用程序或测试套件可能会出现故障。

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

这将创建正确的时区信息，包括闰秒。在嵌入式系统中，若空间紧张且您不打算更新时区数据，或不在意时间准确性，可以通过省略该部分节省 1.9MB 空间
正确的 目录

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

这将创建 `posixrules` 文件。我们选择纽约是因为 POSIX 要求夏令时规则必须符合美国标准。

确定本地时区的一种方法是运行以下脚本：

```
tzselect
```

回答几个关于地理位置的问题后，脚本会输出时区名称（例如 `America/Edmonton`）。在 `/usr/share/zoneinfo` 目录下还存在其他可用时区（如 `Canada/Eastern` 或 `EST5EDT`），这些时区虽未被脚本识别但同样可以使用。

然后通过以下命令创建 `/etc/localtime` 文件：

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

请将替换为所选时区名称（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`)会在程序运行时搜索 `/usr/lib` 目录以寻找所需的动态库。但如果动态库存放在 `/usr/lib` 以外的目录，就需要将这些目录添加到 `/etc/ld.so.conf` 文件中，以便动态加载器能够找到它们。通常包含额外库文件的两个目录是 `/usr/local/lib` 和 `/opt/lib`，因此需要将这些目录加入动态加载器的搜索路径。

通过以下命令创建新的 `/etc/ld.so.conf` 文件：

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

```
EOF
```

若需要，动态加载器还可以搜索某个目录并包含其中文件的内容。通常该包含目录中的文件仅包含一行指定所需库路径的内容。要添加此功能，请运行以下命令：

```
cat >> /etc/ld.so.conf << "EOF"
# 添加包含目录 include
/etc/ld.so.conf.d/*.conf
```

```
EOF mkdir -pv /etc/ld.so.conf.d
```

8.5.3. Glibc 安装内容

已安装程序：

已安装库文件：

已安装的目录：

gencat、getconf、getent、iconv、iconvconfig、ldconfig、ldd、lddlibc4、ld.so
 (指向 ld-linux-x86-64.so.2 或 ld-linux.so.2 的符号链接)、locale、localeddef、
 makedb、mtrace、pcprofiledump、pldd、sln、sotruss、sprof、tzselect、
 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}、
 /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、

简要描述

gencat 生成消息目录

getconf 显示文件系统特定变量的系统配置值

getent 从管理数据库中获取条目

iconv 执行字符集转换

iconvconfig 创建快速加载的 iconv 模块配置文件

ldconfig 配置动态链接器运行时绑定

ldd 报告每个给定程序或共享库所需的共享库

lddlibc4 辅助 ldd 处理目标文件。在 x86_64 等新架构上不存在该工具

locale 打印当前区域设置的各种信息

localeddef 编译区域设置规范

makedb 根据文本输入创建简单数据库

mtrace 读取并解析内存跟踪文件，以人类可读格式显示摘要信息

pcprofiledump 转储由 PC 性能分析生成的信息

pldd 列出运行进程使用的动态共享对象

sln 静态链接的 ln 程序

sotruss 追踪指定命令的共享库过程调用

sprof 读取并显示共享对象性能分析数据

tzselect 询问用户系统所在位置并返回对应时区描述

xtrace 通过打印当前执行的函数来追踪程序的执行过程

zdump 时区信息转储工具

zic 时区编译器

时区编译器

ld-* .so

共享库可执行文件的辅助程序

libBrokenLocale

Glibc 内部使用的一种粗糙解决方案，用于使存在缺陷的程序（例如某些 Motif 应用程序）能够运行。更多信息请参阅 glibc-2.41/locale/broken_cur_max.c 文件中的注释

libanl

空函数库。原为异步名称查询库，其功能现已整合至 libc

libc

主 C 语言库

libc_malloc_debug

预加载时启用内存分配检查

libdl

空函数库。原为动态链接接口库，其功能现已整合至 libc

libg

空函数库。先前是 g++ 的运行时库

libm

数学函数库

libmvec

向量数学库，在使用 libm 时会根据需要链接

libmcheck

当链接时启用内存分配检查

libmemusage

被 memusage 用于帮助收集程序内存使用情况的信息

libnsl

网络服务库（现已弃用）

libnss_*

名称服务切换模块，包含解析主机名、用户名、组名、别名、服务、协议等功能的函数库。由 libc 根据 /etc/nsswitch.conf 中的配置加载

libpcprofile

可预加载至 PC 配置文件的可执行程序

libpthread

空函数库，不含任何功能函数。先前版本包含提供 POSIX.1c 线程扩展规范大部分接口的函数，以及 POSIX.1b 实时扩展规范中信号量接口的函数，现这些功能已迁移至 libc 库

libresolv

包含用于创建、发送和解析互联网域名服务器数据包的函数

librt

包含提供 POSIX.1b 实时扩展规范中大部分接口的函数

libthread_db

包含用于构建多线程程序调试器的实用函数

libutil

空函数库。先前包含用于多种 Unix 工具的“标准”函数代码，这些函数现已移至 libc 库

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例程。

预计编译时间：少于 0.1 SBU

所需磁盘空间：6.4 MB

8.6.1. Zlib 的安装

准备编译 Zlib：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

移除无用的静态库：

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

8.6.2. Zlib 安装内容

已安装库文件： libz.so

简要描述

libz 包含某些程序使用的压缩与解压功能

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 压缩文本文件相比传统的 gzip 能获得更高的压缩率。

预计构建时间：少于 0.1 SBU

所需磁盘空间：7.2 MB

8.7.1. 安装 Bzip2

应用一个补丁来安装该软件包的文档：

```
patch -Np1 -i ../bzip2-1.0.8-install_docs-1.patch
```

以下命令确保符号链接的安装是相对路径：

```
sed -i 's@\(\ln -s -f \)\$PREFIX/bin/@\1@' Makefile
```

确保将手册页安装到正确位置：

```
sed -i "s@(PREFIX)/man@(PREFIX)/share/man@g" Makefile
```

准备用以下命令编译 Bzip2：

```
make -f Makefile-libbz2_so make
clean
```

make 参数的含义：

```
-f Makefile-libbz2_so
```

这将导致 Bzip2 使用不同的 Makefile 文件进行构建，此处使用的是 Makefile-libbz2_so 文件，该文件会创建一个动态库 libbz2.so 并将 Bzip2 工具链接到此库。

编译并测试软件包：

```
make
```

安装程序：

```
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
```

将共享版 bzip2 二进制文件安装到/usr/bin 目录，并用符号链接替换两份 bzip2 副本：

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

移除无用的静态库：

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

8.7.2. Bzip2 安装内容

已安装程序：

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) 和 bzmore

已安装的库文件：libbz2.so

已安装目录： /usr/share/doc/bzip2-1.0.8

简要描述

bunzip2 解压缩经过 bzip 处理的文件

bzcat 解压缩到标准输出

bzcmp 对 bzip 压缩文件执行 cmp 比较

bzdiff 对 bzip 压缩文件执行 diff 差异比对

bzegrep 在 bzip 压缩文件上运行 egrep 命令

bzfgrep 在 bzip 压缩文件上运行 fgrep 命令

bzgrep 在 bzip 压缩文件上运行 grep 命令

bzip2 采用 Burrows-Wheeler 块排序文本压缩算法结合霍夫曼编码对文件进行压缩；其压缩率优于使用 "Lempel-Ziv" 算法的传统压缩工具（如 gzip）

bzip2recover 尝试从损坏的 bzip 压缩文件中恢复数据

bzless 对 bzip 压缩文件执行 less 查看操作

bzmore 对 bzip 压缩文件执行 more 命令

libbz2 实现无损块排序数据压缩的库，采用 Burrows-Wheeler 算法

8.8. Xz-5.6.4 Xz 软件包包含用于压缩和解压文件的程序。它支持 lzma 和更新的 xz 压缩格式。使用 xz 压缩文本文件能获得比传统 gzip 或 bzip2 命令更高的压缩率。

预计构建时间: 0.1 SBU

所需磁盘空间: 21 MB

8.8.1. Xz 的安装

通过以下命令准备编译 Xz:

```
./configure --prefix=/usr \
--disable-static \
docdir=/usr/share/doc/xz-5.6.4
```

编译该软件包:

make

要测试结果, 请执行:

make check

安装该软件包:

make install

8.8.2. 已安装的 Xz 程序内容:

lzcat (链接到 xz)、lzcmp (链接到 xzdiff)、lzdif (链接到 xzdiff)、lzegrep (链接到 xzgrep)、lzfgrep (链接到 xzgrep)、lzgrep (链接到 xzgrep)、lzless (链接到 xzless)、lzma (链接到 xz)、lzmadec、lzmainfo、lzmore (链接到 xzmore)、unlzma (链接到 xz)、unxz (链接到 xz)、xz、xzcat (链接到 xz)、xzcmp (链接到 xzdiff)、xzdec、xzdiff、xzegrep (链接到 xzgrep)、xzfgrep (链接到 xzgrep)、

已安装库文件: liblzma.so

已安装目录:

/usr/include/lzma 及 /usr/share/doc/xz-5.6.4

简要描述

lzcat 解压内容至标准输出

lzcmp 对 LZMA 压缩文件执行 cmp 比较

lzdif 对 LZMA 压缩文件执行 diff 差异比对

lzegrep 对 LZMA 压缩文件执行 egrep 扩展正则搜索

lzfgrep 对 LZMA 压缩文件执行 fgrep 固定字符串搜索

lzgrep 在 LZMA 压缩文件上运行 grep 命令

lzless 对 LZMA 压缩文件执行 less 操作

lzma 使用 LZMA 格式压缩或解压文件

lzmadec 针对 LZMA 压缩文件的小型快速解码器

lzmainfo 显示 LZMA 压缩文件头中存储的信息

lzmore 对 LZMA 压缩文件执行 more 命令

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 格式压缩或解压文件

xzcat 解压缩到标准输出

xzcmp 对 XZ 压缩文件执行 cmp 比较操作

xzdec 一个针对 XZ 压缩文件的小型快速解码器

xzdiff 对 XZ 压缩文件运行 diff 命令

xzegrep 对 XZ 压缩文件运行 egrep 命令

xzfgrep 对 XZ 压缩文件运行 fgrep 命令

xzgrep 对 XZ 压缩文件执行 grep 搜索

xzless 对 XZ 压缩文件执行 less 查看

xzmore 对 XZ 压缩文件执行 more 查看

liblzma 采用 Lempel-Ziv-Markov 链算法实现无损块排序数据压缩的库

8.9. Lz4-1.10.0 Lz4 是一种无损压缩算法，单核压缩速度超过 500MB/s。其解码器速度极快，单核可达数 GB/s。Lz4 可与 Zstandard 协同工作，使两种算法都能更快地压缩数据。

预计编译时间：0.1 SBU 所需磁盘空

间：4.2 MB

8.9.1. 安装 Lz4 编译该软件包：

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

要测试结果，请执行：

```
make -j1 check
```

安装该软件包：

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

8.9.2. Lz4 安装内容

已安装程序： lz4、lz4c（链接到 lz4）、lz4cat（链接到 lz4）以及 unlz4（链接到 lz4）

已安装的库： liblz4.so

简要描述

lz4 使用 LZ4 格式压缩或解压文件

lz4c 使用 LZ4 格式压缩文件

lz4cat 列出使用 LZ4 格式压缩的文件内容

unlz4 使用 LZ4 格式解压文件

liblz4 实现无损数据压缩的库，采用 LZ4 算法

8.10. Zstd-1.5.7 Zstandard 是一种实时压缩算法，能提供高压缩比。它在压缩速度与压缩率之间提供了非常广泛的调节空间，同时拥有极快的解码速度。

预计编译时间：0.4 SBU

所需磁盘空间：85 MB

8.10.1. 安装 Zstd

编译该软件包：

```
make prefix=/usr
```

注意

在测试输出中有多处显示'failed'，这些是预期情况，只有'FAIL'才表示实际测试失败。测试不应出现任何失败结果。

要测试结果，请执行：

```
make check
```

安装该软件包：

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

移除静态库：

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

8.10.2. Zstd 安装内容

已安装程序： zstd、zstdcat（链接至 zstd）、zstdgrep、zstdless、zstdmt（链接至 zstd）以及
已安装的库： unzstd（链接至 zstd） libzstd.so

简要描述

zstd 使用 ZSTD 格式压缩或解压文件

zstdgrep 在 ZSTD 压缩文件上运行 grep 命令

zstdless 在 ZSTD 压缩文件上运行 less 命令

libzstd

实现无损数据压缩的库，采用 ZSTD 算法

8.11. File-5.46 File 软件包包含一个用于确定给定文件类型的实用工具。

预计编译时间：少于 0.1 SBU

所需磁盘空间：19 MB

8.11.1. 安装 File

为编译 File 做准备：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.11.2. 安装的文件内容：

已安装的库：	file libmagic.so
--------	---------------------

简要描述

file	尝试对每个给定文件进行分类；通过执行多项测试实现此功能——包括文件系统测试、魔数测试和语言测试
------	---

libmagic 库	包含用于魔数识别的例程，被 file 程序所使用
------------	--------------------------

8.12. Readline-8.2.13

Readline 软件包是一组提供命令行编辑和历史记录功能的库。

预计构建时间：少于 0.1 SBU

所需磁盘空间：16 MB

8.12.1. Readline 的安装

重新安装 Readline 会导致旧库文件被重命名为.old。虽然这通常不会造成问题

在某些情况下，这可能会触发 ldconfig 中的链接错误。通过执行以下两个 sed 命令可以避免该问题：

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

防止将库搜索路径(rpath)硬编码到共享库中。该软件包安装到标准位置时不需要 rpath，且 rpath 有时会导致不良影响甚至安全问题：

```
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-8.2.13
```

新配置选项的含义：

--with-curses

该选项告知 Readline 可以在 curses 库而非单独的 termcap 库中查找 termcap 库函数。这将生成正确的 readline.pc 文件。

编译该软件包：

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

该 make 选项的含义：

SHLIB_LIBS="-lncursesw"

此选项强制 Readline 链接到 libncursesw 库。详情请参阅软件包 README 文件中的“共享库”章节。

该软件包不包含测试套件。

安装该软件包：

```
make install
```

如需安装文档，请执行：

```
install -v -m644 doc/*. {ps, pdf, html, dvi} /usr/share/doc/readline-8.2.13
```

8.12.2. Readline 安装内容 已安装库文件：

libhistory.so 和 libreadline.so

已安装的目录： /usr/include/readline 和 /usr/share/doc/readline-8.2.13

简要描述

libhistory

提供统一的用户界面用于回溯历史命令记录

libreadline

提供一组命令，用于操作在程序交互会话中输入的文本

8.13. M4-1.4.19 M4 软件包包含一个宏处理器。

预计编译时间：0.3 SBU 所需磁盘空间：49 MB

8.13.1. M4 的安装 准备编译 M4：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.13.2. 已安装的 M4 程序内容：

m4

简要描述

m4 在复制给定文件时会展开其中包含的宏。这些宏可以是内置的，也可以是用户

定义的，并且可以接受任意数量的参数。除了执行宏展开外，m4 还具有内置功能，包括引入指定文件、运行 Unix 命令、执行整数运算、操作文本、递归等。m4 程序既可以作为编译器的前端使用，也可以单独作为宏处理器使用

8.14. Bc-7.0.3 Bc 软件包包含一个任意精度的数值处理语言。

预计编译时间：少于 0.1 SBU
所需磁盘空间：
7.8 MB

8.14.1. Bc 的安装 准备编译 Bc：

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

配置选项的含义：

`CC=gcc`

该参数指定要使用的编译器。

`-G`

省略测试套件中在 bc 程序安装前无法运行的部分。

`-O3`

指定要使用的优化级别。

`-r`

启用 Readline 功能以增强 bc 的行编辑特性。

编译该软件包：

```
make
```

要测试 bc， 请运行：

```
make test
```

安装该软件包：

```
make install
```

8.14.2. Bc 安装内容 已安装程序：

bc 和 dc

简要描述

bc 一款命令行计算器

dc 一款逆波兰式命令行计算器

8.15. Flex-2.6.4 Flex 软件包包含一个用于生成文本模式识别程序的工具。

预计编译时间: 0.1 SBU 所需磁盘空间: 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
```

要测试结果, 请执行:

```
make check
```

安装该软件包:

```
make install
```

部分程序尚未适配 flex, 仍会尝试运行其前身 lex。为支持这些程序, 需创建一个名为 lex 的符号链接以 flex 模拟模式运行, 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 安装内容

已安装程序: flex、flex++ (指向 flex 的链接) 及 lex (指向 flex 的链接)

已安装的库: libfl.so 已安装目录:

/usr/share/doc/flex-2.6.4

简要描述

flex 一款用于生成文本模式识别程序的工具; 它支持灵活指定模式查找规则, 无需专门开发定制程序

flex++ flex 的扩展版本, 用于生成 C++代码和类。它是 flex 的符号链接

lex 一个以 lex 模拟模式运行 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 (Tool Command Language)，这是一种健壮的通用脚本语言。Expect 软件包就是用 Tcl (发音为"tickle") 编写的。

预计编译时间：3.1 SBU

所需磁盘空间：91 MB

8.16.1. Tcl 的安装 本软件包及随后两个 (Expect 和 DejaGNU) 的安装是为了支持运行 Binutils、GCC 及其他软件包的测试套件。仅为测试目的安装三个软件包看似有些过度，但了解这些最重要的工具是否正常工作非常令人安心，即便不是绝对必要。

准备编译 Tcl:

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

新配置参数的含义：

--disable-rpath

该参数可防止将库搜索路径(rpath)硬编码到二进制可执行文件和共享库中。此软件包安装到标准位置时不需要 rpath，且 rpath 有时会导致不良影响甚至安全问题。

构建该软件包：

make

```
sed -e "s|${SRCDIR}/unix|/usr/lib|" \
-e "s|${SRCDIR}|/usr/include|" \
-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|" \
-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|" \
-pkgs/itcl4.3.2/itclConfig.sh
```

取消设置 SRCDIR 变量

"make" 命令后的多条 "sed" 指令会从配置文件中移除对构建目录的引用，并将其替换为安装目录路径。这对后续 LFS 构建并非强制要求，但若之后构建的某个软件包需要使用 Tcl 时可能需要此操作。

要测试结果，请执行：

make test

安装该软件包：

```
make install
```

将已安装的库设为可写，以便后续可以移除调试符号：

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

安装 Tcl 的头文件。下一个软件包 Expect 需要这些头文件。

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

现在创建一个必要的符号链接：

```
ln -sfv tclsh8.6 /usr/bin/tclsh
```

重命名与 Perl 手册页冲突的手册页：

```
mv /usr/share/man/man3/{Thread, Tc1_Thread}.3
```

可选择通过执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 安装内容 已安装程序：

tclsh (链接到 tclsh8.6) 和 tclsh8.6

已安装的库： libtcl8.6.so 和 libtclstub8.6.a

简要描述

tclsh8.6 Tcl 命令解释器

tclsh tclsh8.6 的链接

libtcl8.6.so 库文件 Tcl 程序库

libtclstub8.6.a 静态库 Tcl 存根库

8.17. Expect-5.45.4 Expect 软件包包含通过脚本对话实现自动化的工具，可用于 telnet、ftp、passwd、fsck、rlogin 和 tip 等交互式应用程序。该工具不仅适用于测试这些应用程序，还能简化其他方式难以完成的各种任务。DejaGnu 测试框架就是用 Expect 编写的。

预计构建时间：0.2 SBU

所需磁盘空间：3.9 MB

8.17.1. Expect 的安装

Expect 需要 PTY 才能正常工作。通过在 chroot 环境中执行以下命令来验证 PTY 是否正常工作：

一个简单的测试：

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

该命令应输出 ok。若输出显示 OSError: out of pty devices，则表明当前环境未正确配置 PTY 操作。您需要退出 chroot 环境，重新阅读第 7.3 节“准备虚拟内核文件系统”，并确保 devpts 文件系统（及其他虚拟内核文件系统）已正确挂载。随后按照第 7.4 节“进入 Chroot 环境”重新进入。此问题必须在继续前解决，否则需要 Expect 的测试套件（例如 Bash、Binutils、GCC、GDBM 以及 Expect 自身的测试套件）将彻底失败，还可能引发其他隐性故障。

现在进行以下修改以支持 gcc-14.1 及以上版本：

```
patch -Np1 -i ../expect-5.45.4-gcc14-1.patch
```

准备编译 Expect：

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

配置选项的含义：

--with-tcl=/usr/lib

该参数用于告知 configure 脚本 tclConfig.sh 文件的位置。

--with-tclinclude=/usr/include

该参数明确告知 Expect 程序 Tcl 内部头文件的查找位置。

构建该软件包：

```
make
```

要测试结果，请执行：

```
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 安装内容

已安装程序： expect
libexpect5.45.4.so

简要描述

expect	根据脚本与其他交互程序进行通信
libexpect-5.45.4.so	包含允许将 Expect 作为 Tcl 扩展使用或直接从 C/C++（无需 Tcl）调用的函数

8.18. DejaGNU-1.6.3 DejaGNU 软件包包含一个在 GNU 工具上运行测试套件的框架。它使用 expect 编写，而 expect 本身又使用 Tcl（工具命令语言）。

预计编译时间：少于 0.1 SBU

所需磁盘空间：6.9 MB

8.18.1. 安装 DejaGNU 上游建议在专用构建目录中构建 DejaGNU：

```
mkdir -v build
进入目录 构建
```

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
```

要测试结果，请执行：

```
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 安装内容

dejagnu 和 runtest

简要描述

dejagnu DejaGNU 辅助命令启动器

runtest 一个包装脚本，用于定位正确的 expect shell 并运行 DejaGNU

8.19. Pkgconf-2.3.0 **pkgconf** 软件包是 **pkg-config** 的继任者，包含一个工具，用于在软件包安装的配置和编译阶段向构建工具传递头文件路径和/或库路径。

预计构建时间：少于 0.1 SBU

所需磁盘空间：4.7 MB

8.19.1. 安装 Pkgconf

准备编译 Pkgconf：

```
./configure --prefix=/usr \
--disable-static \
--docdir=/usr/share/doc/pkgconf-2.3.0
```

编译该软件包：

```
make
```

安装该软件包：

```
make install
```

为保持与原始 **Pkg-config** 的兼容性，请创建两个符号链接：

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

8.19.2. Pkgconf 安装内容

已安装程序： **pkgconf**、**pkg-config**（指向 **pkgconf** 的链接）以及 **bomtool**

已安装库文件： **libpkgconf.so** 安装目录：

/usr/share/doc/pkgconf-2.3.0

简要描述

pkgconf 返回指定库或软件包的元信息

bomtool 从 **pkg-config** 的.pc 文件生成软件物料清单

libpkgconf 库 包含 **pkgconf** 的大部分功能，同时允许 IDE 和编译器等其他工具使用其框架

8.20. Binutils-2.44 Binutils 软件包包含链接器、汇编器以及其他用于处理目标文件的工具。

预计编译时间：1.6 SBU 所需磁盘空间：819 MB

8.20.1. Binutils 的安装 Binutils 文档建议在专用构建目录中编译 Binutils：

```
mkdir -v build
进入目录 构建
```

为编译准备 Binutils 工具链：

```
../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
```

新配置参数的含义：

--enable-ld=default
构建原始的 bfd 链接器，并将其同时安装为 ld（默认链接器）和 ld.bfd。

--enable-plugins
启用链接器的插件支持。

--with-system-zlib
使用已安装的 zlib 库，而非编译内置版本。

编译该软件包：

```
make tooldir=/usr
```

make 参数的含义：

tooldir=/usr
通常情况下，tooldir（最终存放可执行文件的目录）会被设置为\$(exec_prefix)/\$(target_alias)。例如，x86_64 机器会将其扩展为/usr/x86_64-pc-linux-gnu。由于这是一个定制系统，/usr 目录下不需要这个特定目标目录。如果系统用于交叉编译（例如在 Intel 机器上编译生成能在 PowerPC 机器上执行的代码包），才会使用\$(exec_prefix)/\$(target_alias)。

重要提示

本节中 Binutils 的测试套件至关重要。在任何情况下都不得跳过该测试。

测试结果：

```
make -k check
```

要查看失败的测试列表，请运行：

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

安装该软件包：

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

移除无用的静态库及其他文件：

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

8.20.2. Binutils 安装内容 已安装程序：

addr2line, ar, as, c++filt, dwp, elfedit, gprof, gprofng, ld, ld.bfd, nm, objcopy, objdump, ranlib, readelf, size, strings, 和 strip

已安装的库文件： libbfd.so、 libctf.so、 libctf-nobfd.so、 libgprofng.so、 libopcodes.so 以及 libsframe.so
安装目录：
 /usr/lib/ldscripts

简要描述

addr2line	将程序地址转换为文件名和行号；给定一个地址和可执行文件名称，它会利用可执行文件中的调试信息来确定与该地址关联的源文件及行号
ar	创建、修改和提取归档文件
as	将 gcc 输出汇编成目标文件的汇编器
c++filt	链接器用于解析 C++ 和 Java 符号修饰，防止重载函数冲突的工具
dwp	DWARF 打包工具
elfedit	更新 ELF 文件的 ELF 头部信息
gprof	显示调用关系图性能数据
gprofng	收集并分析性能数据
ld	将多个目标文件和归档文件合并为单一文件，重定位数据并绑定符号引用的链接器
ld.bfd	指向 ld 的硬链接
nm	列出给定目标文件中的符号
objcopy	将一种类型的目标文件转换为另一种类型
objdump	显示指定目标文件的相关信息，可通过选项控制具体显示内容；这些信息对开发编译工具的程序员非常有用
ranlib	生成归档文件内容的索引并存入该归档文件；该索引列出了所有由可重定位目标文件构成的归档成员所定义的符号
readelf	显示 ELF 类型二进制文件的相关信息
size	列出指定目标文件的各段大小及总大小

字符串	对每个给定文件，输出长度不小于指定值（默认为 4）的可打印字符序列；对于目标文件，默认仅打印初始化和加载段的字符串，而对于其他类型文件，则会扫描整个文件
剥离	从目标文件中丢弃符号
libbfd	二进制文件描述符库
libctf	兼容 ANSI-C 类型格式的调试支持库
libctf-nobfd	不依赖 libbfd 功能的 libctf 变体库
libgprofng	包含 gprofng 使用的大部分例程的库
libopcodes	一个用于处理操作码（处理器指令的“可读文本”版本）的库；用于构建如 objdump 等实用工具
libsframe	支持使用简单展开器进行在线回溯的库

8.21. GMP-6.3.0 GMP 软件包包含数学函数库，提供用于任意精度算术运算的有用功能。

预计编译时间：0.3 SBU

所需磁盘空间：54 MB

8.21.1. GMP 安装注意事项

如果您正在为 32 位 x86 架构构建，但您的 CPU 支持 64 位代码运行且已在环境变量中指定了 CFLAGS，配置脚本会尝试配置为 64 位模式并导致失败。通过以下方式调用 configure 命令可避免此问题：

```
ABI=32 ./configure ...
```

注意

GMP 的默认设置会生成针对主机处理器优化的库。如果需要适用于性能低于主机 CPU 的处理器的库，可以通过在配置命令后追加-

`-host=none-linux-gnu` 选项来创建通用库。

准备 GMP 进行编译：

```
./configure --prefix=/usr \
--enable-cxx \
--disable-static \
--docdir=/usr/share/doc/gmp-6.3.0
```

新增配置选项的含义：

`--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存放位置

编译该软件包并生成 HTML 文档：

`make`

生成 HTML 文档

重要提示

本节 GMP 的测试套件至关重要。在任何情况下都不得跳过该测试。

测试结果：

执行检查命令并将输出同时显示在屏幕和日志文件中：

```
make check 2>&1 | tee gmp-check-log
```

注意

GMP 库的代码针对构建时的处理器进行了高度优化。偶尔会出现处理器检测代码误判系统能力的情况，导致测试过程或使用 GMP 库的应用程序出现“Illegal instruction”错误。若发生此类情况，应使用`--host=none-linux-gnu` 配置选项重新配置并重建 GMP 库。

确保测试套件中至少有 199 项测试通过。通过运行以下命令检查结果：

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

安装软件包及其文档：

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

8.21.2. 已安装的 GMP 库内容：

libgmp.so 和 libgmpxx.so

安装目录： /usr/share/doc/gmp-6.3.0

简要描述

libgmp 包含高精度数学函数

libgmpxx 包含 C++ 高精度数学函数

8.22. MPFR-4.2.1 MPFR 软件包包含多精度数学运算函数。

预计编译时间：0.2 SBU 所需磁盘空间：43 MB

8.22.1. MPFR 的安装 准备编译 MPFR：

```
./configure --prefix=/usr \
--disable-static \ --enable-thread-safe \ --
docdir=/usr/share/doc/mpfr-4.2.1
```

编译该软件包并生成 HTML 文档：

```
make
生成 HTML 文档
```

重要提示

本节 MPFR 的测试套件至关重要。在任何情况下都不得跳过。

测试结果并确保全部 198 项测试通过：

```
make check
```

安装软件包及其文档：

```
make install
安装 HTML 文档
```

8.22.2. MPFR 安装内容

已安装库文件：	libmpfr.so /usr/share/doc/mpfr-4.2.1
---------	---

简要描述

libmpfr	包含多精度数学函数
---------	-----------

8.23. MPC-1.3.1 MPC 软件包包含一个用于复数运算的函数库，支持任意高精度计算并能正确对结果进行舍入。

预计编译时间：0.1 SBU 所需磁盘空间：22 MB

8.23.1. MPC 的安装 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
生成 HTML 文档
```

要测试结果，请执行：

```
make check
```

安装软件包及其文档：

```
make install
安装 HTML 文档
```

8.23.2. MPC 安装的库文件内容：

安装目录：	libmpc.so /usr/share/doc/mpc-1.3.1
-------	---------------------------------------

简要描述

libmpc 数学库 包含复数数学运算函数

8.24. Attr-2.5.2 Attr 软件包包含用于管理文件系统对象扩展属性的工具。

预计构建时间：少于 0.1 SBU 所需磁盘空间：

4.1 MB

8.24.1. Attr 的安装 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

测试必须在支持扩展属性的文件系统（如 ext2、ext3 或 ext4 文件系统）上运行。要测试结果，请执行：

make check

安装该软件包：

make install

8.24.2. Attr 安装内容

已安装程序： attr、getfattr 和 setattr

已安装库： libattr.so

已安装目录： /usr/include/attr 和 /usr/share/doc/attr-2.5.2

简要描述

attr 扩展文件系统对象的属性

getfattr 获取文件系统对象的扩展属性

setattr 设置文件系统对象的扩展属性

libattr 包含用于操作扩展属性的库函数

8.25. Acl-2.3.2 Acl 软件包包含用于管理访问控制列表（ACL）的工具，这些工具可为文件和目录定义细粒度的自主访问权限。

预计编译时间：少于 0.1 SBU

所需磁盘空间：6.5 MB

8.25.1. Acl 的安装 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
```

ACL 测试必须在支持访问控制权限的文件系统上运行。要测试结果，请执行：

```
make check
```

已知名为 test/cp.test 的测试会失败，因为 Coreutils 尚未构建 ACL 支持功能。

安装该软件包：

```
make install
```

8.25.2. ACL 安装内容

已安装程序： chacl、getfacl 和 setfacl

已安装库文件：libacl.so 已安装目录：

/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

设置文件访问控制列表

libacl 包含用于操作访问控制列表的库函数

8.26. Libcap-2.73

Libcap 软件包实现了 Linux 内核中 POSIX 1003.1e 功能特性的用户空间接口。

这些功能将全能型的 root 权限划分为一组独立的权限。

预计编译时间：少于 0.1 SBU 所需磁盘空间：

3.0 MB

8.26.1. Libcap 的安装 防止安装静态库：

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

编译该软件包：

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

该 make 选项的含义：

lib=lib

该参数将库目录设置为 /usr/lib 而非 x86_64 架构下的 /usr/lib64。在 x86 架构上此参数无效。

要测试结果，请执行：

```
make test
```

安装该软件包：

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

8.26.2. Libcap 安装内容

已安装程序：	capsh、getcap、getpcaps 和 setcap
已安装的库：	libcap.so 和 libpsx.so

简要描述

capsh 用于探索和限制能力支持的 shell 封装工具

getcap 检查文件能力

getpcaps 显示查询进程的能力

setcap 设置文件能力

libcap 包含用于操作 POSIX 1003.1e 能力的库函数

libpsx 包含支持与 pthread 库相关的系统调用 POSIX 语义的函数

8.27. Libxcrypt-4.4.38

Libxcrypt 软件包包含一个用于密码单向哈希加密的现代函数库。

预计编译时间: 0.1 SBU

所需磁盘空间: 12 MB

8.27.1. Libxcrypt 的安装

准备编译 Libxcrypt:

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

新增配置选项的含义:

--enable-hashes=strong, glibc

构建适用于安全场景的强哈希算法，同时保留传统 Glibc libcrypt 提供的哈希算法以保持兼容性。

--enable-obsolete-api=no

禁用已废弃的 API 函数。对于从源代码构建的现代 Linux 系统而言，这些函数并非必需。

--disable-failure-tokens

禁用故障令牌功能。该功能是为了兼容某些平台的传统哈希库而设计，但基于 Glibc 的 Linux 系统无需此功能。

编译该软件包:

make

要测试结果，请执行:

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，因为通过源码编译安装的软件包在运行时不会链接这些函数。但已知唯一需要链接这些函数的闭源应用程序仅支持 ABI 版本 1。若您因某些闭源程序或为符合 LSB 规范必须启用这些函数，请使用以下命令重新构建该软件包:

```
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 安装内容

libcrypt.so

已安装库文件：
摘要描述

libcrypt 包含密码哈希处理函数

8.28. Shadow-4.17.3 Shadow 软件包包含以安全方式处理密码的程序。

预计构建时间：0.1 SBU 所需磁盘空间：114 MB

8.28.1. Shadow 的重要安装说明

若已安装 Linux-PAM，应遵循 BLFS 指南而非本页说明来构建（或重新构建/升级）shadow。

注意

若需强制使用高强度密码，请先安装并配置 Linux-PAM。接着安装支持 PAM 的 shadow 工具。最后安装 libpwquality 并配置 PAM 以启用该组件。

禁用 groups 程序及其手册页的安装，因为 Coreutils 提供了更优版本。同时防止安装已在 8.3 节“Man-pages-6.12”中安装过的手册页：

```
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 / /{}';
```

使用更安全的 YESCRYPT 密码加密方式替代默认的 crypt 方法，该方法还支持超过 8 字符的密码长度。同时需要将 Shadow 默认使用的过时用户邮箱路径/var/spool/mail 更新为当前通用的/var/mail 路径。此外，从 PATH 环境变量中移除/bin 和/sbin 目录，因为它们仅是/usr 下对应目录的符号链接。

警告

在 PATH 变量中包含/bin 和/或/sbin 可能导致某些 BLFS 软件包构建失败，因此不要在.bashrc 文件或其他任何地方这样做。

```
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：

```
创建 /usr/bin/passwd 文件 ./configure --  
sysconfdir=/etc \  
--disable-static \ --with-{b,yes}crypt \  
without-libbsd \ --with-group-name-max-  
length=32
```

新配置选项的含义：

创建 /usr/bin/passwd 文件

文件/usr/bin/passwd 必须存在，因为某些程序中硬编码了该路径；如果该文件尚不存在，安装脚本会在错误的位置创建它。

```
--with-{b,yes} crypt
```

Shell 会将其扩展为两个选项：--with-bcrypt 和--with-yescrypt。这些选项允许 shadow 使用 Libxcrypt 实现的 Bcrypt 和 Yescrypt 算法进行密码哈希。与传统 SHA 算法相比，这些算法更安全（尤其对基于 GPU 的攻击具有更强的抵抗力）。

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

用户名最长允许 32 个字符。将组名的最大长度设置为相同值。

```
--without-libbsd
```

不使用 LFS 中不包含的 libbsd 库的 readpassphrase 函数，转而使用内部副本。

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 软件包

该软件包包含用于添加、修改和删除用户及组的工具；设置和更改密码；以及执行其他管理任务。有关密码影子化含义的完整说明，请参阅解压后源码树中的 doc/HOWTO 文件。如果使用 Shadow 支持，请注意需要验证密码的程序（显示管理器、FTP 程序、pop3 守护进程等）必须兼容 Shadow。也就是说，它们必须能够处理影子密码。

要启用影子密码，请运行以下命令：

```
pwconv
```

要启用影子组密码，请运行：

```
grpconv
```

Shadow 对 useradd 工具的默认配置需要做些说明。首先，useradd 工具的默认行为是创建用户及与该用户同名的用户组。默认情况下用户 ID(UID)和组 ID(GID)编号从 1000 开始。这意味着如果不向 useradd 传递额外参数，每个用户都将成为系统中唯一用户组的成员。若不需要此特性，需向 useradd 传递-g 或-N 参数，或修改/etc/login.defs 中的 USERGROUPS_ENAB 设置。详见 useradd(8) 手册页。

其次，要修改默认参数，必须创建并定制/etc/default/useradd 文件以满足特定需求。创建命令如下：

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

/etc/default/useradd	参数说明
----------------------	------

```
GROUP=999
```

该参数设置/etc/group 文件中使用的组号起始值。特定的 999 值来源于上述的--gid 参数。您可以将其设置为任意期望值。请注意 useradd 程序永远不会重复使用 UID 或 GID。如果该参数指定的数字已被占用，程序将使用下一个可用数字。还需注意的是，如果系统中不存在 ID 等于该数值的组，那么首次在不带

使用 `-g` 参数时，即使账户已正确创建，仍会产生错误提示——`useradd: unknown GID 999`。这就是为什么我们在 7.6 节“创建必要文件与符号链接”中预先创建了该组 ID 的 `users` 用户组。

```
CREATE_MAIL_SPOOL=yes
```

此参数使 `useradd` 为每个新用户创建邮箱文件。`useradd` 会将此文件的属组设置为 `mail` 组，并赋予 `0660` 权限。若不想创建这些文件，可执行以下命令：

```
sed -i '/MAIL/s/yes/no/' /etc/default/useradd
```

8.28.3. 设置 root 密码 为用户 root 选择一个密码并通过运行以下命令进行设置：

```
passwd root
```

8.28.4. Shadow 软件包内容 安装的程序包括：

`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` (链接到 `newgrp`), `su`, `useradd`, `userdel`, `usermod`, `vigr` (链接到 `visudo`)

已安装目录：/etc/default 和 /usr/include/shadow

已安装库文件： libsubid.so

简要描述

`chage` 用于修改强制更改密码的最长间隔天数

`chfn` 用于修改用户全名及其他信息

`chgpasswd` 用于批量更新组密码

`chpasswd` 用于批量更新用户密码

`chsh` 用于更改用户的默认登录 shell

`expiry` 检查并强制执行当前密码过期策略

~~用田~~ 检查登录失败日志、设置账户被锁定前的最大失败次数，以及重置失败计数 `getsubids` 用于列出用户的附属 ID 范围 `gpasswd` 用于添加和删除组成员及管理员

`groupadd` 使用给定名称创建用户组

`groupdel` 删除指定名称的用户组

`groupmems` 允许用户管理自己的组成员列表，无需超级用户权限

`groupmod` 用于修改指定组的名称或组 ID

`grpck` 用于验证组文件 /etc/group 和 /etc/gshadow 的完整性

`grpconv` 从普通组文件创建或更新影子组文件

`grpunconv` 根据 /etc/gshadow 更新 /etc/group 文件，随后删除前者

登录 系统用于让用户登录的功能

logoutd 是一个守护进程，用于强制执行登录时间和端口的限制

newgidmap 用于设置用户命名空间的 gid 映射

newgrp 用于在登录会话期间更改当前 GID

newuidmap 用于设置用户命名空间的 UID 映射

newusers 用于创建或更新一系列用户账户

nologin 禁止用 ~~登录~~ 提交登录，说明账户不可用；该功能设计用于作为禁用账户的默认 shell

passwd 用于更改用户或组账户的密码

pwck 验证密码文件/etc/passwd 和/etc/shadow 的完整性

pwconv 从普通密码文件创建或更新影子密码文件

pwunconv 根据/etc/shadow 更新/etc/passwd，然后删除前者

sg 以指定组的 GID 执行给定命令

su 以替代用户和组 ID 运行 shell

useradd 创建具有指定名称的新用户，或更新默认新用户信息

userdel 删除指定的用户账户 usermod 用于修改给定用户的登录名、用户标识(UID)、shell、初始组、主目录等

vigr 编辑/etc/group 或/etc/gshadow 文件

vipw 编辑/etc/passwd 或/etc/shadow 文件

libsubid 用于处理用户和组从属 ID 范围的库

8.29. GCC-14.2.0 GCC 软件包包含 GNU 编译器集合，其中涵盖 C 和 C++ 编译器。

预计编译时间：46 SBU（含测试） 所需磁盘空间：6.3 GB

8.29.1. GCC 的安装 若在 x86_64 架构上构建，需将 64 位库的默认目录名改为"lib":

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

GCC 文档建议在专用构建目录中编译 GCC:

```
mkdir -v build
进入目录 构建
```

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 支持七种不同的计算机语言，但其中大多数语言的依赖项尚未安装。
关于如何构建 GCC 支持的所有语言，请参阅 BLFS 手册的 GCC 页面说明。

新配置参数的含义：

LD=ld

该参数使配置脚本使用本章先前构建的 Binutils 软件包安装的 ld 程序，而非默认会使用的交叉编译版本。

--disable-fixincludes

默认情况下，在安装 GCC 过程中会对某些系统头文件进行"修复"以适配 GCC。现代 Linux 系统无需此操作，且若在安装 GCC 后重新安装软件包，这种修复可能造成危害。
该选项可阻止 GCC 对头文件执行"修复"操作。

--with-system-zlib

该选项指示 GCC 链接系统安装的 Zlib 库副本，而非其内部自带的版本。

注意

PIE（位置无关可执行文件）是一种能够加载到内存任意位置的二进制程序。若不启用 PIE，名为 ASLR（地址空间布局随机化）的安全特性仅能应用于共享库，而无法作用于可执行文件本身。启用 PIE 后，除了共享库之外，可执行文件也能获得 ASLR 保护，从而缓解某些基于可执行文件中敏感代码或数据固定地址的攻击手段。

SSP（栈溢出保护）是一种确保参数栈不被破坏的技术。栈损坏可能导致子程序返回地址被篡改，从而将控制权转移至危险代码（这些代码可能存在于程序或共享库中，也可能由攻击者通过某种方式注入）。

编译该软件包：

```
make
```

重要提示

本节将 GCC 测试套件视为重要环节，但其耗时较长。建议首次构建者运行该测试套件。通过在下方 make -k check 命令中添加-jx 参数（x 代表系统 CPU 核心数），可显著缩短测试运行时间。

GCC 在编译某些极其复杂的代码模式时可能需要更大的栈空间。为防止宿主发行版的栈空间限制过小，我们预先将栈大小硬限制设为无限。多数宿主发行版（以及最终的 LFS 系统）默认硬限制本就是无限，但显式设置并无害处。无需修改栈空间软限制，因为只要不超过硬限制，GCC 会自动将其设为适当值：

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

现在移除/修复若干已知的测试失败项：

```
sed -e '/cpython/d' -i ..../gcc/testsuite/gcc.dg/plugin/plugin.exp
sed -e 's/no-pic /&-no-pie /' -i ..../gcc/testsuite/gcc.target/i386/pr113689-1.c
sed -e 's/300000/(1|300000)/' -i ..../libgomp/testsuite/libgomp.c-c++-common/pr109062.c
sed -e 's/{ target nonpic } //'\'
-e '/GOTPCREL/d' -i ..../gcc/testsuite/gcc.target/i386/fentryname3.c
```

以非特权用户身份测试结果，但遇到错误时不要停止：

```
chown -R tester .
```

以测试用户身份执行带环境变量的检查命令：

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

要提取测试套件结果的摘要，请运行：

```
..../contrib/test_summary
```

若只需过滤摘要内容，可将输出通过管道传递给 grep -A7 Summ 命令

可将结果与以下网址的日志进行对比：<https://www.linuxfromscratch.org/lfs/build-logs/12.3/> 和 <https://gcc.gnu.org/ml/gcc-testresults/>。

已知 tsan 测试在某些宿主发行版上会失败。

少量意外失败总是难以完全避免。某些情况下测试失败与系统具体硬件相关。只要测试结果与上述网址差异不大，即可安全继续。

安装该软件包：

```
make install
```

GCC 构建目录现在归 tester 所有，而已安装头文件目录（及其内容）的所有权不正确。将其所有权更改为 root 用户和组：

```
chown -v -R root:root \
/usr/lib/gcc/$(gcc -dumpmachine)/14.2.0/include{,-fixed}
```

出于“历史遗留”原因，按照 FHS 规范创建一个符号链接。

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

许多软件包使用名称 cc 来调用 C 编译器。我们已在 gcc-pass2 阶段创建了 cc 作为符号链接，现在同样为其创建手册页的符号链接：

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

添加兼容性符号链接以支持使用链接时优化(LTO)构建程序：

```
ln -sfv ../../libexec/gcc/$(gcc -dumpmachine)/14.2.0/liblto_plugin.so \
/usr/lib/bfd-plugins/
```

现在我们的最终工具链已就位，再次确保编译和链接能按预期工作非常重要。我们通过执行一些完整性检查来实现这一点：

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

不应出现任何错误，且最后一条命令的输出应为（允许动态链接器名称存在平台差异）：

[请求的程序解释器：/lib64/ld-linux-x86-64.so.2]

现在确认我们已配置使用正确的启动文件：

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

最后一条命令的输出应为：

```
/usr/lib/gcc/x86_64-pc-linux-gnu/14.2.0/../../../../lib/Scrt1.o 成功 /usr/lib/gcc/x86_64-pc-linux-gnu/14.2.0/../../../../lib/crti.o 成功 /usr/lib/gcc/x86_64-pc-linux-gnu/14.2.0/../../../../lib/crtn.o 成功
```

根据您的机器架构，上述路径可能略有不同。主要差异在于/usr/lib/gcc 后面的目录名称。这里需要重点确认的是 gcc 在/usr/

lib 目录下找到了所有三个 crt*.o 文件。

验证编译器是否在搜索正确的头文件：

```
grep -B4 '^ /usr/include' dummy.log
```

该命令应返回以下输出：

```
#include <...> 搜索从这里开始: /usr/lib/gcc/x86_64-pc-linux-gnu/14.2.0/include /usr/local/include /usr/lib/gcc/x86_64-pc-linux-gnu/14.2.0/include-fixed /usr/include
```

再次说明，以您的目标三元组命名的目录可能与上述不同，具体取决于您的系统架构。

接下来，验证新链接器是否使用了正确的搜索路径：

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

包含"-linux-gnu"组件的路径引用应被忽略，除此之外，最后一条命令的输出应为：

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

32 位系统可能会使用其他几个目录。例如，以下是一台 i686 机器的输出：

```
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");
```

接下来确保我们使用的是正确的 libc 库：

在虚拟日志中查找"/lib.*/libc.so.6"

最后一条命令的输出应为：

尝试打开/usr/lib/libc.so.6 成功

确保 GCC 使用正确的动态链接器：

在虚拟日志中查找到匹配项

最后一条命令的输出应为（允许动态链接器名称存在平台差异）：

在/usr/lib/ld-linux-x86-64.so.2 找到 ld-linux-x86-64.so.2

若输出与上述不符或未显示任何结果，则表明存在严重错误。请检查并回溯操作步骤以定位问题所在并进行修正。必须解决所有问题后才能继续后续流程。

确认一切运行正常后，清理测试文件：

删除测试文件：

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

最后，移动位置不当的文件：

创建目标目录并移动调试脚本：

```
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 安装内容

已安装程序:

c++、cc (链接至 gcc)、cpp、g++、gcc、gcc-ar、gcc-nm、gcc-ranlib、gcov、gcov-dump、gcovtool 以及 lto-dump

已安装库文件:

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、/usr/include/c++、/usr/lib/gcc、/usr/libexec/gcc 和 /usr/share/gcc-14.2.0

已安装的目录:

简要描述

c++	C++ 编译器
cc	C 编译器
cpp	C 预处理器；编译器用它来展开源文件中的 #include、#define 等指令
g++ C++ 编译器	
gcc	C 编译器
gcc-ar	一个封装了 ar 命令的包装器，用于向命令行添加插件。该程序仅用于实现“链接时优化”，在默认构建选项下并无实际用途。
gcc-nm	一个封装了 nm 命令的包装器，用于向命令行添加插件。该程序仅用于实现“链接时优化”，在默认构建选项下并无实际用途。
gcc-ranlib	一个封装了 ranlib 命令的包装器，用于向命令行添加插件。该程序仅用于实现“链接时优化”，在默认构建选项下并无实际用途。
gcov	覆盖率测试工具；用于分析程序以确定优化措施将产生最大效果的区域
gcov-dump	gcda 和 gcno 离线配置文件转储工具
gcov-tool	gcda 离线配置文件处理工具
lto-dump	用于转储 GCC 启用 LTO 后生成的目标文件的工具
libasan	地址消毒器运行时库
libatomic	GCC 原子操作内置运行时库
libcc1	允许 GDB 利用 GCC 功能的库
libgcc	包含对 gcc 的运行时支持
libgcov	当 GCC 被指示启用性能分析时，该库会被链接到程序中
libgomp	GNU 为 C/C++ 和 Fortran 语言实现的多平台共享内存并行编程 OpenMP API
libhwasan	硬件辅助地址消毒器运行时库
libitm	GNU 事务内存库
liblsan	内存泄漏检测器运行时库
liblto_plugin	GCC 的 LTO 插件使 Binutils 能够处理由启用 LTO 的 GCC 生成的目标文件
libquadmath	数学库
	GCC 四精度数学库 API

libssp

包含支持 GCC 栈溢出保护功能的例程。通常不会使用它，因为 Glibc 也提供了这些例程。

libstdc++

标准 C++ 库

libstdc++exp

实验性 C++ 合约库

libstdc++fs

ISO/IEC TS 18822:2015 文件系统库

libsupc++

为 C++ 编程语言提供支持例程

libtsan

线程消毒器运行时库

libubsan

未定义行为检测器运行时库

8.30. Ncurses-6.5 Ncurses 软件包包含用于终端无关字符屏幕处理的函数库。

预计编译时间：0.2 SBU 所需磁盘空间：46 MB

8.30.1. 安装 Ncurses 准备编译 Ncurses：

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

新增配置选项的含义：

--with-shared

该选项使 Ncurses 构建并安装共享的 C 语言库。

--without-normal

该选项阻止 Ncurses 构建并安装静态的 C 语言库。

--without-debug

这将阻止 Ncurses 构建和安装调试库。

--with-cxx-shared

该选项使 Ncurses 构建并安装共享的 C++ 绑定库，同时会阻止构建和安装静态的 C++ 绑定库。

--enable-pc-files

此选项会生成并安装用于 pkg-config 的.pc 文件。

编译该软件包：

make

该软件包包含测试套件，但只能在安装完成后运行。测试文件位于 test/ 目录下。详见该目录下的 README 文件说明。

安装此软件包将直接覆盖原有的 libncursesw.so.6.5 文件，可能导致正在使用该库文件代码和数据的 shell 进程崩溃。建议使用 DESTDIR 参数安装，并通过 install 命令正确替换库文件（同时修改 curses.h 头文件以确保使用宽字符 ABI，如我们在 6.3 节“Ncurses-6.5”中所做的那样）：

```
make DESTDIR=$PWD/dest install install -v -m755
dest/usr/lib/libncursesw.so.6.5 /usr/lib rm -v
dest/usr/lib/libncursesw.so.6.5 sed -e 's/^#if.*XOPEN.*$/#if 1/' \
-i dest/usr/include/curses.h cp -av
dest/* /
```

许多应用程序仍期望链接器能够找到非宽字符版本的 Ncurses 库。通过符号链接欺骗这些应用程序链接宽字符库（注意 .so 链接仅在 curses.h 被修改为始终使用宽字符 ABI 时才是安全的）：

```
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，确保那些在构建时查找-lcurses 的老旧应用程序仍能正常构建：

```
ln -sfv libncursesw.so /usr/lib/libcurses.so
```

如需安装 Ncurses 文档：

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

注意

上述操作不会生成非宽字符版的 Ncurses 库，因为从源码编译安装的软件包在运行时都不会链接这些库。不过目前已知仅依赖非宽字符版 Ncurses 库的闭源程序都需要 5.x 版本。若您因某些闭源程序或为符合 LSB 规范必须使用这类库，请通过以下命令重新编译该软件包：

```
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 安装内容

已安装程序：
已安装库文件：

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
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

已安装的目录：/usr/share/tabset、/usr/share/terminfo 和 /usr/share/doc/ncurses-6.5

简要描述

captoinfo 将 termcap 描述转换为 terminfo 描述

clear 如果可能的话，清空屏幕

infocmp 比较或打印 terminfo 描述

infotocap 将 terminfo 描述转换为 termcap 描述

ncursesw6-config 提供 ncurses 的配置信息

reset 将终端重新初始化为默认值

tabs 清除并设置终端的制表位

终端信息编译器	将 terminfo 文件从源格式转换为 ncurses 库例程所需的二进制格式的终端信息条目描述编译器[terminfo 文件包含特定终端的功能信息]
终端类型枚举器	列出所有可用终端类型，显示每个类型的主名称和描述
tput	将终端相关功能的值提供给 shell 使用；也可用于重置或初始化终端，或报告其完整名称
tset	可用于初始化终端
libncursesw	包含在终端屏幕上以多种复杂方式显示文本的函数；内核执行 make menuconfig 时显示的菜单就是这些函数应用的一个典型示例
libncurses++w	包含针对本软件包中其他库的 C++ 绑定接口
libformw	包含实现表单功能的函数
libmenuw	包含实现菜单功能的函数
libpanelw	包含实现面板功能的函数

8.31. Sed-4.9 Sed 软件包包含一个流编辑器。

预计编译时间: 0.3 SBU 所需磁盘空间: 30 MB

8.31.1. Sed 的安装 准备编译 Sed:

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

编译该软件包并生成 HTML 文档:

```
make  
make html
```

要测试结果, 请执行:

```
chown -R tester .  
以测试用户身份执行检查命令: "PATH=$PATH make check"
```

安装软件包及其文档:

```
make install  
创建文档目录并设置权限: install -d -m755 /usr/share/doc/sed-4.9 复制文档文件: install -m644 doc/sed.html /usr/share/doc/sed-4.9
```

8.31.2. Sed 软件包安装内容 已安装程序:

sed
安装目录: /usr/share/doc/sed-4.9

简要描述

sed 单次遍历过滤和转换文本文件

8.32. Psmisc-23.7 Psmisc 软件包包含用于显示运行进程信息的程序。

预计构建时间：少于 0.1 SBU 所需磁盘空间：
6.7 MB

8.32.1. 安装 Psmisc 准备编译 Psmisc：

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

编译该软件包：

```
make
```

要运行测试套件，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.32.2. Psmisc 安装内容包含的程序：

fuser、killall、peekfd、prtstat、pslog、pstree 及 pstree.x11（指向 pstree 的链接）

简要描述

fuser 报告使用指定文件或文件系统的进程 ID (PID)

killall 按名称终止进程；该命令会向所有运行指定命令的进程发送信号

peekfd 查看指定 PID 运行进程的文件描述符

prtstat 打印进程相关信息

pslog 报告进程当前日志路径

pstree 以树状结构显示正在运行的进程

pstree.x11 功能与 pstree 相同，但在退出前会等待确认

8.33. Gettext-0.24 Gettext 软件包包含国际化和本地化工具。这些工具允许程序在编译时加入 NLS（本地语言支持）功能，使其能够以用户母语输出消息。

预计编译时间：1.7 SBU

所需磁盘空间：290 MB

8.33.1. Gettext 的安装

准备编译 Gettext：

```
./configure --prefix=/usr \
--disable-static \
--docdir=/usr/share/doc/gettext-0.24
```

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

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

8.33.2. Gettext 安装内容 已安装程序：

已安装库文件：
autopoint、envsubst、gettext、gettext.sh、gettextize、msgattrib、msgcat、msgcmp、msgcomm、msgconv、msgen、msgexec、msgfilter、msgfmt、msggrep、msginit、msgmerge、msgunfmt、msguniq、ngettext、recode-sr-libasprintf.so、libgettextlib.so、libgettextpo.so、libgettextsrc.so、libtextstyle.so 和 preloadable_libintl.so

已安装的目录：
/usr/lib/gettext、/usr/share/doc/gettext-0.24、/usr/share/gettext 以及 /usr/share/gettext-0.24

简要描述

autopoint 将标准 Gettext 基础设施文件复制到源代码包中

envsubst 替换 shell 格式字符串中的环境变量

gettext 通过查询消息目录中的翻译，将自然语言消息转换为用户所使用的语言

gettext.sh 主要作为 gettext 的 shell 函数库使用

gettextize 将标准 Gettext 文件复制到软件包的指定顶层目录，开始国际化处理

msgattrib 根据消息属性筛选翻译目录中的消息，并操作这些属性

msgcat 合并并整合给定的.po 文件

msgcmp 比较两个.po 文件，检查它们是否包含相同的 msgid 字符串集

msgcomm	查找给定.po 文件中共同的消息
msgconv	将翻译目录转换为不同的字符编码
msgen	创建英文翻译目录
msgexec	对翻译目录中的所有译文执行命令
消息过滤器	对翻译目录中的所有译文应用过滤器
msgfmt 从翻译目录生成二进制消息目录	
msggrep	提取翻译目录中与给定模式匹配或属于某些指定源文件的所有消息
msginit	创建新的.po 文件，使用用户环境中的值初始化元信息
msgmerge 将两个原始翻译合并为单个文件	
msgunfmt 将二进制消息目录反编译为原始翻译文本	
msguniq 在翻译目录中合并重复的翻译项	
ngettext 显示根据数字变化语法形式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母转写为拉丁字母	
xgettext 从给定源文件中提取可翻译的字符串信息，生成首个翻译模板	
libasprintf	定义 autosprintf 类，使 C 语言格式化输出例程可在 C++ 程序中使用，适用于字符串和流
libgettextlib	包含各种 Gettext 程序使用的公共例程；这些例程不适用于通用场景
libgettextpo	用于编写处理.po 文件的专用程序；当 Gettext 自带的标准工具（如 msgcomm、msgcmp、msgattrib 和 msigen）无法满足需求时，会使用此库
libgettextsrc	提供各类 Gettext 程序使用的通用例程；这些例程不适用于常规用途
libtextstyle	文本样式库
preloadable_libintl	一个旨在通过 LD_PRELOAD 使用的库，用于帮助 libintl 记录未翻译的消息

8.34. Bison-3.8.2 Bison 软件包包含一个解析器生成器。

预计编译时间：2.1 SBU 所需磁盘空间：62 MB

8.34.1. Bison 的安装 准备编译 Bison：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.34.2. Bison 安装内容

bison 和 yacc

~~已安装库文件~~：liby.a 已安装目录：

/usr/share/bison

简要描述

bison 根据一系列规则生成用于分析文本文件结构的程序；Bison 是替代品为 Yacc（又一个编译器编译器）

yacc 是 bison 的一个封装器，用于那些仍调用 yacc 而非 bison 的程序；它会以 -y 选项调用 bison liby 包含 Yacc 兼容的 yyerror 和 main 函数实现的 Yacc 库；该库通常不太实用，但 POSIX 规范要求提供它

8.35. Grep-3.11 Grep 软件包包含用于搜索文件内容的程序。

预计编译时间：0.4 SBU 所需磁盘空间：39 MB

8.35.1. 安装 Grep 首先移除关于使用 egrep 和 fgrep 的警告，该警告会导致某些软件包的测试失败：

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

准备编译 Grep：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.35.2. 已安装的 Grep 程序内容：

egrep、fgrep 和 grep

简要描述

egrep 打印匹配扩展正则表达式的行。已废弃，请改用 grep -E
fgrep 打印匹配固定字符串列表的行。已废弃，请改用 grep -F

grep 打印匹配基本正则表达式的行

8.36. Bash-5.2.37 Bash 软件包包含 Bourne-Again Shell (Bash shell)。

预计编译时间：1.4 SBU 所需磁盘空间：53 MB

8.36.1. Bash 的安装 为编译 Bash 做准备：

```
./configure --prefix=/usr \
--without-bash-malloc \
--with-installed-readline \
--docdir=/usr/share/doc/bash-5.2.37
```

新配置选项的含义：

--with-installed-readline

该选项指示 Bash 使用系统上已安装的 readline 库，而非其自带的 readline 版本。

编译该软件包：

make

若不运行测试套件，可直接跳转至“安装软件包”步骤。

准备测试前，请确保测试用户对源码树具有写入权限。

chown -R tester .

该软件包的测试套件设计为由拥有标准输入终端的非 root 用户运行。为满足此要求，需使用 Expect 生成一个新的伪终端，并以测试用户身份运行测试：

```
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 工具检测测试脚本输出与预期输出之间的差异。任何来自 diff 的输出（以<和>为前缀）均表示测试失败，除非有消息说明该差异可被忽略。已知名为 run-builtins 的测试在某些主机发行版上会失败，表现为输出首行存在差异。

安装该软件包：

make install

运行新编译的 bash 程序（替换当前正在执行的版本）：

执行 /usr/bin/bash --login

8.36.2. 已安装的 Bash 程序内容：

bash、bashbug 以及 sh (链接到 bash)

安装目录：/usr/include/bash、/usr/lib/bash 和 /usr/share/doc/bash-5.2.37

简要描述

bash 广泛使用的命令解释器；它在执行给定命令行前会进行多种类型的扩展和替换，这使得该解释器成为一个强大的工具。³⁷ `bashbug` 是一个帮助用户编写并发送标准格式 bash 错误报告的 shell 脚本

sh 指向 bash 程序的符号链接；当以 sh 名称调用时，bash 会尽可能模拟历史版本 sh 的启动行为，同时遵循 POSIX 标准

8.37. Libtool-2.5.4 Libtool 软件包包含 GNU 通用库支持脚本。它通过提供一致且可移植的接口，简化了共享库的使用。

预计编译时间：0.6 SBU

所需磁盘空间：44 MB

8.37.1. 安装 Libtool

准备编译 Libtool：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

移除无用的静态库：

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

8.37.2. Libtool 安装内容

libtool 和 libtoolize

~~已安装程序~~： libltdl.so

/usr/include/libltdl 和 /usr/share/libtool

已安装目录：

~~简要描述~~

libtool 提供通用的库构建支持服务

libtoolize 提供向软件包添加 libtool 支持的标准方法

libltdl 隐藏了动态加载库的各种困难

8.38. GDBM-1.24 GDBM 软件包包含 GNU 数据库管理器。这是一个采用可扩展哈希算法的数据库函数库，其功能类似于标准 UNIX dbm。该库提供存储键值/数据对、通过键值搜索检索数据以及删除键值及其关联数据的基础功能。

预计编译时间：少于 0.1 SBU

所需磁盘空间：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

该选项用于构建 libgdbm 兼容库。LFS 之外的部分软件包可能需要它提供的旧版 DBM 例程。

编译该软件包：

make

要测试结果，请执行：

make check

安装该软件包：

make install

8.38.2. GDBM 安装内容

已安装程序：

gdbm_dump、gdbm_load 和 gdbmtool

已安装库文件：

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 包含操作哈希数据库的函数库

libgdbm_compat 兼容性函数库，包含旧版 DBM 函数

8.39. Gperf-3.1 Gperf 可根据关键字集合生成完美哈希函数。

预计编译时间：少于 0.1 SBU
所需磁盘空间：
6.1 MB

8.39.1. 安装 Gperf 准备编译 Gperf：

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

编译该软件包：

```
make
```

已知在同时运行多个测试（-j 选项值大于 1）时会导致测试失败。要测试结果，请执行：

```
make -j1 check
```

安装该软件包：

```
make install
```

8.39.2. Gperf 安装内容

已安装程序：
gperf
安装目录：
/usr/share/doc/gperf-3.1

简要描述

gperf 从键集生成完美哈希

8.40. Expat-2.6.4 Expat 软件包包含一个面向流的 C 语言 XML 解析库。

预计编译时间: 0.1 SBU
所需磁盘空间: 14 MB

8.40.1. 安装 Expat 准备编译 Expat:

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

编译该软件包:

```
make
```

要测试结果, 请执行:

```
make check
```

安装该软件包:

```
make install
```

如需安装文档, 请执行:

将文档文件 (*.{html,css}) 安装至 /usr/share/doc/expat-2.6.4 目录, 设置权限为 644

8.40.2. Expat 安装内容

xmlwf 程序

已安装库文件: libexpat.so

已安装目录: /usr/share/doc/expat-2.6.4

简要描述

xmlwf 这是一个用于检查 XML 文档格式是否良好的非验证性工具

libexpat 包含用于解析 XML 的 API 函数

8.41. Inetutils-2.6 Inetutils 软件包包含基础网络工具程序

预计编译时间：0.2 SBU 所需磁盘空间：35 MB

8.41.1. Inetutils 的安装 首先使该软件包支持 gcc-14.1 或更高版本编译：

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

为编译 Inetutils 做准备：

```
./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

该选项阻止 Inetutils 安装 logger 程序（脚本通常使用该程序向系统日志守护进程传递消息）。由于 Util-linux 软件包会安装更新的版本，因此无需安装此组件。

--disable-whois

该选项禁用构建过时的 Inetutils whois 客户端。BLFS 手册中提供了更优 whois 客户端的安装指南。

--disable-r*

这些参数禁用构建因安全问题不应使用的过时程序。这些程序提供的功能可由 BLFS 手册中的 openssh 软件包替代实现。

--disable-servers

该选项禁用安装 Inetutils 软件包中包含的各种网络服务器。这些服务器被认为不适合基础 LFS 系统。其中一些服务器本质上就不安全，仅在可信网络中才被视为安全。请注意，其中许多服务器都有更好的替代方案。

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

将程序移动到正确位置：

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

8.41.2. Inetutils 安装内容

已安装程序： dnsdomainname、ftp、ifconfig、hostname、ping、ping6、talk、telnet、tftp 和 traceroute
简要描述

dnsdomainname 显示系统的 DNS 域名

ftp 是文件传输协议程序

主机名 报告或设置主机名称

ifconfig 管理网络接口

ping 发送回显请求数据包并报告响应耗时

ping6 专用于 IPv6 网络的 ping 版本

talk 用于与其他用户聊天

telnet TELNET 协议的接口程序

tftp 简易文件传输程序

traceroute 路由追踪追踪数据包从您所在主机到网络中另一主机的传输路径，显示沿途所有中间跃点（网关）

8.42. Less-668 Less 软件包包含一个文本文件查看器。

预计编译时间：少于 0.1 SBU 所需磁盘空间：14 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

该选项指示软件包创建的程序在/etc 目录下查找配置文件。

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.42.2. 已安装的 Less 程序内容：

less、lessecho 和 lesskey

简要描述

less 文件查看器或分页器；它显示给定文件的内容，允许用户滚动、查找字符串并跳转到标记 lessecho 用于在 Unix 系统中扩展文件名中的元字符，如*和? lesskey 用于指定 less 的按键绑定

8.43. Perl-5.40.1 Perl 软件包包含实用提取与报表语言。

预计编译时间：1.3 SBU 所需磁盘空间：245 MB

8.43.1. Perl 的安装 此版本 Perl 会构建 Compress::Raw::Zlib 和 Compress::Raw::BZip2 模块。默认情况下 Perl 会使用内置的源代码进行构建。执行以下命令可使 Perl 使用系统中已安装的库：

系统：

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

要完全掌控 Perl 的配置方式，可以从以下命令中移除"-des"选项，手动选择该软件包的构建方式。或者直接使用如下所示的命令，采用 Perl 自动检测的默认配置：

```
sh Configure -des
-D prefix=/usr \
-D vendorprefix=/usr \
-D privlib=/usr/lib/perl5/5.40/core_perl \
-D archlib=/usr/lib/perl5/5.40/core_perl \
-D sitelib=/usr/lib/perl5/5.40/site_perl \
-D sitearch=/usr/lib/perl5/5.40/site_perl \
-D vendorlib=/usr/lib/perl5/5.40/vendor_perl \
-D vendorarch=/usr/lib/perl5/5.40/vendor_perl \
-D man1dir=/usr/share/man/man1 \
-D man3dir=/usr/share/man/man3 \
-D pager="/usr/bin/less -isR" \
-D useshrplib \
-D usethreads
```

新配置选项的含义：

-D pager="/usr/bin/less -isR"

这确保使用 less 而非 more。

-D man1dir=/usr/share/man/man1 \
-D man3dir=/usr/share/man/man3

由于 Groff 尚未安装，Configure 将不会为 Perl 生成手册页。这些参数将覆盖此默认行为。

-D usethreads

构建支持线程的 Perl。

编译该软件包：

make

要测试结果，请执行：

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

安装软件包并清理：

```
make install
取消设置 BUILD_ZLIB 和 BUILD_BZIP2
```

8.43.2. 已安装的 Perl 内容

已安装程序：

corelist、cpan、enc2xs、encguess、h2ph、h2xs、instmodsh、json_pp、libnetcfg、perl、perl5.40.1 (perl 的硬链接)、perlbug、perldoc、perlivp、perlthanks (perlbug 的硬链接)、piconv、pl2pm、pod2html、pod2man、pod2text、pod2usage、podchecker、podselect、prove、ptar、ptardiff、

已安装库文件：

数不胜数；无法在此全部列出 /usr/lib/perl5

简要描述

corelist	Module::CoreList 的命令行前端工具
cpan	通过命令行与综合 Perl 存档网络 (CPAN) 进行交互
enc2xs	根据 Unicode 字符映射或 Tcl 编码文件为 Encode 模块构建 Perl 扩展
encguess	猜测一个或多个文件的编码类型
h2ph	将.h 格式的 C 语言头文件转换为.ph 格式的 Perl 头文件
h2xs	将.h 格式的 C 语言头文件转换为 Perl 扩展模块
instmodsh	用于检查已安装 Perl 模块的 Shell 脚本；可从已安装模块创建压缩包
json_pp	在特定输入和输出格式之间转换数据
libnetcfg	可用于配置 Perl 模块 libnet
perl	融合了 C、sed、awk 和 sh 等语言诸多优秀特性的瑞士军刀式编程语言
perl5.40.1	指向 perl 的硬链接
perlbug	用于生成关于 Perl 或其自带模块的错误报告，并通过邮件发送
perldoc	显示嵌入在 Perl 安装目录或 Perl 脚本中的 pod 格式文档
perlivp	Perl 安装验证程序；可用于验证 Perl 及其库是否已正确安装
perlthanks	用于生成感谢信息并发送给 Perl 开发者
piconv	字符编码转换工具 iconv 的 Perl 实现版本
pl2pm	一个将 Perl4 的.pl 文件转换为 Perl5 的.pm 模块的简易工具
pod2html	将文件从 pod 格式转换为 HTML 格式
pod2man	将 pod 数据转换为格式化*roff 输入
pod2text	将 pod 数据转换为格式化 ASCII 文本
pod2usage	从文件内嵌的 pod 文档中打印使用说明
podchecker	检查 pod 格式文档文件的语法
podselect	显示 pod 文档的选定部分
prove	用于针对 Test::Harness 模块运行测试的命令行工具
ptar	用 Perl 编写的类 tar 程序
ptardiff	一个比较已解压存档与未解压存档的 Perl 程序

ptargrep	一个 Perl 程序，用于对 tar 归档文件中文件内容进行模式匹配
shasum	打印或校验 SHA 校验和
splain	用于强制 Perl 输出详细的警告诊断信息
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 开发的 XML 解析器 Expat 的 Perl 接口。

预计构建时间：少于 0.1 SBU

所需磁盘空间：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
```

要测试结果，请执行：

```
make test
```

安装该软件包：

```
make install
```

8.44.2. 已安装的 XML::Parser 模块内容：

Expat.so

简要描述

Expat 提供 Perl 的 Expat 接口

8.45. Intltool-0.51.0

Intltool 是一个国际化工具，用于从源文件中提取可翻译字符串。

预计编译时间：少于 0.1 SBU

所需磁盘空间：1.5 MB

8.45.1. 安装 Intltool

首先修复由 perl-5.22 及更高版本引起的警告：

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

注意

上述正则表达式由于包含大量反斜杠而显得不同寻常。其作用是在序列 '\${' 中的右花括号前添加反斜杠，最终生成 '\$\{'。

准备编译 Intltool：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

执行安装命令：

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

8.45.2. Intltool 安装内容

已安装程序：

intltool-extract、intltool-merge、intltool-prepare、intltool-update 和 intltoolize
 已安装的目录：
 /usr/share/doc/intltool-0.51.0 和 /usr/share/intltool

简要描述

intltoolize	为软件包启用 intltool 工具做准备
intltool-extract	生成可被 gettext 读取的头文件
intltool-merge	将翻译后的字符串合并到各类文件中
intltool-prepare	更新 pot 文件并将其与翻译文件合并
intltool-update	更新 po 模板文件并将其与翻译内容合并

8.46. Autoconf-2.72 Autoconf 软件包包含用于生成能自动配置源代码的 shell 脚本的程序。

预计编译时间：少于 0.1 SBU（含测试约 0.4 SBU）

所需磁盘空间： 25 MB

8.46.1. Autoconf 的安装准备

为编译 Autoconf 做准备：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.46.2. Autoconf 的安装内容

已安装的程序： autoconf、autoheader、autom4te、autoreconf、autoscan、autoupdate 和 ifnames
安装目录： /usr/share/autoconf

简要描述

autoconf 生成能自动配置软件源代码包以适应多种类 Unix 系统的 shell 脚本；其生成的配置脚本具有独立性——运行这些脚本无需 autoconf 程序

autoheader 为 configure 创建 C 语言#define 声明模板文件的工具

autom4te M4 宏处理器的封装程序

自动重新配置 自动按正确顺序运行 autoconf、autoheader、aclocal、automake、gettextize 和 libtoolize
当 autoconf 和 automake 模板文件发生变更时，可节省时间

自动扫描 帮助为软件包创建 configure.in 文件；它会检查目录树中的源文件，寻找常见的可移植性问题，并生成一个作为初步 configure.in 文件的 configure.scan 文件

autoupdate 将仍使用旧名称调用 autoconf 宏的 configure.in 文件修改为使用当前宏名称

ifnames 在编写软件包的 configure.in 文件时提供帮助；它会打印出该包在 C 预处理器条件中使用的标识符[如果某个软件包已经具备一定的可移植性设置，该程序可帮助确定 configure 需要检查的内容。它还能填补由 autoscan 生成的 configure.in 文件中的遗漏项。]

8.47. Automake-1.17

Automake 软件包包含用于生成与 Autoconf 配套使用的 Makefile 的程序。

预计构建时间：少于 0.1 SBU（含测试约 1.1 SBU）

所需磁盘空间： 121 MB

8.47.1. 安装 Automake

准备编译 Automake：

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.17
```

编译该软件包：

```
make
```

由于单个测试存在内部延迟，即使逻辑核心数较少的系统，使用四个并行作业也能加速测试。要测试结果，请运行：

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

若不想使用全部核心，请将\$((...))替换为你希望使用的逻辑核心数量。

安装该软件包：

```
make install
```

8.47.2. Automake 安装内容

已安装程序：

aclocal、aclocal-1.17（与 aclocal 硬链接）、automake 以及 automake-1.17（与 automake 硬链接）

已安装的目录：

/usr/share/aclocal-1.17、/usr/share/automake-1.17 以及 /usr/share/doc/automake-1.17

简要描述

aclocal

根据 configure.in 文件内容生成 aclocal.m4 文件

aclocal-1.17

指向 aclocal 的硬链接

automake

一个用于从 Makefile.am 文件自动生成 Makefile.in 文件的工具[要为软件包创建所有 Makefile.in 文件，请在顶层目录运行此程序。通过扫描 configure.in 文件会自动查找每个对应的 Makefile.am 文件，并生成相应的 Makefile.in 文件。]

automake-1.17 automake 的硬链接

8.48. OpenSSL-3.4.1 OpenSSL 软件包包含与密码学相关的管理工具和库。这些工具和库可用于为其他软件包（如 OpenSSH、电子邮件应用程序和网页浏览器）提供加密功能，例如访问 HTTPS 网站。

预计编译时间：1.8 SBU

所需磁盘空间：920 MB

8.48.1. OpenSSL 的安装

准备编译 OpenSSL：

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

编译该软件包：

make

要测试结果，请执行：

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

已知有一个测试项 30-test_afalg.t 会失败，若宿主机内核未启用 CONFIG_CRYPTO_USER_API_SKCIPHER 配置，或未启用任何提供 AES-CBC 实现的选项（例如同时启用 CONFIG_CRYPTO_AES 和 CONFIG_CRYPTO_CBC，或当 CPU 支持 AES-NI 时启用 CONFIG_CRYPTO_AES_NI_INTEL）。该测试失败可安全忽略。

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称中，以与其他软件包保持一致：

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-3.4.1
```

如需安装额外文档，可执行以下操作：

将文档复制到指定目录：

```
cp -vfr doc/* /usr/share/doc/openssl-3.4.1
```

注意

当修复安全漏洞的新版 OpenSSL 发布时，您应及时更新。自 OpenSSL 3.0.0 起，版本号采用主版本号.次版本号.补丁号 (MAJOR.MINOR.PATCH) 格式。相同主版本号可确保 API/ABI 兼容性。由于 LFS 仅安装共享库，当升级至相同主版本号的新版本时，无需重新编译链接 libcrypto.so 或 libssl.so 的软件包。

但需注意：所有正在运行且关联这些库的程序必须停止后重启。具体操作请参阅第 8.2.1 节“升级注意事项”中的相关条目。

8.48.2. OpenSSL 安装内容

已安装程序: c_rehash 和 openssl

已安装库文件: libcrypto.so 和 libssl.so

已安装的目录: 安装目录: /etc/ssl、/usr/include/openssl、/usr/lib/engines 以及
/usr/share/doc/openssl-3.4.1

简要描述

c_rehash

是一个 Perl 脚本，用于扫描目录中的所有文件并为其哈希值创建符号链接。c_rehash 的使用已被视为过时，应替换为 openssl rehash 命令

openssl

是一个命令行工具，用于通过 shell 调用 OpenSSL 加密库的各种密码学功能。该工具可实现多种功能，具体说明详见 openssl(1) 手册页

libcrypto.so

实现了多种互联网标准中使用的加密算法。该库提供的服务被 OpenSSL 的 SSL、TLS 和 S/MIME 实现所采用，同时也用于实现 OpenSSH、OpenPGP 等其他加密标准

libssl.so

实现了传输层安全协议(TLS v1)。它提供了丰富的 API 接口，相关文档可在 ssl(7) 手册页查阅

8.49. Elfutils-0.192 中的 Libelf 库 Libelf 是一个用于处理 ELF（可执行与可链接格式）文件的库。

预计编译时间：0.3 SBU
所需磁盘空间：135 MB

8.49.1. Libelf 的安装 Libelf 是 elfutils-0.192 软件包的组成部分。请使用 elfutils-0.192.tar.bz2 文件作为源码包。

为 Libelf 编译做准备：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

仅安装 Libelf：

```
make -C libelf install install -v m64 config/libelf.pc
/usr/lib/pkgconfig rm /usr/lib/libelf.a
```

8.49.2. Libelf 安装内容 已安装库：

libelf.so	
安装目录：	/usr/include/elfutils

简要描述

libelf.so	包含用于处理 ELF 目标文件的 API 函数
-----------	-------------------------

8.50. Libffi-3.4.7 Libffi 库为各种调用约定提供了可移植的高级编程接口。这使得程序员能够在运行时调用由调用接口描述指定的任何函数。

FFI 代表外部函数接口（Foreign Function Interface）。通过 FFI，用一种语言编写的程序可以调用另一种语言编写的程序。具体而言，Libffi 能够在解释器（如 Perl 或 Python）与用 C 或 C++ 编写的共享库子程序之间建立桥梁。

预计构建时间：1.7 SBU

所需磁盘空间：11 MB

8.50.1. Libffi 的安装说明

与 GMP 类似，Libffi 会针对当前使用的处理器进行特定优化编译。若需为其他系统构建，请将下列命令中的 `--with-gcc-arch=` 参数值更改为目标系统 CPU 完全支持的架构名称。若不进行此操作，所有链接到 libffi 的应用程序都将触发非法操作错误。

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

确保 GCC 针对当前系统进行优化。若未指定此参数，系统将自行猜测，生成的代码可能不正确。若需将生成的代码从原生系统复制至性能较低的系统，请将性能较低的系统作为参数。有关替代系统类型的详细信息，请参阅 GCC 手册中的 x86 选项章节。

编译该软件包：

make

要测试结果，请执行：

make check

安装该软件包：

make install

8.50.2. Libffi 安装内容

已安装库： libffi.so

简要描述

libffi 包含外部函数接口 API 函数

8.51. Python-3.13.2 Python 3 软件包包含 Python 开发环境。它适用于面向对象编程、编写脚本、大型程序原型设计以及开发完整应用程序。Python 是一种解释型计算机语言。

预计编译时间：2.1 SBU

所需磁盘空间：501 MB

8.51.1. Python 3 的安装

准备编译 Python：

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

配置选项的含义：

--with-system-expat

该选项启用与系统版 Expat 库的链接。

--enable-optimizations

该选项启用耗时但全面的优化步骤。解释器会被构建两次：首次构建的测试结果将用于优化最终版本。

编译该软件包：

make

已知某些测试可能会无限期挂起。因此为测试结果，需运行测试套件但为每个测试用例设置 2 分钟时限：

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

对于运行较慢的系统，可能需要增加时限，1 个 SBU（使用单核 CPU 构建 Binutils 第一遍时测得）应已足够。某些测试存在不稳定性，测试套件会自动重试失败的测试。若某测试初次失败但重试后通过，则应视为通过。已知 test_ssl 测试在 chroot 环境中会失败。

安装该软件包：

make install

本书多处使用 pip3 命令以 root 身份为所有用户安装 Python 3 程序和模块。这与 Python 开发者的推荐做法相冲突：他们建议将包安装到虚拟环境中，或安装到普通用户的主目录（通过以该用户身份运行 pip3）。当 root 用户执行 pip3 时，会触发多行警告信息。

推荐此做法的主要原因是避免与系统包管理器（例如 dpkg）产生冲突。由于 LFS 没有系统级的包管理器，因此不存在这个问题。此外，pip3 每次运行时都会检查自身是否有新版本。由于 LFS 的 chroot 环境中尚未配置域名解析功能，pip3 无法检查自身更新，此时会产生警告信息。

当我们启动 LFS 系统并建立网络连接后，会出现另一种警告，提示用户从 PyPI 的预构建 wheel 更新 pip3（每当有新版本时）。但 LFS 将 pip3 视为 Python 3 的组成部分，因此不应单独更新。而且通过预构建 wheel 进行更新会导致偏离

从我们的目标出发：从源代码构建一个 Linux 系统。因此，关于 pip3 新版本的警告同样可以忽略。若您愿意，可通过运行以下命令创建配置文件来屏蔽所有此类警告：

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

重要提示

在 LFS 和 BLFS 中，我们通常使用 pip3 命令构建并安装 Python 模块。请确保两本书中所有 pip3 install 命令都以 root 用户身份运行（除非是针对 Python 虚拟环境）。以非 root 用户身份运行 pip3 install 看似能成功，但会导致其他用户无法访问已安装的模块。

pip3 install 命令不会自动重新安装已安装的模块。当使用 pip3 install 命令升级模块时（例如从 meson-0.61.3 升级到 meson-0.62.0），请在命令行中添加 --upgrade 选项。若确实需要降级模块或出于某些原因重新安装相同版本，请在命令行中添加 --force-reinstall --no-deps 选项。

如需安装预格式化的文档：

```
install -v -dm755 /usr/share/doc/python-3.13.2/html
```

```
tar --strip-components=1 \
--no-same-owner \
--no-same-permissions \
-C /usr/share/doc/python-3.13.2/html \
-xvf ../python-3.13.2-docs-html.tar.bz2
```

文档安装命令的含义：

--no-same-owner 和 --no-same-permissions

确保安装的文件具有正确的所有权和权限。若不使用这些选项，tar 会使用上游创建者的原始值来安装包文件。

8.51.2. Python 3 安装内容：

已安装程序：2to3、idle3、pip3、pydoc3、python3 和 python3-config

已安装库文件：libpython3.13.so 和 libpython3.so

已安装的目录：	安装目录：/usr/include/python3.13、/usr/lib/python3 和 /usr/share/doc/python-3.13.2
---------	--

简要描述

2to3	是一个 Python 程序，用于读取 Python 2.x 源代码并应用一系列修复将其转换为有效的 Python 3.x 代码
idle3	是一个包装脚本，用于打开支持 Python 的 GUI 编辑器。要运行此脚本，必须在安装 Python 前先安装 Tk，以便构建 Tkinter Python 模块。
pip3	Python 的包安装工具。您可以使用 pip 从 Python 包索引和其他索引安装包。

pydoc3 是 Python 的文档工具

python3 是 Python 语言的解释器，Python 是一种解释型、交互式、面向对象的编程语言

8.52. Flit-Core-3.11.0 Flit-core 是 Flit（一个简易 Python 模块打包工具）中负责构建分发包的核心组件。

预计构建时间：少于 0.1 SBU

所需磁盘空间：1.0 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 配置选项及命令的含义：

wheel

此命令将为此软件包构建 wheel 归档文件。

-w dist

指示 pip 将创建的 wheel 包放入 dist 目录。

--no-cache-dir

阻止 pip 将创建的 wheel 包复制到/root/.cache/pip 目录中。

安装

该命令用于安装软件包。

--no-build-isolation、--no-deps 和 --no-index

这些选项可防止从在线软件包仓库（PyPI）获取文件。若软件包已按正确顺序安装，pip 首先就无需获取任何文件；这些选项能在用户操作失误时提供额外保障。

--find-links dist

指示 pip 在 dist 目录中搜索 wheel 归档文件。

8.52.2. Flit-Core 安装目录内容：

/usr/lib/python3.13/site-packages/flit_core 和 /usr/lib/python3.13/site-packages/flit_core-3.11.0.dist-info

8.53. Wheel-0.45.1 Wheel 是一个 Python 库，它是 Python wheel 打包标准的参考实现。

预计构建时间：少于 0.1 SBU

所需磁盘空间：1.6 MB

8.53.1. Wheel 的安装

使用以下命令编译 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
```

8.53.2. Wheel 安装内容 已安装程序：

wheel

已安装的目录：

/usr/lib/python3.13/site-packages/wheel 和 /usr/lib/python3.13/site-packages/wheel-0.45.1.dist-info

简要描述

wheel 是一个用于解包、打包或转换 wheel 归档文件的实用工具

8.54. Setuptools-75.8.1 Setuptools 是一款用于下载、构建、安装、升级和卸载 Python 软件包的工具。

预计构建时间：少于 0.1 SBU

所需磁盘空间：26 MB

8.54.1. Setuptools 的安装

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
```

8.54.2. Setuptools 安装目录内容：

/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-75.8.1.dist-info

8.55. Ninja-1.12.1 Ninja 是一个专注于速度的小型构建系统。

预计构建时间: 0.2 SBU

所需磁盘空间: 37 MB

8.55.1. Ninja 的安装

运行时, `ninja` 通常会尽可能多地并行使用进程。默认情况下, 这个数量等于系统核心数加二。这可能导致 CPU 过热或系统内存耗尽。当从命令行调用 `ninja` 时, 传递 `-jN` 参数可以限制并行进程数。某些软件包会嵌入 `ninja` 的执行, 但不会向其传递 `-j` 参数。

使用以下可选步骤, 用户可以通过环境变量 `NINJAJOBS` 来限制并行进程数。例如设置:

```
export NINJAJOBS=4
```

这将把 `ninja` 的并行进程数限制为 4 个。

如果需要让 `ninja` 识别环境变量 `NINJAJOBS`, 可以运行以下流编辑器命令:

```
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
```

使用以下命令构建 Ninja:

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

构建选项的含义:

`--bootstrap`

该参数强制 `Ninja` 为当前系统重新构建自身。

`--verbose`

该参数使 `configure.py` 显示构建 `Ninja` 的进度。

该软件包测试无法在 chroot 环境中运行, 它们需要 `cmake`。不过该软件包的基本功能已通过自行重建 (使用`--bootstrap` 选项) 得到验证。

安装该软件包:

```
安装 -v 755 ninja /usr/bin/
安装 -vDm644 misc/bash-completion /usr/share/bash-completion/completions/ninja
安装 -vDm644 misc/zsh-completion /usr/share/zsh/site-functions/_ninja
```

8.55.2. Ninja 安装内容

已安装程序: `ninja`

简要描述

`ninja` 是 `Ninja` 构建系统

8.56. Meson-1.7.0 Meson 是一款开源构建系统，旨在实现极速构建并尽可能提升用户友好性。

预计编译时间：少于 0.1 SBU

所需磁盘空间：44 MB

8.56.1. Meson 的安装

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出 LFS 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
```

安装参数的含义：

`-w dist`

将生成的 wheel 包放入 dist 目录

`--find-links dist`

从 dist 目录安装 wheel 包

8.56.2. Meson 安装内容 已安装程序：

`meson`

安装目录：/usr/lib/python3.13/site-packages/meson-1.7.0.dist-info 和 /usr/lib/python3.13/site-packages/mesonbuild

简要描述

`meson` 一个高效能的构建系统

8.57. Kmod-34 Kmod 软件包包含用于加载内核模块的库和实用工具

预计构建时间：少于 0.1 SBU 所需磁盘空间：11 MB

8.57.1. 安装 Kmod 为编译准备 Kmod：

`mkdir -p build`
进入目录 构建

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

配置选项的含义：

`-D manpages=false`

该选项禁用需要外部程序支持的手册页生成功能。

编译该软件包：

忍者

该软件包的测试套件需要原始内核头文件（而非先前安装的“净化版”内核头文件），这已超出 LFS 的范围。

现在安装该软件包：

```
ninja install
```

8.57.2. Kmod 安装内容

已安装程序：

`depmod` (链接到 `kmod`)、`insmod` (链接到 `kmod`)、`kmod`、`lsmod` (链接到 `kmod`)、`modinfo` (链接到 `kmod`)、`modprobe` (链接到 `kmod`) 以及 `rmmmod` (链接到 `kmod`)

已安装的库：

`libkmod` 该库被其他程序用于加载和卸载内核模块

`kmod` 加载和卸载内核模块

`lsmod` 列出当前已加载的模块

`modinfo` 检查与内核模块关联的目标文件并显示其能获取的所有信息

`modprobe` 使用由 `depmod` 创建的依赖文件来自动加载相关模块

`rmmmod` 从运行中的内核卸载模块

8.58. Coreutils-9.6 Coreutils 软件包包含每个操作系统所需的基本实用程序

预计构建时间: 1.2 SBU

所需磁盘空间: 182 MB

8.58.1. Coreutils 的安装

POSIX 标准要求 Coreutils 中的程序即使在多字节语言环境下也能正确识别字符边界。以下补丁修复了该违规问题及其他国际化相关的缺陷。

```
patch -Np1 -i ../coreutils-9.6-i18n-1.patch
```

注意

该补丁中已发现多处缺陷。向 Coreutils 维护者报告新缺陷时, 请先确认这些缺陷在不应用此补丁的情况下是否仍可复现。

现在准备编译 Coreutils:

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

命令及配置选项的含义:

autoreconf -fv

国际化补丁已修改了构建系统, 因此必须重新生成配置文件。通常我们会使用 **-i** 选项来更新标准辅助文件, 但由于 `configure.ac` 指定了旧版 `gettext`, 此选项对该软件包无效。

automake -af

由于缺少 **-i** 选项, `autoreconf` 未能更新 `automake` 辅助文件。此命令将更新这些文件以防止构建失败。

FORCE_UNSAFE_CONFIGURE=1

该环境变量允许以 root 用户身份构建软件包。

--enable-no-install-program=kill,uptime

此开关的作用是防止 Coreutils 安装那些将由其他软件包安装的程序。

编译该软件包:

```
make
```

若不运行测试套件, 可直接跳转至“安装软件包”步骤。

现在测试套件已准备就绪可以运行。首先, 运行需要以 root 用户身份执行的测试:

```
make NON_ROOT_USERNAME=tester check-root
```

我们将以 `tester` 用户身份运行剩余的测试。某些测试要求用户属于多个组。为避免跳过这些测试, 需添加一个临时组并将 `tester` 用户加入其中:

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

修复部分权限，使非 root 用户能够编译和运行测试：

```
chown -R tester .
```

现在运行测试（使用/dev/null 作为标准输入，如果在图形终端或 SSH/GNU Screen 会话中构建 LFS，可能会有两个测试失败，因为标准输入连接到了宿主发行版的 PTY，而 LFS chroot 环境无法访问此类 PTY 的设备节点）：

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

删除临时用户组：

```
groupdel dummy
```

安装该软件包：

```
make install
```

将程序移动到文件系统层次结构标准(FHS)规定的位置：

```
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
```

8.58.2. Coreutils 安装内容

已安装程序：

[, 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, shalsum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty,

已安装库：libstdbuf.so (位于/usr/libexec/coreutils)

安装目录： /usr/libexec/coreutils

简要描述

[是一个实际存在的命令，/usr/bin/[；它是 test 命令的同义词
base32	根据 base32 规范 (RFC 4648) 对数据进行编码和解码
base64	根据 Base64 规范 (RFC 4648) 对数据进行编码和解码
b2sum	打印或校验 BLAKE2 (512 位) 校验和
basename	从文件名中去除路径和指定后缀
basenc	使用多种算法对数据进行编码或解码
cat	将文件内容连接并输出到标准输出
chcon	更改文件和目录的安全上下文
chgrp	更改文件和目录的所属组
chmod	修改文件权限 将每个文件的权限更改为指定模式；该模式可以是待修改权限的符号表示，也可以是代表新权限的八进制数

chown	更改文件和目录的用户及/或组所有权
chroot	以指定目录作为根目录(/)运行命令
cksum	打印每个指定文件的循环冗余校验(CRC)值和字节计数
comm	比较两个已排序文件，以三列形式输出特有行和共有行
cp	复制文件
csplit	将给定文件分割为多个新文件，根据指定模式或行号进行分隔，并输出每个新文件的字节计数
cut	按给定字段或位置选择部分内容，打印行的指定区段
date	以指定格式显示当前日期和时间，或设置系统日期和时间
dd	使用指定块大小和计数复制文件，并可选择对其进行格式转换
df	报告所有已挂载文件系统（或仅包含选定文件的文件系统）的可用（及已用）磁盘空间容量
dir	列出每个给定目录的内容（与 ls 命令相同）或以八进制数字表示的新权限
dircolors	输出用于设置 LS_COLOR 环境变量的命令，以更改 ls 使用的配色方案
dirname	提取给定名称中的目录部分
du	报告当前目录、每个给定目录（包括所有子目录）或每个给定文件所占用的磁盘空间量
echo	显示给定的字符串
环境变量	在修改后的环境中运行命令
展开	将制表符转换为空格
expr	计算表达式
factor	输出指定整数的质因数
false	不执行任何操作且总是返回失败状态码
fmt	重新格式化指定文件中的段落
折叠	对指定文件中的行进行换行处理
groups	报告用户的组成员身份
head	打印每个给定文件的前十行（或指定行数）
hostid	报告主机的数字标识符（十六进制格式）
id	报告当前用户或指定用户的有效用户 ID、组 ID 及所属组信息
安装	复制文件时设置其权限模式，并在可能的情况下设置其所有者和所属组
join	将两个独立文件中具有相同连接字段的行合并
link	创建指向文件的硬链接（使用指定名称）
ln	在文件之间创建硬链接或软（符号）链接
logname	报告当前用户的登录名
ls	列出每个给定目录的内容
md5sum	报告或校验消息摘要 5 (MD5) 校验和

创建目录 以给定名称创建目录

创建命名管道 以给定名称创建先进先出(FIFO)队列，即 UNIX 术语中的“命名管道”

创建具有指定名称的设备节点；设备节点可以是字符特殊文件、块特殊文件或 FIFO（命名管道）。mktemp 以安全方式创建临时文件，常用于脚本中。mv 用于移动或重命名文件和目录。nice 以修改后的调度优先级运行程序。nl 对给定文件的行进行编号。nohup 运行不受挂断影响的命令，并将其输出重定向到日志文件。nproc 打印进程可用的处理单元数量。numfmt 将数字转换为人类可读字符串或反向转换。

od 以八进制及其他格式转储文件内容

粘贴 将给定文件按行横向合并，用制表符分隔对应行

路径检查 检查文件名是否有效或可移植

pinky 一款轻量级 finger 客户端；可显示指定用户的相关信息

pr 对文件进行分页和分栏处理以便打印

printenv 打印环境变量

printf 按照给定格式打印参数，功能类似于 C 语言中的 printf 函数

ptx 从给定文件内容生成排列索引，每个关键词都保留其上下文

pwd 报告当前工作目录的名称

readlink 报告给定符号链接的值

realpath 打印解析后的路径

rm 删除文件或目录

rmdir 删除空目录

runcon 以指定的安全上下文运行命令

seq 在给定范围内按指定增量打印数字序列

sha1sum 打印或校验 160 位安全散列算法 1(SHA1)校验和

sha224sum 打印或校验 224 位安全散列算法校验和

sha256sum 打印或校验 256 位安全哈希算法校验和

sha384sum 打印或校验 384 位安全哈希算法校验和

sha512sum 打印或校验 512 位安全哈希算法校验和

shred 安全擦除反覆用复杂模式覆盖指定文件，使数据难以恢复

shuf 随机打乱文本行顺序

sleep 暂停指定时长

排序 对给定文件中的行进行排序

分割 将给定文件按大小或行数分割成若干部分

stat	显示文件或文件系统状态
stdbuf	以修改后的标准流缓冲操作运行命令
stty	设置或报告终端线路设置
sum	打印每个给定文件的校验和及块计数
同步	刷新文件系统缓冲区；强制将更改的块写入磁盘并更新超级块
tac	以逆序连接给定的文件
tail	打印每个给定文件的最后十行（或指定行数）
tee	从标准输入读取数据，同时写入标准输出和指定文件
test	比较数值并检查文件类型
timeout	在时间限制内运行命令
touch	更改文件时间戳，将指定文件的访问和修改时间设置为当前时间；不存在的文件会被创建为零字节空文件
tr	对标准输入中的字符进行转换、压缩和删除操作
true	空操作命令，总是以表示成功的状态码退出；可通过文件大小或行数进行截断操作 truncate 将文件收缩或扩展到指定大小
tsort	拓扑排序命令行拓扑排序；根据给定文件中的偏序关系输出完全有序的列表
tty	报告连接到标准输入的终端文件名
uname	报告系统信息
unexpand	将空格转换为制表符
uniq	仅保留连续相同行中的一行
unlink	删除指定文件
user	显示当前登录用户的用户名
虚拟目录列表	功能等同于 ls -l
wc	统计每个给定文件的行数、单词数和字节数，当提供多个文件时还会显示总计
who	报告当前登录系统的用户
whoami	报告与当前有效用户 ID 关联的用户名
yes	持续输出字母 y 或指定字符串，直到进程被终止
libstdbuf	stdbuf 命令调用的底层库

8.59. Check-0.15.2 Check 是一个针对 C 语言的单元测试框架。

预计构建时间: 0.1 SBU (包含测试约需 2.1 SBU)

所需磁盘空间: 11 MB

8.59.1. Check 的安装准备 为编译配置 Check 环境:

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

构建该软件包:

```
make
```

编译现已完成。要运行 Check 测试套件, 请执行以下命令:

```
make check
```

安装该软件包:

```
make docdir=/usr/share/doc/check-0.15.2 install
```

8.59.2. Check 安装内容 已安装程序:

checkmk

libcheck.so

简要描述

checkmk 用于生成 Check 单元测试框架的 C 语言单元测试的 Awk 脚本

libcheck.so 包含允许从测试程序调用 Check 的函数

8.60. Diffutils-3.11

Diffutils 软件包包含用于显示文件或目录差异的程序。

预计构建时间: 0.4 SBU 所需磁盘空间: 50 MB

8.60.1. Diffutils 的安装 准备编译 Diffutils:

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

编译该软件包:

```
make
```

要测试结果, 请执行:

```
make check
```

安装该软件包:

```
make install
```

8.60.2. Diffutils 的内容 已安装的程序:

cmp、diff、diff3 和 sdiff

简要描述

cmp 逐字节比较两个文件并报告差异

diff 比较两个文件或目录并报告文件中的不同行

diff3 逐行比较三个文件

sdiff 交互式合并两个文件并输出结果

8.61. Gawk-5.3.1 Gawk 软件包包含用于操作文本文件的程序。

预计编译时间：0.2 SBU 所需磁盘空间：43 MB

8.61.1. 安装 Gawk 首先确保不会安装一些不必要的文件：

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

准备编译 Gawk：

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

编译该软件包：

```
make
```

要测试结果，请执行：

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

安装该软件包：

```
rm -f /usr/bin/gawk-5.3.1
install
```

该命令的含义：

```
rm -f /usr/bin/gawk-5.3.1
```

如果硬链接 gawk-5.3.1 已存在，构建系统不会重新创建它。删除该链接可确保更新第 6.9 节“Gawk-5.3.1”中先前安装的硬链接。

安装过程已创建了指向 gawk 的 awk 符号链接，现在同样为其创建手册页的符号链接：

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

如需安装文档，请执行：

```
install -vDm644 doc/{awkforai.txt,*.{eps,pdf,jpg}} -t /usr/share/doc/gawk-5.3.1
```

8.61.2. Gawk 安装内容 已安装程序：

awk (链接至 gawk)、gawk 以及 gawk-5.3.1

已安装库文件：filefuncs.so、fnmatch.so、fork.so、inplace.so、intdiv.so、ordchr.so、

readdir.so、readfile.so、revoutput.so、revtwoWay.so、rwarray.so 和 time.so
/usr/lib/gawk/lib/gawk/libexec/awk、/usr/share/awk 以及 /usr/share/doc/gawk-5.3.1

简要描述

awk gawk 的链接

gawk 一个用于处理文本文件的程序；这是 GNU 实现的 awk

gawk-5.3.1 指向 gawk 的硬链接

8.62. Findutils-4.10.0 Findutils 软件包包含用于查找文件的程序。这些程序能够遍历目录树中的所有文件进行搜索，并能创建、维护和查询数据库（通常比递归查找更快，但若数据库未及时更新则不可靠）。Findutils 还提供 xargs 程序，可对搜索选中的每个文件执行指定命令。

预计编译时间：0.7 SBU

所需磁盘空间：63 MB

8.62.1. Findutils 的安装

准备编译 Findutils：

```
./configure --prefix=/usr --localstatedir=/var/lib/locate
```

配置选项的含义：

--localstatedir

该选项将定位数据库移动到/var/lib/locate 目录，这是符合 FHS 标准的位置。

编译该软件包：

```
make
```

要测试结果，请执行：

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

安装该软件包：

```
make install
```

8.62.2. Findutils 安装内容 已安装程序：

查找、定位、更新数据库和 xargs 命令

安装目录： /var/lib/locate 目录

简要描述

find 在指定目录树中搜索符合条件文件

locate 在数据库中搜索并报告包含给定字符串或匹配给定模式的文件名

updatedb 更新定位数据库；它会扫描整个文件系统（包括当前挂载的其他文件系统）

mount 挂载（除非被告知不要挂载），并将找到的每个文件名存入数据库

xargs 可用于对文件列表执行指定命令

8.63. Groff-1.23.0

Groff 软件包包含用于处理和格式化文本及图像的程序。

预计构建时间: 0.2 SBU

所需磁盘空间: 108 MB

8.63.1. Groff 的安装

Groff 期望环境变量 PAGE 包含默认纸张尺寸。美国用户应设置为 PAGE=letter，其他国家/地区用户可能更适合 PAGE=A4。虽然默认纸张尺寸在编译时配置，但后续可通过向/etc/papersize 文件写入 "A4" 或 "letter" 来覆盖该设置。

准备编译 Groff:

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

构建该软件包:

```
make
```

要测试结果，请执行:

```
make check
```

安装该软件包:

```
make install
```

8.63.2. 已安装的 Groff 程序内容:

addftinfo、afmtodit、chem、eqn、eqn2graph、gdiffmk、glilypond、gperl、gpinyin、grap2graph、grn、grofdi、groffer、grog、grolbp、grolj4、gropdf、grops、grotty、hpftodit、indxbib、lkbib、lookbib、mmroff、neqn、nroff、pdfmom、pdfroff、pfbtaps、pic、pic2graph、post-grohtml、preconv、pre-grohtml、refer、roff2dvi、roff2html、roff2pdf、roff2ps、roff2text、/usr/lib/groff 和 /usr/share/doc/groff-1.23.0、/usr/share/groff

已安装的目录:

简要描述

addftinfo 读取 troff 字体文件，并添加 groff 系统所需的额外字体度量信息

afmtodit 创建用于 groff 和 grops 的字体文件

chem 用于生成化学结构图的 Groff 预处理器

eqn 将嵌入在 troff 输入文件中的方程描述编译为 troff 可理解的命令

eqn2graph 将 troff EQN (方程) 转换为裁剪后的图像

gdiffmk 标记 groff/nroff/troff 文件之间的差异

glilypond 将使用 lilypond 语言编写的乐谱转换为 groff 语言

gperl groff 的预处理器，允许在 groff 文件中插入 perl 代码

gpinyin groff 的预处理器，允许在 groff 文件中插入拼音（用罗马字母拼写的汉语普通话）

grap2graph	将 grap 程序文件转换为裁剪后的位图图像 (grap 是一种古老的 Unix 编程语言, 用于创建图表)
grn	用于 gremlin 文件的 groff 预处理器
grodvi	生成 TeX dvi 格式输出文件的 groff 驱动程序
groff	groff 文档格式化系统的前端接口; 通常它会运行 troff 程序以及适用于所选设备的后处理器
groffer	在 X 窗口和 tty 终端上显示 groff 文件及手册页
grog 命令	读取文件并推测打印时需要哪些 groff 选项 (-e、-man、-me、-mm、-ms、-p、-s 及-t 选项), 最终输出包含这些选项的 groff 完整命令
grolbp 打印机驱动	适用于佳能 CAPSL 打印机 (LBP-4 和 LBP-8 系列激光打印机) 的 groff 驱动程序
grolj4	生成适用于 HP LaserJet 4 打印机的 PCL5 格式输出的 groff 驱动程序
gropdf	将 GNU troff 的输出转换为 PDF 格式
grops	将 GNU troff 的输出转换为 PostScript 格式
grotty	将 GNU troff 的输出转换为适用于类打字机设备的格式
hpftodit	根据 HP 标记的字体度量文件创建字体文件, 供 groff -Tlj4 使用
indxbib	为文献数据库创建倒排索引, 指定文件供 refer、lookbib 和 lkbib 使用
lkbib	在书目数据库中搜索包含指定关键词的文献, 并报告所有找到的文献
lookbib	在标准错误输出上显示提示 (除非标准输入不是终端设备), 从标准输入读取包含一组关键词的行, 在指定文件的书目数据库中搜索包含这些关键词的文献, 将找到的文献打印到标准输出, 并重复此过程直至输入结束
mmroff	groff 的简单预处理器
neqn	为美国信息交换标准码 (ASCII) 输出格式化方程式
nroff	一个使用 groff 模拟 nroff 命令的脚本
pdfmom	groff 的封装工具, 便于将使用 mom 宏格式化的文件转换为 PDF 文档
pdfroff	使用 groff 创建 pdf 文档
pfbtops	将.pfb 格式的 PostScript 字体转换为 ASCII 格式
图片	将嵌入在 troff 或 TeX 输入文件中的图片描述编译成 TeX 或 troff 能够理解的命令
pic2graph	将 PIC 图表转换为裁剪后的图像
post-grohtml	将 GNU troff 的输出转换为 HTML 格式
preconv	将输入文件的编码转换为 GNU troff 可识别的格式
pre-grohtml	将 GNU troff 的输出转换为 HTML 格式
refer	将文件内容复制到标准输出, 但处理方式如下: 位于.[和.]之间的行会被解释为引用内容, 位于.R1 和.R2 之间的行则被解释为引用处理指令

roff2dvi	R2 被解释为处理引用方式的命令 将 roff 文件转换为 DVI 格式
roff2html	将 roff 文件转换为 HTML 格式
roff2pdf	将 roff 文件转换为 PDF 格式
roff2ps	将 roff 文件转换为 ps 格式文件
roff2text	将 roff 文件转换为文本文件
roff2x	将 roff 文件转换为其他格式
soelim	读取文件并将形如.so file 的行替换为指定文件的内容
tbl	将嵌入在 troff 输入文件中的表格描述编译成 troff 能够理解的命令
tfmtodit	创建用于 groff -Tdvi 的字体文件
troff	与 Unix troff 高度兼容；通常应使用 groff 命令调用，该命令还会以正确的顺序和适当的选项运行预处理程序和后处理程序

8.64. GRUB-2.12 GRUB 软件包包含 GRand 统一引导加载程序。

预计构建时间：0.3 SBU 所需磁盘空间：
间：166 MB

8.64.1. GRUB 安装注意事项

若您的系统支持 UEFI 且希望以 UEFI 方式启动 LFS 系统，请按照 BLFS 手册页面的说明安装支持 UEFI 的 GRUB（及其依赖项）。您可以选择跳过本软件包，或同时安装本软件包与 BLFS 提供的 UEFI 版 GRUB 软件包（二者不会冲突，BLFS 页面提供了两种情况的安装指导）。

警告

清除可能影响构建的环境变量：

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

切勿尝试使用自定义编译标志来“调优”此软件包。该软件包是引导加载程序，源代码中的底层操作可能会被激进的优化破坏。

为发布压缩包补充缺失的文件：

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

准备编译 GRUB：

```
./configure --prefix=/usr \
--sysconfdir=/etc \
--disable-efiemu \
--disable-werror
```

新增配置选项的含义：

--disable-werror

这样即使 Flex 新版本产生警告信息，也能完成构建。

--disable-efiemu

该选项通过禁用某项功能并移除 LFS 不需要的测试程序，以最小化构建内容。

编译该软件包：

```
make
```

不建议运行此软件包的测试套件。大多数测试依赖于有限的 LFS 环境中不可用的软件包。若仍要运行测试，请执行 make check 命令。

安装该软件包，并将 Bash 自动补全支持文件移至 Bash 自动补全维护者推荐的位置：

```
make install
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

使用 GRUB 使您的 LFS 系统可启动将在第 10.4 节"使用 GRUB 设置启动流程"中讨论。

8.64.2. GRUB 安装内容

已安装程序：

grub-bios-setup、grub-editenv、grub-file、grub-fstest、grub-glue-efi、grub-install、grubkbdcomp、grub-macbless、grub-menulst2cfg、grub-mkconfig、grub-mkimage、grubmklayout、grub-mknodir、grub-mkpasswd-pbkdf2、grub-mkrelpath、grub-mkrescue、grub-mkstandalone、grub-ofpathname、grub-probe、grub-reboot、grub-render-label、grub-script-check、grub-set-/usr/lib/grub、/etc/grub.d、/usr/share/grub 以及/boot/grub（首次运行 grub-install 时创建）

已安装的目录：

简要描述

grub-bios-setup 是 grub-install 的辅助程序，用于 grub-editenv

用于编辑环境块的工具

grub-file 检查给定文件是否属于指定类型

grub-fstest 用于调试文件系统驱动的工具

grub-glue-efi 将 32 位和 64 位二进制文件合并为单一文件（适用于苹果设备）

grub-install 将 GRUB 安装到您的驱动器

grub-kbdcomp 将 xkb 键盘布局转换为 GRUB 可识别的格式的脚本

grub-macbless Mac 风格的 bless 命令适用于 HFS 或 HFS+ 文件系统（bless 是苹果电脑特有的功能，它能使设备可启动）

grub-menulst2cfg 将 GRUB Legacy 的 menu.lst 转换为适用于 GRUB 2 的 grub.cfg 文件

grub-mkconfig 生成 grub.cfg 配置文件

grub-mkimage 生成可启动的 GRUB 镜像文件

grub-mklayout 生成 GRUB 键盘布局文件

grub-mknodir 准备 GRUB 网络启动目录 grub-mkpasswd-pbkdf2 为启动菜单生成加密的 PBKDF2 密码

grub-mkrelpath 生成相对于系统根目录的路径名

grub-mkrescue 制作适用于软盘、CDROM/DVD 或 USB 驱动器的 GRUB 可启动镜像

grub-mkstandalone 生成独立镜像

grub-ofpathname 辅助程序，用于打印 GRUB 设备的路径

grub-probe 探测给定路径或设备的设备信息

grub-reboot 仅设置下一次启动时 GRUB 的默认启动项

grub-render-label 为苹果 Mac 电脑渲染 Apple .disk_label 文件

grub-script-check 检查 GRUB 配置脚本的语法错误

grub-set-default 设置 GRUB 的默认启动项

grub-sparc64-setup 是 grub-setup 的辅助程序

将 syslinux 配置文件转换为 grub.cfg 格式

8.65. Gzip-1.13 Gzip 软件包包含用于文件压缩与解压缩的程序。

预计编译时间: 0.3 SBU 所需磁盘空间: 21 MB

8.65.1. Gzip 的安装 准备编译 Gzip:

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

编译该软件包:

```
make
```

要测试结果, 请执行:

```
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 硬链接)、zcat、zcmp、zdiff、zegrep、zfgrep、zforce、zgrep、zless、zmore 和 znew

简要描述

gunzip 解压缩 gzip 格式的文件

gzexe 创建自解压可执行文件

gzip 使用 Lempel-Ziv (LZ77) 编码压缩指定文件

uncompress 解压压缩文件

zcat 将指定的 gzip 压缩文件解压到标准输出

zcmp 对 gzip 压缩文件执行 cmp 比较

zdiff 对 gzip 压缩文件执行 diff 差异比对

zegrep 对 gzip 压缩文件执行 egrep 扩展正则搜索

zfgrep 对 gzip 压缩文件执行 fgrep 固定字符串搜索

zforce 强制为所有已 gzip 压缩的文件添加.gz 扩展名, 防止 gzip 重复压缩; 这在文件名因文件传输被截断时特别有用

zgrep 对 gzip 压缩文件执行 grep 搜索

zless 对 gzip 压缩文件执行 less 查看

zmore 对 gzip 压缩文件执行 more 命令查看

znew 将文件从 compress 格式 (.Z) 重新压缩为 gzip 格式 (.gz)

8.66. IPRoute2-6.13.0 IPRoute2 软件包包含基于 IPv4 的基础和高级网络工具程序。

预计编译时间: 0.1 SBU 所需磁盘空间: 17 MB

8.66.1. 安装 IPRoute2 由于该软件包包含的 arpd 程序依赖未在 LFS 中安装的 Berkeley DB，因此不会构建该程序。但系统仍会安装 arpd 的目录和手册页。通过运行以下命令可避免这种情况：

```
sed -i /ARPD/d Makefile rm -fv  
man/man8/arpd.8
```

编译该软件包：

```
make NETNS_RUN_DIR=/run/netns
```

该软件包没有可用的测试套件。

安装该软件包：

```
make SBINDIR=/usr/sbin install
```

如需安装文档，请执行：

```
install -vDm644 COPYING README* -t /usr/share/doc/iproute2-6.13.0
```

8.66.2. IPRoute2 安装内容

已安装程序： 已安装的目录：	bridge、ctstat (链接至 linstat)、genl、ifstat、ip、linstat、nstat、routel、rtacct、 rtmon、rtpr、rtstat (链接至 linstat)、ss 以及 tc /etc/iproute2、/usr/lib/tc 和 /usr/share/doc/iproute2-6.13.0
---------------------------------	---

简要描述

bridge 配置网络桥接
ctstat 连接状态工具

genl 通用网络链接实用工具前端 ifstat 显示接口统计信息，包括各接口收发数据包数量

ip	主可执行程序。具备多种功能，包括： ip link <设备> 允许用户查看设备状态并进行修改 ip addr 允许用户查看地址及其属性，添加新地址或删除旧地址 ip neighbor 允许用户查看邻居绑定及其属性，添加新邻居条目或删除旧条目 ip rule 允许用户查看路由策略并进行修改 ip route 允许用户查看路由表并修改路由规则 ip tunnel 允许用户查看 IP 隧道及其属性，并进行修改 ip maddr 允许用户查看组播地址及其属性，并进行修改 ip mroute 允许用户设置、修改或删除组播路由
-----------	--

ip monitor 允许用户持续监控设备状态、地址和路由

或删除组播路由。Instat 提供 Linux 网络统计信息，它是旧版 rtstat 程序的通用化且功能更完善的替代品
nstat 显示网络统计信息。routel 是 ip route 的组件，用于列出路由表。rtacct 显示 /proc/net/rt_acct 的内容。rtmon
是路由监控工具

rtpm 将 ip -o 的输出转换为可读格式

路由状态工具 rtstat

ss 类似于 netstat 命令；显示活动连接

tc 服务质量(QoS)和服务等级(CoS)实现的流量控制

 tc qdisc 允许用户设置队列规则

 tc class 允许用户基于队列规则调度设置分类

 tc filter 允许用户设置 QoS/CoS 数据包过滤

 tc monitor 可用于查看内核中对流量控制的修改

8.67. Kbd-2.7.1 Kbd 软件包包含键盘映射表文件、控制台字体和键盘实用工具。

预计编译时间：0.1 SBU 所需磁盘空间：34 MB

8.67.1. Kbd 的安装 在 Kbd 软件包提供的键盘映射中，退格键和删除键的行为表现不一致。以下

补丁修复了 i386 键盘映射的这个问题：

```
patch -Np1 -i ../kbd-2.7.1-backspace-1.patch
```

打补丁后，退格键将生成代码为 127 的字符，而删除键会生成一个常见的转义序列。

移除冗余的 resizecons 程序（该程序需要已废弃的 svgalib 来提供视频模式文件——在常规使用中 setfont 会正确设置控制台尺寸）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
```

该选项阻止构建 vlock 工具，因为它需要 PAM 库，而该库在 chroot 环境中不可用。

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

注意

对于某些语言（如白俄罗斯语），Kbd 软件包并未提供实用的键盘映射方案，因为默认的“by”键位映射采用 ISO-8859-5 编码，而通常使用的是 CP1251 键位映射。这些语言的用户需要单独下载可用的键盘映射文件。

如需安装文档，请执行：

```
cp -R -v docs/doc -T /usr/share/doc/kbd-2.7.1
```

8.67.2. 已安装的 Kbd 程序内容：

chvt、deallocvt、dumpkeys、fgconsole、getkeycodes、kbdinfo、kbd_mode、
 kbdrate、loadkeys、loadunimap、mapscrn、openvt、psfaddtable (链接至
 psfxtable)、psfgettable (链接至 psfxtable)、psfstriptable (链接至 psfxtable)、
 psfxtable、setfont、setkeycodes、setleds、setmetamode、setvtrgb、
 /usr/share/consolefonts、/usr/share/consoletrans、/usr/share/keymaps、/usr/sh
 are/doc/kbd-2.7.1 以及/usr/share/unimaps

已安装的目录：

/usr/share/consolefonts、/usr/share/consoletrans、/usr/share/keymaps、/usr/sh
 are/doc/kbd-2.7.1 以及/usr/share/unimaps

简要描述

chvt	切换前台虚拟终端
deallocvt	释放未使用的虚拟终端
dumpkeys	转储键盘转换表
fgconsole	显示当前活动虚拟终端的编号
getkeycodes	显示内核扫描码到键码的映射表
kbdinfo	获取控制台状态信息
kbd_mode	报告或设置键盘模式
kbdrate	设置键盘重复率和延迟速率
loadkeys	加载键盘转译表
loadunimap	加载内核 Unicode 到字体映射表
mapscrn	一个已废弃的程序，曾用于将用户定义的输出字符映射表加载到控制台驱动程序中；该功能现由 setfont 实现
openvt	在新虚拟终端(VT)上启动程序
psfaddtable	向控制台字体添加 Unicode 字符表
psfgettable	从控制台字体提取内嵌的 Unicode 字符表
psfstriptable	移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	更改控制台上的增强图形适配器(EGA)和视频图形阵列(VGA)字体
setkeycodes	加载内核扫描码到键码的映射表条目；当键盘上有特殊按键时特别有用
setleds	设置键盘标志位和发光二极管(LED)状态
setmetamode	定义键盘元键处理方式
setvtrgb	设置所有虚拟终端中的控制台颜色映射表
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 码
unicode_start	将键盘和控制台切换至 UNICODE 模式[除非您的键盘映射文件采用 ISO-8859-1 编码，否则请勿使用此程序。对于其他编码格式，该工具会产生错误结果。]
unicode_stop	将键盘和控制台从 UNICODE 模式恢复

8.68. Libpipeline-1.5.8 Libpipeline 软件包包含一个用于以灵活便捷方式操作子进程管道的函数库。

预计编译时间: 0.1 SBU 所需磁盘空间: 11 MB

8.68.1. Libpipeline 的安装 为编译准备 Libpipeline:

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

编译该软件包:

```
make
```

要测试结果, 请执行:

```
make check
```

安装该软件包:

```
make install
```

8.68.2. Libpipeline 的内容 已安装的库文件:

libpipeline.so

简要描述

libpipeline

该库用于在子进程之间安全地构建管道

8.69. Make-4.4.1 Make 软件包包含一个用于控制从源代码生成可执行文件及其他非源代码文件的程序。

预计编译时间：0.7 SBU 所需磁盘空间：13 MB

8.69.1. Make 的安装 准备编译 Make:

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

编译该软件包：

```
make
```

要测试结果，请执行：

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

安装该软件包：

```
make install
```

8.69.2. 已安装程序的内容：

```
make
```

简要描述

make 自动判断软件包的哪些部分需要（重新）编译，并执行相应的命令

8.70. Patch-2.7.6 Patch 软件包包含一个通过应用"补丁"文件（通常由 diff 程序生成）来修改或创建文件的程序。

预计编译时间：0.2 SBU

所需磁盘空间：12 MB

8.70.1. Patch 的安装准备工作

为编译 Patch 做准备：

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

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

安装该软件包：

```
make install
```

8.70.2. Patch 包含的内容

已安装程序： 补丁

简要描述

补丁程序根据补丁文件修改文件（补丁文件通常是由 diff 程序生成的差异列表。通过将这些差异应用到原始文件上，补丁程序就能生成修改后的版本。）

8.71. Tar-1.35 Tar 软件包提供了创建 tar 归档文件的功能，并能执行多种其他类型的归档操作。Tar 可用于从现有归档中提取文件、追加存储新文件，或更新/列出已存储的文件内容。

预计编译时间：0.6 SBU

所需磁盘空间：43 MB

8.71.1. Tar 的安装

准备编译 Tar：

```
FORCE_UNSAFE_CONFIGURE=1 \
./configure --prefix=/usr
```

配置选项的含义：

```
FORCE_UNSAFE_CONFIGURE=1
```

这会强制以 root 身份运行 mknod 测试。通常认为以 root 用户运行此测试存在风险，但由于当前仅在部分构建的系统上执行，因此覆盖该设置是安全的。

编译该软件包：

```
make
```

要测试结果，请执行：

```
make check
```

已知其中一项关于能力机制（capabilities）的测试——二进制存储/恢复——会因 LFS 缺少 selinux 而失败。但如果宿主机的内核不支持构建 LFS 所用文件系统的扩展属性或安全标签，该测试将被跳过。

安装该软件包：

```
make install make -C doc install-html docdir=/usr/share/doc/tar-1.35
```

8.71.2. 已安装的 Tar 程序内容：

tar

安装目录： /usr/share/doc/tar-1.35

简要描述

tar 用于创建、提取文件以及列出归档文件（也称为 tarball）的内容

8.72. Texinfo-7.2

Texinfo 软件包包含用于读取、编写和转换 info 页面的程序。

预计编译时间: 0.3 SBU

所需磁盘空间: 160 MB

8.72.1. Texinfo 的安装

准备编译 Texinfo:

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

编译该软件包:

```
make
```

要测试结果, 请执行:

```
make check
```

安装该软件包:

```
make install
```

可选地, 安装属于 TeX 安装的组件:

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

make 参数的含义:

TEXMF=/usr/share/texmf

TEXMF 这个 makefile 变量用于指定 TeX 目录树的根路径, 例如后续安装 TeX 软件包时会用到该路径。

Info 文档系统使用纯文本文件来存储其菜单条目列表。该文件位于 /usr/share/info/dir 文件。遗憾的是, 由于各软件包 Makefile 中偶尔出现的问题, 该文件有时会与系统中安装的 info 页面不同步。如果需要重新创建 /usr/share/info/dir 文件, 可以通过以下可选命令完成该任务:

```
pushd /usr/share/info rm -v
dir for f in *
do install-info $f dir 2>/dev/null done popd
```

8.72.2. Texinfo 安装内容 已安装程序:

信息文档工具: install-info、makeinfo (链接至 texi2any)、pdftexi2dvi、pod2texi、texi2any、texi2dvi、texi2pdf 以及 texindex

已安装库文件: MiscXS.so、ParseTeXi.so 和 XSParagraph.so (均位于 /usr/lib/texinfo 目录)

已安装的目录: 安装路径: /usr/share/texinfo 与 /usr/lib/texinfo

简要描述

info 命令

用于阅读信息页, 其功能类似手册页, 但通常比单纯解释所有可用命令行选项更为深入 [例如, 可以对比 man bison 和 info bison 的内容差异]。

install-info	用于安装 info 页面；它会更新 info 索引文件中的条目
makeinfo	将给定的 Texinfo 源文档转换为 info 页面、纯文本或 HTML 格式
pdftexi2dvi	用于将给定的 Texinfo 文档格式化为便携式文档格式(PDF)文件
pod2texi	将 Pod 转换为 Texinfo 格式
texi2any	将 Texinfo 源文档转换为多种其他格式
texi2dvi	用于将给定的 Texinfo 文档格式化为可打印的设备无关文件
texi2pdf	用于将给定的 Texinfo 文档格式化为便携式文档格式(PDF)文件
texindex	用于对 Texinfo 索引文件进行排序

8.73. Vim-9.1.1166 Vim 软件包包含一个功能强大的文本编辑器。

预计编译时间：3.4 SBU

所需磁盘空间：251 MB

Vim 的替代方案

如果您偏好其他编辑器——例如 Emacs、Joe 或 Nano——请参考 <https://www.linuxfromscratch.org/blfs/view/12.3/postlfs/editors.html> 获取推荐的安装指南。

8.73.1. Vim 的安装步骤

首先，将 vimrc 配置文件的默认位置更改为 /etc 目录：

```
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
```

确保测试用户 tester 对源代码树具有写入权限，并排除一个包含需要 curl 或 wget 的测试文件：

```
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：

```
以测试用户身份执行命令：su tester -c "TERM=xterm-256color LANG=en_US.UTF-8 make -j1 test" \
&> vim-test.log
```

测试套件会向屏幕输出大量二进制数据。这可能干扰当前终端的设置（特别是当我们覆盖 TERM 变量以满足测试套件的某些假设时）。如上所示将输出重定向到日志文件可避免该问题。测试成功后，日志文件末尾会出现 ALL DONE 字样。

安装该软件包：

```
make install
```

许多用户会条件反射地输入 vi 而非 vim。为使用户在习惯性输入 vi 时仍能执行 vim 程序，请为二进制文件和各语言手册页创建符号链接：

```
执行链接操作：ln -sv vim /usr/bin/vi
并为所有语言版本的手册页创建链接：for L in
/usr/share/man/*vim* done
```

默认情况下，Vim 的文档安装在 /usr/share/vim 目录下。以下符号链接允许通过 /usr/share/doc/vim-9.1.1166 访问文档，使其与其他软件包的文档位置保持一致：

```
ln -sv .. /vim/vim91/doc /usr/share/doc/vim-9.1.1166
```

如果要在 LFS 系统上安装 X Window 系统，可能需要在安装 X 后重新编译 Vim。Vim 附带了一个需要 X 和额外库支持的图形界面版本编辑器。有关此过程的更多信息，请参阅 Vim 文档及 BLFS 手册中的 Vim 安装页面：<https://www.linuxfromscratch.org/blfs/view/12.3/postlfs/vim.html>。

8.73.2. 配置 Vim

默认情况下，vim 以与 vi 不兼容的模式运行。这对于过去使用其他编辑器的用户可能是新特性。下方包含的“nocompatible”设置既是为了强调正在使用新行为，也提醒那些想切换至“compatible”模式的用户：这必须是配置文件中的首选设置。因为该设置会改变其他参数，所有覆盖配置都必须在此设置之后进行。现在创建一个默认的 vim 配置文件

通过运行以下命令：

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

" 在自定义设置前确保加载默认配置，而非之后 source $VIMRUNTIME/defaults.vim let
skip_defaults_vim=1
```

```
set nocompatible
set backspace=2
set mouse=
if (&term == "xterm") || (&term == "putty")
syntax on
set background=dark
endif
```

```
" 结束 /etc/vimrc 配置
EOF
```

set nocompatible 设置使 vim 以更实用的方式（默认模式）运行，而非 vi 兼容模式。若删除“no”则保留旧式 vi 行为。set backspace=2 设置允许退格键跨行删除、取消自动缩进及回退插入起始点。syntax on 参数启用 vim 的语法高亮功能。set mouse= 设置确保在 chroot 环境或远程连接时能正确使用鼠标粘贴文本。最后的 if 语句配合 set background=dark 设置可修正 vim 对某些终端模拟器背景色的误判，从而在这些程序的黑色背景上呈现更优的高亮配色方案。

要获取其他可用选项的文档，可运行以下命令：

```
vim -c ':options'
```

注意

默认情况下，vim 仅安装英语的拼写检查文件。如需安装您首选语言的拼写检查文件，请将对应语言及字符编码的.spl 文件（可选.sug 文件）从 runtime/spell 目录复制到/usr/share/vim/vim91/spell/中。

要使用这些拼写检查文件，需要在/etc/vimrc 中进行一些配置，例如：

```
set spelllang=en,ru
set spell
```

更多信息请参阅 runtime/spell/README.txt。

8.73.3. 已安装的 Vim 程序内容：

ex (链接到 vim)、rview (链接到 vim)、rvim (链接到 vim)、vi (链接到 vim)、
view (链接到 vim)、vim、vimdiff (链接到 vim)、vimtutor 以及 xxd
安装目录：
/usr/share/vim

简要描述

ex	以 ex 模式启动 vim
rview	是 view 的受限版本；无法执行 shell 命令且无法暂停 view
rvim	是 vim 的受限版本；无法启动 shell 命令且无法挂起 vim
vi	链接到 vim
view	以只读模式启动 vim
vim	是编辑器 vimdiff 使用 vim 编辑文件的两个或三个版本并显示差异 vimtutor 教授 vim 的基本按键和命令
xxd	为给定文件创建十六进制转储；它还可以执行反向操作，因此可用于二进制补丁

8.74. MarkupSafe-3.0.2 **MarkupSafe** 是一个实现 XML/HTML/XHTML 标记安全字符串的 Python 模块。

预计构建时间：少于 0.1 SBU

所需磁盘空间：500 KB

8.74.1. MarkupSafe 的安装

使用以下命令编译 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 安装目录内容：

/usr/lib/python3.13/site-packages/MarkupSafe-3.0.2.dist-info

8.75. Jinja2-3.1.5 Jinja2 是一个实现了简洁 Python 风格模板语言的 Python 模块。

预计编译时间：少于 0.1 SBU

所需磁盘空间：2.5 MB

8.75.1. Jinja2 的安装

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 安装目录内容：

/usr/lib/python3.13/site-packages/Jinja2-3.1.5.dist-info

8.76. 来自 Systemd-257.3 的 Udev

Udev 软件包包含用于动态创建设备节点的程序。

预计构建时间: 0.3 SBU

所需磁盘空间: 161 MB

8.76.1. Udev 的安装

Udev 是 systemd-257.3 软件包的组成部分。请使用 systemd-257.3.tar.xz 文件作为源码包。

从默认 udev 规则中移除两个不必要的用户组 render 和 sgx:

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

移除一条需要完整 Systemd 安装的 udev 规则:

```
sed -i '/systemd-sysctl/s/^#/' rules.d/99-systemd.rules.in
```

调整独立安装 udev 时网络配置文件的硬编码路径:

```
sed -e '/NETWORK_DIRS/s/systemd/udev/' \
-i src/libsystemd/sd-network/network-util.h
```

准备编译 Udev:

```
mkdir -p build  
进入目录 构建
```

```
meson setup .. \
--prefix=/usr \ --buildtype=release \
-D mode=release \ -D dev-kvm-mode=0660 \
\ -D link-udev-shared=false \ -D
logind=false \ -D vconsole=false
```

meson 选项的含义:

--buildtype=release

该选项会覆盖默认的构建类型 ("debug")，后者会生成未经优化的二进制文件。

-D mode=release

禁用上游认为仍处于实验阶段的部分功能。

-D dev-kvm-mode=0660

默认的 udev 规则允许所有用户访问/dev/kvm。编者认为这存在安全隐患。该选项将覆盖此默认设置。

-D link-udev-shared=false

该选项阻止 udev 链接到 systemd 内部共享库 libsystemd-shared。该共享库设计初衷是为多个 systemd 组件共用，对于仅安装 udev 的场景显得过于冗余。

-D logind=false -D vconsole=false

这些选项可防止生成属于其他 Systemd 组件的多个 udev 规则文件，这些组件我们不会安装。

获取已发布的 udev 助手程序列表，并将其保存到环境变量中（严格来说无需导出该变量，但这样能简化普通用户构建或使用包管理器的流程）：

```
export udev_helpers=$(grep "' name'" ./src/udev/meson.build | \
    awk '{print $3}' | tr -d ',' | grep -v 'udevadm')
```

仅构建 udev 所需的组件：

```
ninja udev$@minjatend+hwdbp -Eo '(src/(lib)?udev|rules.d|hwdb.d)/[^ ]*' \ $(realpath libudev.so
--relative-to .) \ $udev_helpers
```

安装该软件包：

```
安装 -vm755 -d {/usr/lib,/etc}/udev/{hwdb.d, rules.d, network}
安装 -vm755 -d /usr/{lib, share}/pkgconfig
安装 -vm755 udevadm /usr/bin/
安装 -vm755 systemd-hwdb /usr/bin/udev-hwdb
创建符号链接 -svfn ../bin/udevadm /usr/sbin/udevd
复制 -av libudev.so{,[0-9]} /usr/lib/
安装 -vm644 ../src/libudev/libudev.h /usr/include/
安装 -vm644 src/libudev/*.pc /usr/lib/pkgconfig/
安装 -vm644 src/udev/*.pc /usr/share/pkgconfig/
安装 -vm644 ../src/udev/udev.conf /etc/udev/
安装 -vm644 rules.d/* ../rules.d/README /usr/lib/udev/rules.d/
安装 -vm644 $(find ../rules.d/*.*.rules \
-not -name '*power-switch') /usr/lib/udev/rules.d/
安装 -vm644 hwdb.d/* ../hwdb.d/{*,hwdb, README} /usr/lib/udev/hwdb.d/
安装 -vm755 $udev_helpers /usr/lib/udev
安装 -vm644 ../network/99-default.link /usr/lib/udev/network
```

在 LFS 环境中安装一些有用的自定义规则和支持文件：

```
tar -xvf ../../udev-lfs-20230818.tar.xz make -f udev-lfs-
20230818/Makefile.lfs install
```

安装手册页：

```
tar -xf ../../systemd-man-pages-257.3.tar.xz \
--no-same-owner --strip-components=1 \ -C /usr/share/man --wildcards '*/udev*' '*/libudev*'
'*/systemd.link.5' \ '*/systemd-
{hwdb, udevd. service}.8

sed 's|systemd/network|udev/network|' \
/usr/share/man/man5/systemd.link.5 \ > /usr/share/man/man5/udev.link.5

sed 's/systemd\\(\\\\?-\)/udev\\1/' /usr/share/man/man8/systemd-hwdb.8 \
> /usr/share/man/man8/udev-hwdb.8

sed 's|lib.*udevd|sbin/udevd|' \
/usr/share/man/man8/systemd-udevd. service.8 \ > /usr/share/man/man8/udevd.8
```

删除 /usr/share/man/man*/systemd* 文件

最后取消设置 udev_helpers 变量：

```
unset udev_helpers
```

8.76.2. 配置 Udev 硬件设备信息保存在 /etc/udev/hwdb.d 和 /usr/lib/udev/hwdb.d 目录中。Udev

需要将这些信息编译成二进制数据库 /etc/udev/hwdb.bin。创建初始数据库：

```
udev-hwdb update
```

每次硬件信息更新后都需要运行此命令。

8.76.3. 已安装的 Udev 内容：

udevadm、udevd（指向 udevadm 的符号链接）以及 udev-hwdb

已安装的库文件：libudev.so 已安装的目录：

/etc/udev 和 /usr/lib/udev

简要描述

udevadm 通用 udev 管理工具：控制 udevd 守护进程，提供来自 Udev 数据库的信息，

监控 uevents 事件，等待 uevents 完成，测试 Udev 配置，并为指定设备触发 uevents

udevd 一个监听 netlink 套接字上 uevents 事件的守护进程，根据这些事件创建设备并运行配置的外部程序

udev-hwdb 用于更新或查询硬件数据库。

libudev 访问 udev 设备信息的库接口

/etc/udev 包含 Udev 配置文件、设备权限及设备命名规则

8.77. Man-DB-2.13.0 Man-DB 软件包包含用于查找和查看 man 手册页的程序。

预计编译时间：0.3 SBU 所需磁盘空间：44 MB

8.77.1. Man-DB 的安装 准备编译 Man-DB：

```
./configure --prefix=/usr/share/doc/man-db-2.13.0 \
            sysconfdir=/etc \
            --disable-setuid \
            --enable-cache-owner=bin \
            --with-browser=/usr/bin/lynx \
            --with-vgrind=/usr/bin/vgrind \
            --with-grap=/usr/bin/grap \
            --with-systemdtmpfilesdir= \
            --with-systemdsystemunitdir=
```

配置选项的含义：

--disable-setuid

该选项禁用将 man 程序设置为用户 man 的 setuid 权限。

--enable-cache-owner=bin

该选项将系统级缓存文件的所有权更改为用户 bin。

--with-...

这三个参数用于设置一些默认程序。lynx 是基于文本的网页浏览器（安装说明请参阅 BLFS），vgrind 将程序源代码转换为 Groff 输入格式，grap 则对在 Groff 文档中排版图表很有帮助。通常查看手册页时不需要 vgrind 和 grap 程序。它们不属于 LFS 或 BLFS 的组成部分，但如果需要，可以在完成 LFS 后自行安装。

--with-systemd...

这些参数可避免安装不必要的 systemd 目录和文件。

编译该软件包：

make

要测试结果，请执行：

make check

安装该软件包：

make install

8.77.2. LFS 中的非英语手册页 下表展示了 Man-DB 假定安装在 /usr/share/man/ 目录下的手册页所使用的字符集编码。除此之外，Man-DB 还能正确判断该目录下手册页是否采用 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		
韩语 (ko)	EUC-KR		
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	塞尔维亚语拉丁字母 (sr@latin)	ISO-8859-2 字符编码
罗马尼亚语 (ro)	ISO-8859-2		
保加利亚语 (bg)	ISO-8859-1	字符编码 塞尔维亚语 (sr) ISO-8859-5 字符编码 挪威尼诺斯	
希腊语 (el)	ISO-8859-7	字符编码 土耳其语 (tr)	ISO-8859-9 字符编码
冰岛语 (is)	ISO-8859-1		
斯洛伐克语 (sk)	ISO-8859-2		
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	瑞典语
斯洛文尼亚语 (sl)	ISO-8859-1	简体中文 (zh_CN) GBK 白俄罗斯语 (be) CP1251 简体中文, 新加坡	
挪威书面语 (nb)			
		GBK	
	(zh_SG)		
保加利亚语 (bg)	CP1251 繁体中文 (香港)		BIG5 香港扩展字符集
	(zh_HK)		
捷克语 (cs)	ISO-8859-2 繁体中文 (zh_TW)		BIG5

注意

不支持列表中未列出的语言的手册页。

8.77.3. Man-DB 安装的程序内容：

accessdb、apropos (链接至 whatis)、catman、lexgrog、man、man-recode、mandb、manpath 和 whatis

已安装库文件: libman.so 和 libmandb.so (均位于 /usr/lib/man-db 目录)

已安装的目录: /usr/lib/man-db、/usr/libexec/man-db 以及 /usr/share/doc/man-db-2.13.0

简要描述

accessdb 以人类可读格式转储 whatis 数据库内容

`apropos` 在 whatis 数据库中搜索并显示包含指定字符串的系统命令的简短描述

`catman` 创建或更新预格式化的手册页

`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` 将手册页转换为另一种编码

`mandb` 创建或更新 whatis 数据库

`manpath` 显示\$MANPATH 的内容，若\$MANPATH 未设置则基于 man.conf 配置和用户环境显示合适的搜索路径

`whatis` 检索 whatis 数据库并显示包含给定关键词（作为独立单词）的系统命令简短描述

`libman` 包含对 man 命令的运行时支持

`libmandb` 库 包含对 man 命令的运行时支持

8.78. Procps-ng-4.0.5 软件包包含用于监控进程的程序

预计编译时间: 0.1 SBU

所需磁盘空间: 28 MB

8.78.1. Procps-ng 的安装

准备编译 Procps-ng:

```
./configure --prefix=/usr --sysconfdir=/usr/share/doc/procps-ng-4.0.5 \
            --disable-static \
            --disable-kill \
            --enable-watch8bit
```

配置选项的含义:

--disable-kill

该选项禁用构建 kill 命令; 该命令将从 Util-linux 软件包中安装。

--enable-watch8bit

该选项为 watch 命令启用 ncursesw 支持, 使其能够处理 8 位字符。

编译该软件包:

make

要运行测试套件, 请执行:

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

已知若主机内核未启用 CONFIG_BSD_PROCESS_ACCT 配置, 名为 ps 的测试 (输出标志为 bsdtime,cputime,etime,etimes) 会失败。此外, 在 chroot 环境中某个 pgrep 测试可能失败。

安装该软件包:

make install

8.78.2. Procps-ng 安装内容:

已安装程序:

free、pgrep、pidof、pkill、pmap、ps、pwdx、slabtop、sysctl、tload、top、uptime、vmstat、w 及 watch

已安装库文件: libproc-2.so 已安装目录:

/usr/include/procps 和 /usr/share/doc/procps-ng-4.0.5

简要描述

free 报告系统中空闲和已用内存量 (包括物理内存和交换内存)

pgrep 根据进程名称及其他属性查找进程

pidof 报告指定程序的进程 ID(PID)

pkill 根据进程名称及其他属性向进程发送信号

pmap 报告指定进程的内存映射

ps	列出当前运行的进程
pwdx	报告进程的当前工作目录
slabtop	实时显示详细的内核 slab 缓存信息
sysctl	在运行时修改内核参数
tload	打印当前系统平均负载的图形
top	显示 CPU 占用最高的进程列表；实时持续查看处理器活动情况
系统运行时间	报告系统已运行时长、当前登录用户数以及系统负载平均值
虚拟内存统计	报告虚拟内存统计信息，提供有关进程、内存、分页、块输入/输出(IO)、陷阱和 CPU 活动的数据
w	显示当前登录的用户、登录位置及登录时间
watch	重复运行指定命令，显示其输出的首屏内容；允许用户观察输出随时间的变化
libproc-2	包含本软件包中大多数程序所使用的函数

8.79. Util-linux-2.40.4

Util-linux 软件包包含各种实用程序工具。其中包括用于处理文件系统、控制台、分区和消息的实用工具。

预计构建时间：0.5 SBU

所需磁盘空间：316 MB

8.79.1. Util-linux 的安装

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 \
--without-systemd \
--without-systemdsystemunitdir \
ADJTIME_PATH=/var/lib/hwclock/adjtime \
--docdir=/usr/share/doc/util-linux-2.40.4
```

--disable 和--without 选项用于禁用那些需要 LFS 未包含的依赖包，或与其他软件包安装程序存在冲突的组件构建，以避免警告信息。

编译该软件包：

make

如需运行测试套件，可创建一个虚拟的/etc/fstab 文件来满足两项测试要求，并以非 root 用户身份执行：

警告

以 root 用户身份运行测试套件可能对系统造成损害。要运行测试，当前运行系统中必须提供内核的 CONFIG_SCSI_DEBUG 选项，且该选项必须编译为模块形式。若直接编译进内核将导致系统无法启动。要实现完整测试覆盖，还需安装其他 BLFS 软件包。如需测试，可启动完整的 LFS 系统后执行以下命令：

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

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

如果主机内核未启用 CONFIG_CRYPTO_USER_API_HASH 选项，或未配置任何提供 SHA256 实现的选项（例如 CONFIG_CRYPTO_SHA256 或 CONFIG_CRYPTO_SHA256_SSSE3（若 CPU 支持 Supplemental SSE3）已启用。此外，若内核未启用 CONFIG_NETLINK_DIAG 选项，lsfd: inotify 测试将失败。

另外两项测试——lsfd: SOURCE 列和 utmp: last——已知在 chroot 环境中会失败。

安装该软件包：

```
make install
```

8.79.2. Util-linux 安装的程序内容：

addpart、agetty、blkdiscard、blkid、blkzone、blockdev、cal、cfdisk、chcpu、chmem、choom、chrt、col、colcrt、colrm、column、ctrlaltdel、delpart、dmesg、eject、fallocate、fdisk、fincore、findfs、findmnt、flock、fsck、fsck.cramfs、fsck.minix、fsfreeze、fstrim、getopt、hardlink、hexdump、hwclock、i386（链接到 setarch）、ionice、ipcmk、ipcrm、ipcs、irqtop、isosize、kill、last、lastb（链接到 last）、ldattach、linux32（链接到 setarch）、linux64（链接到 setarch）、logger、look、losetup、lsblk、lscpu、lsipc、lsirq、lsfd、lslocks、lslogins、lsmem、lsns、mcookie、mesg、mkfs、mkfs.bfs、mkfs.cramfs、mkfs.minix、mkswap、more、mount、mountpoint、namei、nsenter、partx、pivot_root、prlimit、readprofile、rename、renice、resizepart、rev、rfkill、rtcwake、script、scriptlive、scriptreplay、setarch、setsid、setterm、sfdisk、sulogin、swaplabel、swapoff、swapon、

已安装的库文件：libblkid.so^{（块设备 ID）}、libfdisk.so^{（块设备分区表）}、libmount.so^{（挂载点）}、libsmartcols.so^{（智能列）} 和 libuuid.so^{（全局唯一标识符）}

已安装的目录：

/usr/include/blkid、/usr/include/libfdisk、/usr/include/libmount、/usr/include/libsmartcols、/usr/include/uuid、/usr/share/doc/util-linux-2.40.4 以及 /var/lib/hwclock

简要描述

addpart	向 Linux 内核通知新增分区
agetty	打开一个 tty 端口，提示输入登录名，然后调用登录程序
blkdiscard	丢弃设备上的扇区
blkid	一个用于定位和打印块设备属性的命令行工具
blkzone	用于管理分区存储块设备
blockdev	允许用户从命令行调用块设备 ioctl 控制命令
cal	显示简易日历
cfdisk	操作指定设备的分区表
chcpu	修改 CPU 状态
chmem	配置内存
choom	显示并调整 OOM-killer 评分值，用于确定当 Linux 系统内存耗尽时优先终止哪个进程
chrt	调整进程的实时调度属性
列	过滤掉反向换行符
colcrt	为缺少某些功能（如重叠打印和半行显示）的终端过滤 nroff 输出
colrm	移除指定列 过滤指定列
列	将给定文件格式化为多列显示
ctrlaltdel	设置 Ctrl+Alt+Del 组合键的功能为硬重置或软重置

删除分区	请求 Linux 内核移除一个分区
dmesg	转储内核启动消息
eject	弹出设备 弹出可移动媒体
fallocate	为文件预分配空间
fdisk	操作指定设备的分区表
fincore	统计文件内容在内存中的页面数
findfs	通过标签或通用唯一标识符(UUID)查找文件系统
findmnt	是一个用于操作挂载信息、fstab 和 mtab 文件的 libmount 库命令行接口
flock	获取文件锁并在持有锁的情况下执行命令
fsck	用于检查并可选地修复文件系统
fsck.cramfs	对指定设备上的 Cramfs 文件系统执行一致性检查
fsck.minix	对指定设备上的 Minix 文件系统执行一致性检查
fsfreeze	是围绕 FIFREEZE/FITHAW ioctl 内核驱动操作的极简封装工具
fstrim	回收已挂载文件系统中未使用的块
getopt	解析给定命令行中的选项
hardlink	通过创建硬链接来合并重复文件
hexdump	以十六进制、十进制、八进制或 ASCII 格式转储指定文件
hwclock	读取或设置系统硬件时钟（也称为实时时钟 RTC 或基本输入-输出系统 BIOS 时钟
i386	指向 setarch 的符号链接
ionice	获取或设置程序的 I/O 调度类别及优先级
ipcmk	创建各类进程间通信(IPC)资源
ipcrm	移除指定的进程间通信(IPC)资源
ipcs	提供 IPC 状态信息
irqtop	以 top(1) 风格视图显示内核中断计数器信息
isosize	报告 iso9660 文件系统的大小
kill	向进程发送信号
last	显示最近登录（及登出）的用户信息，通过搜索 /var/log/wtmp 文件回溯记录；同时显示系统启动、关机及运行级别变更情况
lastb	显示登录失败尝试，记录于 /var/log/btmp 文件中
ldattach	将线路规程附加到串行线路上
linux32	指向 setarch 的符号链接
linux64	指向 setarch 的符号链接
logger	将给定消息记录到系统日志中
查看	显示以给定字符串开头的行
循环设备设置	设置并控制循环设备

lsblk	以树状格式列出所有或选定块设备的信息
lscpu	打印 CPU 架构信息
lsfd	显示已打开文件的信息；替代 lsof 命令
lsipc	打印系统中当前使用的进程间通信(IPC)设施信息
lsirq	显示内核中断计数器信息
lslocks	列出本地系统锁信息
lslogins	列出用户、群组及系统账户的相关信息
lsmem	列出可用内存范围及其在线状态
lsns	列出命名空间
mcookie	为 xauth 生成魔法 cookie (128 位随机十六进制数)
mesg	控制其他用户能否向当前用户的终端发送消息
mkfs	在设备上构建文件系统 (通常是硬盘分区)
mkfs.bfs	创建圣克鲁兹操作系统(SCO)的 bfs 文件系统
mkfs.cramfs	创建 cramfs 文件系统
mkfs.minix	创建 Minix 文件系统
mkswap	将指定设备或文件初始化为交换分区
more	分页过滤器，每次显示一屏文本内容
mount	将指定设备上的文件系统挂载到文件系统树中的指定目录
挂载点	检查目录是否为挂载点
namei	显示给定路径中的符号链接
nsenter	在其他进程的命名空间中运行程序
partx	向内核报告磁盘分区的存在情况及编号信息
pivot_root	将指定文件系统设为当前进程的新根文件系统
prlimit	获取和设置进程的资源限制
readprofile	读取内核性能分析信息
rename	重命名指定文件，将给定字符串替换为另一个
renice	修改运行中进程的优先级
调整分区大小	请求 Linux 内核调整分区尺寸
反向	反转指定文件的行序
射频控制	无线设备启用/禁用工具
rtcwake	用于使系统进入休眠状态直至指定唤醒时间
script	生成终端会话的文本记录
scriptlive	根据时序信息重放会话文本记录
scriptreplay	使用计时信息回放终端会话记录
setarch	在新程序环境中变更报告的体系架构，并设置个性标志
setsid	在新会话中运行指定程序

setterm	设置终端属性
sfdisk	磁盘分区表操作工具
sulogin	允许 root 用户登录；通常在系统进入单用户模式时由 init 调用
swaplabel	修改交换分区的 UUID 和标签
关闭交换分区	禁用设备和文件的分页与交换功能
启用交换分区	激活设备和文件的分页与交换功能，并列出当前正在使用的设备和文件
切换根目录	将另一个文件系统挂载为根目录树
taskset	获取或设置进程的 CPU 亲和性
uclampset	操作系统或进程的利用率钳位属性操作工具
ul	将下划线转换为当前终端所用下划线转义序列的过滤器
umount	将文件系统从系统的文件树中卸载
uname26	指向 setarch 的符号链接
unshare	在部分命名空间与父进程隔离的环境中运行程序
utmpdump	以用户友好的格式显示指定登录文件的内容
uuidd	一个由 UUID 库使用的守护进程，用于以安全且确保唯一性的方式生成基于时间的 UUID 独特方式
uuidgen	创建新的 UUID。每个新 UUID 都是一个随机数，极大概率（可能生成 2^{122} 种 UUID）确保其 在本地系统和其他系统上、过去和未来创建的所有 UUID 中都是唯一的
uuidparse	解析唯一标识符的工具
wall	在所有当前登录用户的终端上显示文件内容（默认显示标准输入内容）
wdctl	显示硬件看门狗状态
whereis	报告指定命令的二进制文件、源代码和手册页文件的位置
wipefs	擦除设备上的文件系统签名
x86_64	指向 setarch 的符号链接
zramctl	用于设置和控制 zram（压缩内存磁盘）设备的程序
libblkid	包含设备识别和令牌提取功能的程序库
libfdisk	包含操作分区表的例程
libmount	包含块设备挂载与卸载的例程
libsmartcols	包含用于以表格形式辅助屏幕输出的例程
libuuid	包含为可能在本地系统之外访问的对象生成唯一标识符的例程

8.80. E2fsprogs-1.47.2

E2fsprogs 软件包包含用于处理 ext2 文件系统的工具集，同时支持 ext3 和 ext4 日志文件系统。

预计编译时间：机械硬盘约 2.4 SBU，固态硬盘约 0.5 SBU

所需磁盘空间： 占用空间： 99 MB

8.80.1. E2fsprogs 安装

E2fsprogs 文档建议在源码树的子目录中进行编译：

```
mkdir -v build
进入目录 构建
```

准备编译 E2fsprogs：

```
./configure --prefix=/usr \
--sysconfdir=/etc \
--enable-elf-shlibs \
--disable-libblkid \
--disable-libuuid \
--disable-uuid \
--disable-fsck
```

配置选项的含义：

--enable-elf-shlibs

这将创建该软件包中某些程序所需的共享库。

--disable-*

这些选项会阻止构建和安装 libuuid、libblkid 库、uuid 守护进程以及 fsck 封装程序；因为 util-linux 会安装更新的版本。

编译该软件包：

```
make
```

要运行测试，请执行：

```
make check
```

已知名为 m_assume_storage_prezeroed 的测试会失败。另一个名为 m_rootdir_acl 的测试在 LFS 系统使用的文件系统不是 ext4 时也会失败。

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，但未更新系统范围的 dir 文件。请解压该文件，然后使用以下命令更新系统 dir 文件：

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir
/usr/share/info/libext2fs.info
```

如需要，可通过执行以下命令创建并安装额外文档：

```
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
```

8.80.2. 配置 E2fsprogs /etc/mke2fs.conf

该文件包含 mke2fs 各类命令行选项的默认值。您可编辑此文件使默认值符合需求。例如某些工具（不在 LFS 或 BLFS 中）无法识别启用 metadata_csum_seed 特性的 ext4 文件系统。若需使用此类工具，可通过以下命令从默认 ext4 特性列表中移除该特性：

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

详情请参阅 mke2fs.conf(5) 手册页。

8.80.3. E2fsprogs 安装的程序内容：

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、

已安装的库文件：libcom_err.so、libe2p.so、libext2fs.so 和 libss.so

已安装的目录： /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	用于检测设备（通常是磁盘分区）中的坏块
chattr	修改 ext{234} 文件系统上文件的属性
compile_et	错误表编译器；将包含错误代码名称和消息的表格转换为适合与 com_err 库一起使用的 C 源文件
调试文件系统	一款文件系统调试工具，可用于检查和修改 ext{234} 文件系统的状态
显示指定设备上文件系统的超级块和块组信息	
e2freefrag	报告空闲空间碎片信息
e2fsck	用于检查并可选择修复 ext{234} 文件系统
e2image	用于将 ext{234} 文件系统的关键数据保存至文件
e2label	显示或修改指定设备上 ext{234} 文件系统的卷标
e2mmpstatus	检查 ext4 文件系统的多重挂载保护(MMP)状态
e2scrub	检查已挂载的 ext{234} 文件系统内容
e2scrub_all	检查所有已挂载的 ext{234} 文件系统是否存在错误
e2undo	回放设备上 ext{234} 文件系统的撤销日志。[可用于撤销 E2fsprogs 程序执行失败的操作。]
e4crypt	Ext4 文件系统加密工具
e4defrag	ext4 文件系统的在线碎片整理工具

filefrag	报告特定文件的碎片化程度
fsck.ext2	默认检查 ext2 文件系统，是 e2fsck 的硬链接
fsck.ext3	默认检查 ext3 文件系统，是 e2fsck 的硬链接
fsck.ext4	默认检查 ext4 文件系统，是 e2fsck 的硬链接
日志保存	将命令的输出保存到日志文件中
lsattr	列出第二扩展文件系统上文件的属性
mk_cmds	将命令名称与帮助信息表转换为适用于 libss 子系统库的 C 源文件
mke2fs	在指定设备上创建 ext{234}文件系统
mkfs.ext2	默认创建 ext2 文件系统，是 mke2fs 的硬链接
mkfs.ext3	默认创建 ext3 文件系统，是 mke2fs 的硬链接
mkfs.ext4	默认创建 ext4 文件系统，是 mke2fs 的硬链接 mklost+found 在 ext{234}文件系统上创建 lost+found 目录；它会预先为这个目录分配磁盘块 减轻 e2fsck 工作负担的目录
resize2fs	可用于扩展或收缩 ext{234}文件系统
tune2fs	调整 ext{234}文件系统的可调参数
libcom_err	通用错误显示例程
libe2p	被 dumpe2fs、chattr 和 lsattr 使用
libext2fs	包含允许用户级程序操作 ext{234}文件系统的例程
libss	debugfs 使用

8.81. Sysklogd-2.7.0 Sysklogd 软件包包含用于记录系统消息的程序，例如当异常事件发生时内核发出的消息。

预计构建时间：少于 0.1 SBU 所需磁盘空间：

4.1 MB

8.81.1. Sysklogd 的安装 准备编译该软件包：

```
./configure --prefix=/usr \
--sysconfdir=/etc \ --runstatedir=/run \ --without-
logger \ --disable-static \ --
docdir=/usr/share/doc/sysklogd-2.7.0
```

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 install

8.81.2. 配置 Sysklogd 通过运行以下命令创建新的/etc/syslog.conf 文件：

```
cat > /etc/syslog.conf << "EOF"
# 开始 /etc/syslog.conf 配置

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# 不要开放任何互联网端口。secure_mode 2

# /etc/syslog.conf 结束
EOF
```

8.81.3. Sysklogd 安装内容：已安装程序：

syslogd

简要描述

syslogd 记录系统程序提供的日志消息 [每条日志消息至少包含日期

时间戳和主机名，通常还会包含程序名称，但这取决于日志守护进程被配置的信任级别。]

8.82. SysVinit-3.14 SysVinit 软件包包含用于控制系统启动、运行和关闭的程序。

预计构建时间：少于 0.1 SBU

所需磁盘空间：2.9 MB

8.82.1. SysVinit 的安装

首先应用一个补丁，该补丁移除了其他软件包安装的若干程序，优化了提示信息，并修复了编译器

警告：

```
patch -Np1 -i ../sysvinit-3.14-consolidated-1.patch
```

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 install
```

8.82.2. SysVinit 安装内容

bootlogd、fstab-decode、halt、init、killall5、poweroff（指向 halt 的链接）、reboot（指向 halt 的链接）、runlevel、shutdown 以及 telinit（指向 init 的链接）

简要描述

bootlogd 将启动信息记录到日志文件

fstab-decode 使用 fstab 编码参数运行命令

halt 通常调用带-h 选项的 shutdown 命令，但当系统已处于运行级别 0 时，会通知内核停止系统；它会在/var/log/wtmp 文件中记录系统即将关闭

init 内核完成硬件初始化后启动的第一个进程；它接管启动过程并启动其配置文件中指定的所有进程

killall5 向除自身会话内进程外的所有进程发送信号；不会终止其父 shell

poweroff 通知内核停止系统运行并关闭计算机（参见 halt 命令）

重启 通知内核重新启动系统（参见 halt 命令）

运行级别 报告前一个和当前的运行级别，记录在/run/utmp 中的最后一条运行级别记录

关机 以安全方式关闭系统，通知所有进程并告知所有已登录用户

切换运行级别 通知 init 要切换到的运行级别

8.83. 关于调试符号

默认情况下，大多数程序和库在编译时会包含调试符号（通过 `gcc` 的`-g` 选项）。这意味着当调试带有调试信息的程序或库时，调试器不仅能提供内存地址，还能显示例程和变量名称。

这些调试符号的加入会显著增大程序或库的体积。以下是两个示例，展示这些符号占用的空间大小：

- 含调试符号的 `bash` 二进制文件：1200 KB
- 不含调试符号的 `bash` 二进制文件：480 KB（体积减小 60%）
- 带有调试符号的 `Glibc` 和 `GCC` 文件（`/lib` 和 `/usr/lib`）：87 MB
- 不带调试符号的 `Glibc` 和 `GCC` 文件：16 MB（缩小 82%）

具体大小会因使用的编译器和 C 库而有所不同，但剥离调试符号后的程序通常比未剥离版本小 50% 到 80%。由于大多数用户永远不会在系统软件上使用调试器，移除这些符号可以回收大量磁盘空间。下一节将展示如何从程序和库中剥离所有调试符号。

8.84. 剥离调试符号（可选步骤）

如果目标用户不是程序员且不打算对系统软件进行调试，通过从二进制文件和库中移除调试符号及部分不必要的符号表条目，可将系统体积减少约 2GB。这对典型 Linux 用户不会造成实际影响。

大多数使用下述命令的用户不会遇到任何问题。但操作失误可能导致新系统无法使用，因此在执行 `strip` 命令前，建议先对当前状态的 LFS 系统进行备份。

使用`--strip-unneeded` 选项的 `strip` 命令会移除二进制文件或库中的所有调试符号，同时删除链接器（针对静态库）或动态链接器（针对动态链接二进制文件和共享库）不需要的所有符号表条目。

选定库中的调试符号将通过 Zlib 压缩后单独保存。这些调试信息是后续在 BLFS 中运行 `valgrind` 或 `gdb` 回归测试所必需的。

请注意，`strip` 命令会直接覆盖它正在处理的二进制文件或库文件。这可能导致正在使用该文件代码或数据的进程崩溃。如果运行 `strip` 的进程本身受到影响，正在被剥离的二进制文件或库可能会遭到破坏，进而导致系统完全无法使用。为避免此问题，我们先将部分库文件和二进制文件复制到 `/tmp` 目录下，在该目录下执行剥离操作，随后通过 `install` 命令重新安装它们。（8.2.1 节“升级问题”中的相关条目解释了此处使用 `install` 命令的原因。）

注意

ELF 加载器在 64 位系统中名为 `ld-linux-x86-64.so.2`，在 32 位系统中名为 `ld-linux.so.2`。下方构造的代码会根据当前系统架构选择正确的文件名（排除所有以 `g` 结尾的文件名，以防下方命令已被执行过）。

重要提示

如果存在任何软件包的版本与书中指定的版本不同（无论是遵循安全建议还是满足个人偏好），可能需要更新

`save_usrlib` 或 `online_usrlib` 中的库文件名。若未能及时更新，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.6.0.33
    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=zlib $LIB $LIB.debug cp $LIB /tmp/$LIB strip
--strip-unneeded /tmp/$LIB objcopy --add-gnu-debuglink=$LIB.debug /tmp/$LIB install -vm755
/tmp/$LIB /usr/lib rm /tmp/$LIB done
```

在线用户二进制文件="bash find strip" 在线用
户库="libbfd-2.44.so

```
libsframe.so.1.0.0
libhistory.so.8.2
libcursesw.so.6.5
libm.so.6
libreadline.so.8.2
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/$BIN strip --strip-
unneeded /tmp/$BIN install -vm755 /tmp/$BIN
/usr/bin rm /tmp/$BIN done
```

```
for LIB in $online_usrlib; do
将/usr/lib/$LIB 复制到/tmp/$LIB, 然后去
除/tmp/$LIB 中不必要的符号信息, 接着以 755 权
限将/tmp/$LIB 安装到/usr/lib, 最后删
除/tmp/$LIB
```

在/usr/lib 目录下查找所有.so*格式（但不包含*.dbg 后缀）的文件

在/usr/lib 目录下查找所有.a 格式的文件，以及

在/usr/bin、/usr/sbin、/usr/libexec 目录下查找所有文件；然后针对每个找到的文件，检查其是否存在于以下变量中：

```
*$(basename $i)* )
;;
* ) strip --strip-unneeded $i
;;
esac
```

完成

取消设置 BIN LIB save_usrlib online_usrbin online_usrlib

大量文件会被标记为错误，因为它们的文件格式无法识别。这些警告可以安全忽略，表明这些文件是脚本而非二进制文件。

8.85. 清理工作

最后，清理运行测试后残留的多余文件：

```
rm -rf /tmp/*
```

在/usr/lib 和/usr/libexec 目录下还存在若干扩展名为.la 的文件。这些是"libtool 归档"文件。在现代 Linux 系统中，libtool 的.la 文件仅对 libltdl 有用。而 LFS 中的库预计不会被 libltdl 加载，且已知某些.la 文件可能导致 BLFS 软件包构建失败。现在请删除这些文件：

查找 /usr/lib 和 /usr/libexec 目录下所有 .la 文件并删除

关于 libtool 归档文件的更多信息，请参阅 BLFS 手册的"关于 Libtool 归档(.la)文件"章节。

第 6 章和第 7 章中构建的编译器仍部分保留在系统中，现已不再需要。可通过以下命令移除：

查找 /usr 目录下所有名称匹配 \$(uname -m)-lfs-linux-gnu* 的文件/目录并递归删除

最后，删除在前一章开头创建的临时用户账户'tester'。

```
userdel -r tester
```

第 9 章 系统配置

9.1. 简介

启动 Linux 系统涉及多项任务。该过程需要挂载虚拟和真实文件系统、初始化设备、检查文件系统完整性、挂载并激活交换分区或交换文件、设置系统时钟、启动网络服务、运行系统所需的守护进程，以及完成用户指定的其他自定义任务。必须合理组织这一流程，确保各项任务按正确顺序执行，并尽可能快速地完成。

9.1.1. System V

System V 是经典的启动流程，自 1983 年起就应用于 Unix 及类 Unix 系统（如 Linux）。

其核心是一个名为 init 的小型程序，负责建立基本进程（如通过 getty 实现的登录功能）并运行脚本。该脚本通常命名为 rc，用于控制执行一系列附加脚本，这些脚本完成系统初始化所需的各项任务。

init 程序由/etc/inittab 文件控制，并按用户可选择的运行级别进行组织。在 LFS 中，这些运行级别的用途如下：

- 0 – 关机
- 1 – 单用户模式
- 2 – 用户自定义模式
- 3 – 完整多用户模式
- 4 – 用户自定义模式
- 5 – 带显示管理器的完整多用户模式
- 6 – 重启

通常默认的运行级别是 3 或 5。

优势

- 成熟且广为人知的系统。
- 易于定制。

缺点

- 启动速度可能较慢。一个中等速度的基础 LFS 系统需要 8-12 秒完成启动（计时区间从内核第一条消息显示到登录提示符出现）。网络连接通常在登录提示符出现后约 2 秒内建立完成。
- 启动任务采用串行处理。这与前一点相关，任何进程（例如文件系统检查）出现延迟都会拖慢整个启动流程。
- 不直接支持控制组（cgroups）和用户级公平份额调度等高级功能。
- 添加脚本需要手动进行静态排序决策。

9.2. LFS-Bootscripts-20240825 软件包 LFS-Bootscripts 包含一组用于在启动/关机时控制 LFS 系统运行的脚本。后续章节将描述自定义启动流程所需的配置文件与操作步骤。

预计编译时间：少于 0.1 SBU

所需磁盘空间：244 KB

9.2.1. LFS-Bootscripts 的安装 执行以下命令安装该软件包：

```
make install
```

9.2.2. LFS-Bootscripts 内容 已安装脚本：

已安装的目录：	checkfs、cleanfs、console、functions、halt、ifdown、ifup、localnet、modules、mountfs、mountvirtfs、network、rc、reboot、sendsignals、setclock、ipv4-static、swap、sysctl、sysklogd、template、udev 和 udev_retry /etc/rc.d、/etc/init.d（符号链接）、/etc/sysconfig、/lib/services、/lib/lsb（符号链接）
---------	---

简要描述

checkfs 在挂载文件系统前检查其完整性（日志型和基于网络的文件系统除外）

清理文件系统 删除不应在重启后保留的文件，例如位于/run/和/var/lock/目录下的文件；它会重新创建/run/utmp 文件，并移除可能存在的/etc/nologin、/fastboot 和/forcefsck 文件

控制台 为指定的键盘布局加载正确的键位映射表；同时设置屏幕字体

功能函数 包含多个引导脚本共用的通用功能，如错误和状态检查

halt 停止系统运行

ifdown 关闭网络设备

ifup 初始化网络设备

localnet 设置系统主机名和本地回环设备

modules 加载/etc/sysconfig/modules 中列出的内核模块，并使用该文件中提供的参数

挂载文件系统 挂载所有文件系统，但标记为 noauto 或基于网络的挂载点除外

挂载虚拟文件系统 挂载虚拟内核文件系统，如 proc

网络配置 设置网络接口（如网卡）并配置默认网关（如适用）

rc 主运行级别控制脚本；负责按符号链接名称确定的顺序，逐一运行所有其他启动脚本

reboot 重启系统

sendsignals 确保系统重启或关机前终止所有进程

setclock 若硬件时钟未设置为 UTC，则将系统时钟重置为本地时间

ipv4-static 提供为网络接口分配静态互联网协议(IP)地址所需的功能

交换空间	启用或禁用交换文件和交换分区
sysctl	如果存在/etc/sysctl.conf 文件，则将该文件中的系统配置值加载到运行中的内核
系统日志守护进程	启动和停止系统及内核日志守护进程
模板	用于为其他守护进程创建自定义启动脚本的模板
udev	准备/dev 目录并启动 udev 守护进程
udev_retry	重试失败的 udev uevents，并将生成的规则文件从/run/udev 复制到/etc/udev/rules。 d 如果需要

9.3. 设备与模块处理概述 在第 8 章中，我们构建 udev 时安装了 udev 守护进程。在深入探讨 udev 工作原理之前，有必要简要回顾先前处理设备的方法。

传统上，Linux 系统通常采用静态创建设备的方法，即在 /dev 目录下创建大量设备节点（有时甚至多达数千个），无论对应的硬件设备是否真实存在。这通常通过 MAKEDEV 脚本实现，该脚本包含大量对 mknod 程序的调用，其中包含世界上可能存在的所有设备对应的主设备号和次设备号。

采用 udev 方法后，设备节点仅针对内核检测到的设备进行创建。这些设备节点在每次系统启动时生成，并存储在 devtmpfs 文件系统（完全驻留在系统内存中的虚拟文件系统）中。设备节点占用的空间极小，因此消耗的内存可以忽略不计。

9.3.1. 历史背景

2000 年 2 月，一种名为 devfs 的新型文件系统被合并到 2.3.46 内核中，并在 2.4 系列稳定内核期间提供使用。尽管它存在于内核源代码中，但这种动态创建设备的方法始终未能获得核心内核开发者的广泛支持。

devfs 采用的方法存在的主要问题在于其处理设备检测、创建和命名的方式。其中最关键的问题或许是设备节点命名。业界普遍认为，如果设备名称可配置，命名策略应由系统管理员决定，而非由开发者强制指定。devfs 文件系统还存在设计上固有的竞态条件问题，这些问题若不进行内核重大修订就无法解决。devfs 长期被标记为弃用状态，最终于 2006 年 6 月从内核中移除。

随着 2.5 版本不稳定内核树（后来发布为 2.6 系列稳定内核）的发展，一种名为 sysfs 的新型虚拟文件系统应运而生。sysfs 的职责是向用户空间进程提供系统硬件配置信息。通过这种对用户空间可见的呈现方式，开发替代 devfs 的用户空间方案成为可能。

9.3.2. Udev 实现机制

9.3.2.1. Sysfs 文件系统 前文已简要提及 sysfs 文件系统。人们可能会好奇 sysfs 如何知晓系统中存在的设备以及应为这些设备分配哪些设备号。编译进内核的驱动程序会在内核检测到设备时，将其对象注册到 sysfs（内部通过 devtmpfs 实现）。对于编译为模块的驱动程序，注册过程发生在模块加载时。当 sysfs 文件系统挂载到 /sys 目录后，驱动程序向 sysfs 注册的数据便可供用户空间进程和 udevd 处理（包括对设备节点的修改）。

9.3.2.2. 设备节点创建 设备文件由内核在 devtmpfs 文件系统中创建。任何希望注册设备节点的驱动程序都会通过 devtmpfs（经由驱动核心）来完成这一操作。当 devtmpfs 实例挂载到 /dev 目录时，设备节点最初会以固定的名称、权限和所有者暴露给用户空间。

稍后，内核会向 udevd 发送一个 uevent 消息。根据 /etc/udev/rules.d、/usr/lib/udev/rules.d 以及 /run/udev/rules.d 目录中规则文件的设定，udevd 会为该设备节点创建额外的符号链接，或修改其权限、所有者及所属组，亦或更新该对象在 udevd 内部数据库中的条目（名称）。

这三个目录中的规则都经过编号处理，且三个目录的内容会被合并使用。如果 udevd 无法为正在创建设备找到对应规则，则会保留 devtmpfs 最初设置的权限和所有权。

9.3.2.3. 模块加载 编译为模块的设备驱动可能内置了别名。这些别名可通过 modinfo 程序查看，通常与模块所支持设备的特定总线标识符相关。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*bc04sc01i*。对于大多数设备，总线驱动程序会通过 sysfs 导出能够处理该设备的驱动别名。例如，/sys/bus/pci/devices/0000:00:0d.0/modalias 文件可能包含字符串

pci:v00001319d00000801sv00001319sd00001319bc04sc01i00。udev 提供的默认规则会使 udevd 调用/sbin/modprobe，并传入 MODALIAS uevent 环境变量的内容（该内容应与 sysfs 中 modalias 文件内容一致），从而加载所有经通配符扩展后与该字符串匹配的别名模块。

在这个示例中，这意味着除了 snd-fm801 之外，如果 forte 驱动可用，那么过时（且不需要）的该驱动也会被加载。下文将介绍如何防止加载不需要的驱动程序。

内核本身也能够按需加载网络协议、文件系统和 NLS 支持的模块。

9.3.2.4. 处理可热插拔/动态设备 当您插入设备（例如通用串行总线(USB)MP3 播放器）时，内核会识别到该设备已连接并生成一个 uevent。随后该 uevent 会由上文所述的 udevd 进行处理。

9.3.3. 模块加载与创建设备的问题 在自动创建设备节点时可能会出现几个问题。

9.3.3.1. 内核模块未自动加载的情况 Udev 仅在模块具有总线特定别名且总线驱动程序正确将必要别名导出至

sysfs 时才会加载该模块。其他情况下，应通过其他方式安排模块加载。已知在 Linux-6.13.4 系统中，udev 能够为 INPUT、IDE、PCI、USB、SCSI、SERIO 和 FireWire 设备正确加载编写规范的驱动程序。

要判断所需设备驱动程序是否具备 udev 所需支持，请以模块名为参数运行 modinfo 命令。接着尝试在/sys/bus 目录下定位设备目录，并检查是否存在 modalias 文件。

若 sysfs 中存在 modalias 文件，说明驱动程序支持该设备并能直接通信，但缺少别名时属于驱动程序缺陷。此时可不依赖 udev 加载驱动，并等待后续修复该问题。

如果在/sys/bus 下的相关目录中没有 modalias 文件，这意味着内核开发者尚未为该总线类型添加 modalias 支持。在 Linux-6.13.4 版本中，ISA 总线就存在这种情况。预计该问题将在后续内核版本中得到修复。

Udev 并不用于加载诸如 snd-pcm-oss 之类的“封装”驱动以及 loop 等非硬件驱动。

9.3.3.2. 内核模块未自动加载且 Udev 不负责加载它

如果“封装器”模块仅增强其他模块的功能（例如 snd-pcm-oss 通过使声卡可供 OSS 应用程序使用来增强 snd-pcm 的功能），则应配置 modprobe 在 udev 加载被封装模块后加载该封装器。为此，需要在相应的/etc/modprobe 配置文件中添加一条“softdep”命令。例如：

```
softdep snd-pcm post: snd-pcm-oss
```

请注意，"softdep"命令也支持 pre:依赖项，或者同时包含 pre:和 post:依赖项的混合形式。
关于"softdep"语法和功能的更多信息，请参阅 modprobe.d(5)手册页。

如果所讨论的模块不是包装器模块且自身具有实用价值，请配置模块启动脚本在系统引导时加载该模块。为此，只需在/etc/sysconfig/modules 文件中另起一行添加该模块名称即可。这种方法也适用于包装器模块，但在这种情况下并非最优选择。

9.3.3.3. Udev 加载了某些不需要的模块

要么不构建该模块，要么像处理 forte 模块那样，在/etc/modprobe.d/blacklist.conf 文件中将其列入黑名单

在以下示例中：

```
blacklist forte
```

被列入黑名单的模块仍可通过显式 modprobe 命令手动加载。

9.3.3.4. Udev 错误创建设备或生成错误符号链接 这种情况通常发生在某条规则意外匹配到设备时。例如，一条编写不当的规则可能同时匹配到所需的 SCSI 磁盘和错误的对应厂商 SCSI 通用设备。借助 udevadm info 命令找到问题规则并使其更具针对性。

9.3.3.5. Udev 规则工作不稳定 这可能是前一个问题的另一种表现。若非如此，且您的规则使用了 sysfs 属性，则可能是内核时序问题，将在后续内核版本中修复。目前可以通过创建等待所用 sysfs 属性的规则来解决，将其追加到/etc/udev/rules.d/10-wait_for_sysfs.rules 文件中（若文件不存在则创建）。若此方法有效，请通知 LFS 开发邮件列表。

9.3.3.6. Udev 未创建设备 首先确认驱动程序已内置到内核或已作为模块加载，且 udev 并未创建错误命名的设备。

若内核驱动程序未将其数据导出到 sysfs，udev 将缺乏创建设备节点所需的信息。这种情况最可能发生在内核树之外的第三方驱动程序上。请在/usr/lib/

udev/devices 目录下创建具有正确主/次设备号的静态设备节点（参见内核文档中的 devices.txt 文件或第三方驱动供应商提供的文档）。静态设备节点将由 udev 复制到/dev 目录。

9.3.3.7. 设备命名顺序在重启后随机变化 这是由于 udev 在设计上采用并行方式处理 uevent 事件和加载模块，因此顺序具有不可预测性。这个问题永远不会被"修复"。你不应依赖内核设备名称的稳定性，而应创建自定义规则，基于设备的某些稳定属性（如序列号或 udev 安装的各种*_id 工具输出）来建立具有稳定名称的符号链接。具体示例可参阅第 9.4 节"设备管理"和第 9.5 节"通用网络配置"。

9.3.4. 推荐阅读资料 以下网站提供了更多实用文档：

- devfs 的用户空间实现 http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-KroahHartman-OLS2003.pdf

- sysfs 文件系统 <https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf>

9.4. 设备管理

9.4.1. 网络设备

默认情况下，Udev 根据固件/BIOS 数据或总线、插槽、MAC 地址等物理特征来命名网络设备。这种命名规范的目的是确保网络设备命名具有一致性，而非基于网卡被发现的时间顺序。在旧版 Linux 系统中——例如一台装有英特尔和瑞昱双网卡的电脑——英特尔制造的网卡可能被命名为 eth0，而瑞昱网卡则成为 eth1。重启后，这两块网卡的编号有时会互相调换。

在新命名方案中，典型的网络设备名称形如 enp5s0 或 wlp3s0。若不希望采用此命名规范，可实施传统命名方案或自定义方案。

9.4.1.1. 在内核命令行禁用持久化命名

通过在内核命令行添加 `net.ifnames=0` 参数，可以恢复传统的 eth0、eth1 等命名方案。这种方式特别适合仅配备单一类型以太网设备的系统。笔记本电脑通常会有两个以太网接口（分别命名为 eth0 和 wlan0），这类设备同样适用此方法。相关命令行配置需写入 GRUB 配置文件，具体操作参见第 10.4.4 节“创建 GRUB 配置文件”。

9.4.1.2. 创建自定义 Udev 规则

可通过创建自定义 udev 规则来定制命名方案。系统已包含一个用于生成初始规则的脚本。

执行以下命令生成规则：

```
bash /usr/lib/udev/init-net-rules.sh
```

现在，检查`/etc/udev/rules.d/70-persistent-net.rules` 文件，以确定哪个名称被分配给了哪个网络设备：

```
cat /etc/udev/rules.d/70-persistent-net.rules
```

注意

在某些情况下，例如当 MAC 地址被手动分配给网卡时，或在 Qemu 或 Xen 等虚拟环境中，由于地址未被一致分配，可能不会生成网络规则文件。在这些情况下，无法使用此方法。

该文件以注释块开头，随后是每个网卡对应的两行内容。每个网卡的第一行是注释描述，显示其硬件 ID（例如，如果是 PCI 卡，则显示其 PCI 供应商和设备 ID）及其驱动程序（如果能找到驱动程序，则显示在括号中）。硬件 ID 和驱动程序都不用于确定接口的名称；这些信息仅供参考。第二行是与该网卡匹配的 udev 规则，实际为其分配名称。所有 udev 规则都由多个关键字组成，用逗号和可选空格分隔。以下是各关键字的解释：

- `SUBSYSTEM=="net"` - 这指示 udev 忽略非网卡设备。
- `ACTION=="add"` - 这指示 udev 对非添加事件（“remove”和“change”事件也会发生，但无需重命名网络接口）忽略此规则。

- `DRIVERS=="?*"` - 此规则确保 udev 会忽略 VLAN 或桥接子接口（因为这些子接口没有驱动程序）。跳过这些子接口的原因是，若为其分配名称会与父设备发生冲突。
- `ATTR{address}` - 该关键字的值对应网卡的 MAC 地址。
- `ATTR{type}=="1"` - 该规则确保仅匹配某些无线驱动程序创建多个虚拟接口时的主接口。跳过次要接口的原因与跳过 VLAN 和桥接子接口相同：否则会发生名称冲突。
- `NAME` - 该关键字的值是 udev 将分配给此接口的名称。

`NAME` 的值是关键部分。请确保在继续操作前了解每张网卡被分配的名称，并在创建网络配置文件时务必使用该 `NAME` 值。

即使创建了自定义 udev 规则文件，udev 仍可能根据物理特性为网卡分配一个或多个备用名称。如果自定义 udev 规则尝试使用已被其他网卡作为备用名称占用的名称来重命名某张网卡，则该 udev 规则将失效。若发生此问题，您可以创建`/etc/udev/network/99-default`。

链接 使用空替代名称策略的配置文件，覆盖默认配置文件`/usr/lib/udev/network/99-default.link`：

```
sed -e '/^AlternativeNamesPolicy/s/=.*$/=/' \
    /usr/lib/udev/network/99-default.link \>
    /etc/udev/network/99-default.link
```

9.4.2. 光驱符号链接 后续可能安装的某些软件（例如各类媒体播放器）会要求存在`/dev/cdrom` 和 `/dev/dvd` 这两个符号链接，并指向实际的 CD-ROM 或 DVD-ROM 设备。此外，将这些符号链接的引用写入`/etc/fstab` 文件也会带来便利。Udev 自带一个脚本，能根据每个设备的特性生成规则文件来创建这些符号链接，但您需要决定让脚本采用两种工作模式中的哪一种。

第一种是“by-path”模式（USB 和 FireWire 设备默认采用此模式），生成的规则取决于光驱设备的物理路径。第二种是“by-id”模式（IDE 和 SCSI 设备默认采用此模式），生成的规则则取决于存储在光驱设备本身的识别字符串。路径由 udev 的 `path_id` 脚本确定，而识别字符串则由 `ata_id` 或 `scsi_id` 程序从硬件读取，具体取决于您拥有的设备类型。

每种方法各有优势；正确的方法取决于可能发生的设备变更类型。如果您预期设备的物理路径（即其连接的端口和/或插槽）会发生变化，例如计划将驱动器移至不同的 IDE 端口或 USB 接口，则应使用“byid”模式。另一方面，如果您预期设备标识会改变，例如设备可能损坏，并打算用连接到相同接口的不同设备替换它，则应使用“by-path”模式。如果您的驱动器可能发生任一类变更，则应根据预期更频繁发生的变更类型选择模式。

重要提示

外部设备（例如 USB 连接的 CD 驱动器）不应使用 by-path 持久化，因为每次将设备插入新的外部端口时，其物理路径都会改变。如果您编写 udev 规则通过物理路径识别所有外部连接设备，都会遇到这个问题；该问题不仅限于 CD 和 DVD 驱动器。

若想查看 udev 脚本将使用的值, 请针对相应的 CD-ROM 设备, 在/sys 目录下找到对应的子目录 (例如可能是/sys/block/hdd), 然后运行类似以下的命令:

```
udevadm test /sys/block/hdd
```

观察包含各*_id 程序输出的行。"by-id"模式会优先使用 ID_SERIAL 值 (若该值存在且非空), 否则将组合使用 ID_MODEL 和 ID_REVISION 值。"by-path" 模式则会采用 ID_PATH 值。

如果默认模式不适合您的情况, 可以按如下方式修改/etc/udev/rules.d/83-cdrom-symlinks.rules 文件 (其中 mode 参数可选用"by-id"或"by-path"模式):

```
sed -e 's/"write_cd_rules"/"write_cd_rules mode"/' \
-i /etc/udev/rules.d/83-cdrom-symlinks.rules
```

请注意, 此时无需创建规则文件或符号链接, 因为你已将主机的/dev 目录绑定挂载到 LFS 系统中, 且我们假设这些符号链接已存在于主机上。这些规则和符号链接将在你首次启动 LFS 系统时自动创建。

不过, 若你拥有多个 CD-ROM 设备, 由于设备发现顺序不可预测, 此时生成的符号链接可能指向与主机不同的设备。首次启动 LFS 系统时创建的设备分配关系将是稳定的, 因此仅当需要两个系统的符号链接指向同一设备时才会产生问题。若确需如此, 请检查 (必要时编辑) 生成的/etc/udev/rules.d/70-persistent-

cd.rules 系统启动后检查该文件, 确保分配的符号链接符合您的需求。

9.4.3. 处理重复设备 如第 9.3 节"设备和模块处理概述"所述, 功能相同的设备在/dev 目录中的出现顺序本质上是随机的。例如, 若您同时拥有 USB 网络摄像头和电视调谐器, 有时/dev/

video0 指向摄像头而/dev/video1 指向调谐器, 但重启后顺序可能发生改变。除声卡和网卡外, 所有硬件类别均可通过创建 udev 规则生成持久化符号链接来解决此问题。网卡配置详见第 9.5 节"通用网络配置", 声卡配置则可在 BLFS 手册中查阅。

对于每台可能存在此问题的设备 (即使您当前的 Linux 发行版中不存在该问题), 请在/sys/class 或/sys/block 下找到对应的目录。对于视频设备, 可能是/sys/class/video4linux/videoX。找出能唯一标识该设备的属性 (通常使用厂商 ID、产品 ID 和/或序列号即可) :

```
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"
# 为摄像头和调谐器创建持久符号链接
KERNEL=="video*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f", ATTRS{vendor}=="0x109e", SYMLINK+="tvtuner"
EOF
```

最终效果是: /dev/video0 和/dev/video1 设备仍会随机指向调谐器或网络摄像头 (因此永远不应直接使用), 但符号链接/dev/tvtuner 和/dev/webcam 将始终指向正确的设备。

9.5. 通用网络配置

9.5.1. 创建网络接口配置文件 /etc/sysconfig/ 目录中的文件通常决定网络脚本会启用或关闭哪些接口。该目录应为每个需要配置的接口包含一个文件，例如 ifconfig.xyz，其中“xyz”描述网卡。通常使用接口名称（如 eth0）是合适的。每个文件包含一个接口的属性，例如其 IP 地址、子网掩码等。文件名必须以 ifconfig 开头。

注意

如果未使用前一节中的步骤，udev 将根据系统物理特征（如 enp2s1）分配网卡接口名称。如果不确定接口名称是什么，可以在系统启动后运行 ip link 或 ls /sys/class/net 命令查看。

接口名称取决于系统上运行的 udev 守护程序的实现和配置。LFS 系统中的 udev 守护程序（安装于第 8.76 节“Systemd-257.3 中的 Udev”）在 LFS 系统启动前不会运行。因此即使是在 chroot 环境中，也无法始终通过在主发行版上运行这些命令来确定 LFS 系统中的接口名称。

以下命令为 eth0 设备创建了一个使用静态 IP 地址的示例配置文件：

```
cd /etc/sysconfig/ cat > ifconfig.eth0
<< "EOF" ONBOOT=yes IFACE=eth0
SERVICE=ipv4-static
```

```
IP=192.168.1.2
GATEWAY=192.168.1.1
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

斜体标注的数值需在每个文件中进行修改，以确保正确配置网络接口。

若将 ONBOOT 变量设为 yes，System V 网络脚本将在系统启动过程中激活网络接口卡(NIC)。若设为 yes 以外的任何值，该 NIC 将被网络脚本忽略且不会自动启动。可通过 ifup 和 ifdown 命令手动启停网络接口。

IFACE 变量用于定义接口名称（例如 eth0）。所有网络设备配置文件都必须包含此变量，且文件扩展名必须与该值匹配。

SERVICE 变量定义获取 IP 地址的方式。LFS-Bootscripts 软件包采用模块化 IP 分配机制，通过在/lib/services/ 目录下创建附加文件可支持其他 IP 分配方法。

该功能通常用于动态主机配置协议(DHCP)，相关配置方法详见 BLFS 手册。

若存在默认网关 IP 地址，则 GATEWAY 变量应包含该地址。若无，请将该变量整行注释掉。

PREFIX 变量指定子网所用的比特位数。IP 地址每个段占 8 比特。若子网掩码为 255.255.255.0，则表示使用前三个段（24 比特）指定网络号；若掩码为 255.255.255.240，则表示使用前 28 比特。DSL 和有线互联网服务提供商 (ISP) 通常使用超过 24 比特的前缀长度。本例中（PREFIX=24）对应的子网掩码为 255.255.255.0。

请根据实际子网配置调整 PREFIX 变量。若未指定，该变量默认值为 24。

更多信息请参阅 ifup 的手册页。

9.5.2. 创建/etc/resolv.conf 文件

系统需要某种机制来获取域名服务 (DNS) 名称解析功能，以实现互联网域名与 IP 地址的相互转换。最佳实现方式是将 DNS 服务器的 IP 地址（可从

将 ISP 或网络管理员提供的 DNS 服务器信息写入/etc/resolv.conf 文件。通过运行以下命令创建该文件：

```
cat > /etc/resolv.conf << "EOF"
# 开始 /etc/resolv.conf 配置

domain <您的域名> nameserver <主域名服务器 IP 地址> nameserver <备用域
名服务器 IP 地址>

# 结束 /etc/resolv.conf 配置
EOF
```

domain 语句可以省略或用 search 语句替代。更多详情请参阅 resolv.conf 的手册页。

将替换为最适合当前配置的 DNS 服务器 IP 地址。通常会有多个条目（系统要求必须配置备用服务器以确保容错能力）。如果只需要或想要一个 DNS 服务器，请从文件中删除第二个 nameserver 行。该 IP 地址也可以是本地网络中的路由器地址。

注意

谷歌公共 IPv4 DNS 地址为 8.8.8.8 和 8.8.4.4。

9.5.3. 配置系统主机名 在启动过程中，系统会使用/etc/hostname 文件来设置主机名。

通过运行以下命令创建/etc/hostname 文件并输入主机名：

```
echo "<lfs>" > /etc/hostname
```

需要替换为计算机名称。此处不要输入完全限定域名 (FQDN)，该信息应填写在/etc/hosts 文件中。

9.5.4. 自定义/etc/hosts 文件 确定要使用的完全限定域名(FQDN)及可能的别名。如果使用静态 IP

地址，还需确定 IP 地址。hosts 文件条目的语法格式为：

```
IP 地址 myhost.example.org 别名
```

除非该计算机需要对外公开（即拥有注册域名和有效的 IP 地址分配块——大多数用户并不具备此条件），请确保 IP 地址位于私有网络地址范围内。有效范围如下：

私有网络地址范围	常规前缀
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 至 31 范围内的任意数字。y 可以是 0 至 255 范围内的任意数字。

一个有效的私有 IP 地址可以是 192.168.1.2。

若计算机需接入互联网，有效的 FQDN（完全限定域名）可以是域名本身，或是将前缀（通常为主机名）与域名通过“.”连接而成的字符串。此外，您需要联系域名提供商将该 FQDN 解析至您的公网 IP 地址。

即使计算机不接入互联网，某些程序（如邮件传输代理 MTA）仍需 FQDN 才能正常运行。为此可使用特殊 FQDN：localhost.localdomain。

通过以下命令创建 /etc/hosts 文件：

```
cat > /etc/hosts << "EOF"
# 开始 /etc/hosts 文件内容

127.0.0.1 localhost.localdomain localhost
127.0.1.1 <主机名> <192.168.1.2> <主机名> [别名 1] [别名 2 ...] ::1
localhost ip6-localhost ip6-loopback ff02::1 ip6-allnodes ff02::2 ip6-
allrouters
```

```
# /etc/hosts 文件结束
EOF
```

其中<192.168.1.2>、和<主机名>需根据具体用途或要求修改（若网络/系统管理员已分配 IP 地址且主机将接入现有网络）。可选的别名可省略不填。

9.6. System V 启动脚本使用与配置

9.6.1. System V 启动脚本如何工作？ 本版 LFS 采用基于运行级别的 SysVinit 特殊启动机制。不同系统的启动流程可能差异显著；某个 Linux 发行版的特定工作方式并不意味着在 LFS 中也会同样运作。LFS 有其独特实现方式，但仍遵循普遍接受的标准。

另有一种称为 systemd 的替代启动方案。此处不再详述该启动流程，具体说明可访问 <https://www.linux.com/training-tutorials/understanding-and-using-systemd/>。

SysVinit（下文简称“init”）采用运行级别机制。共设 7 个运行级别（0 至 6）。（实际存在更多运行级别，但其余用于特殊场景且通常不使用，详见 init(8) 手册）。每个级别对应计算机启动或关机时应执行的操作，默认运行级别为 3。以下是 LFS 中各级别的具体定义：

- 0: 关机
- 1: 单用户模式
- 2: 保留用于自定义（默认功能与 3 相同）
- 3: 带网络连接的多用户模式
- 4: 保留用于自定义（默认功能与 3 相同）
- 5: 与 4 相同，通常用于图形界面登录（如 GNOME 的 gdm 或 LXDE 的 lxdm）
- 6: 重启计算机

注意

传统上，上述运行级别 2 被定义为“不带网络支持的多用户模式”，但这仅适用于多年前用户可通过串行端口连接系统的场景。在当前环境下已无实际意义，因此我们现在将其定义为“保留状态”。

9.6.2. 配置 SysVinit

在内核初始化过程中，首个运行的程序（若未在命令行中覆盖）是 init。该程序

会读取初始化文件/etc/inittab。使用以下命令创建该文件：

```
cat > /etc/inittab << "EOF"
# 开始 /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc S

10:0:wait:/etc/rc.d/init.d/rc 0
11:S1:wait:/etc/rc.d/init.d/rc 1
12:2:wait:/etc/rc.d/init.d/rc 2
13:3:wait:/etc/rc.d/init.d/rc 3
14:4:wait:/etc/rc.d/init.d/rc 4
15:5:wait:/etc/rc.d/init.d/rc 5
16:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S06:once:/sbin/sulogin
s1:1:respawn:/sbin/sulogin

1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# 结束 /etc/inittab 文件
EOF
```

该初始化文件的详细说明可查阅 inittab 手册页。在 LFS 系统中，核心命令是 rc。上述初始化文件指示 rc 先运行 /etc/rc.d/rcS.d 目录下所有以 S 开头的脚本，再运行 /etc/rc.d/rc?.d 目录下所有以 S 开头的脚本（问号位置由 initdefault 值指定）。

为方便使用，rc 脚本会加载 /lib/lsb/init-functions 函数库。该函数库还会读取可选配置文件 /etc/sysconfig/rc.site。后续章节描述的任何系统配置参数均可置于此文件，从而实现所有系统参数的集中管理。为便于调试，函数脚本还会将所有输出记录至 /run/var/bootlog。由于 /run 目录属于 tmpfs 文件系统，该日志不会在重启后保留；但其内容会被追加到更持久的 /var/log/boot 文件中。

日志 在启动过程结束时。

9.6.2.1. 切换运行级别

切换运行级别使用 init <运行级别> 命令，其中<运行级别>是目标运行级别。例如，要重启计算机，用户可以执行 init 6 命令，这是 reboot 命令的别名。同样地，init 0 是 halt 命令的别名。

在/etc/rc.d 目录下存在多个形如 rc?.d (其中?代表运行级别数字) 的子目录，以及 rcS.d 目录，这些目录内都包含若干符号链接。部分链接以 K 开头，其余以 S 开头，且所有链接名在首字母后都跟随两个数字。K 表示停止 (kill) 某项服务，S 表示启动某项服务。数字决定了脚本的执行顺序，范围从 00 到 99——数字越小，脚本越早执行。当 init 切换到另一个运行级别时，将根据所选运行级别启动或停止相应的服务。

实际脚本存放在/etc/rc.d/init.d 目录中。这些脚本执行实际工作，所有符号链接都指向它们。K 链接和 S 链接实际上指向/etc/rc.d/init.d 中的同一个脚本，因为这些脚本可以通过不同参数调用，如 start (启动)、stop (停止)、restart (重启)、reload (重载) 和 status (状态)。当遇到 K 链接时，对应脚本会以 stop 参数运行；遇到 S 链接时，则以 start 参数运行。

以下是各参数对脚本行为的说明：

启动

服务已启动。

停止

服务已停止。

重新启动

服务停止后再次启动。

重新加载

更新服务的配置。当修改了服务的配置文件且无需重启服务时使用此操作。

状态

显示服务是否正在运行及其对应的进程 ID(PID)。

您可以自由修改系统启动流程的工作方式(毕竟这是您自己的 LFS 系统)。这里提供的文件仅作为实现方式的示例参考。

9.6.3. Udev 启动脚本 /etc/rc.d/init.d/udev 初始化脚本会启动 udevd 守护进程，触发内核已创建的“冷插拔”设备，并等待所有规则执行完毕。该脚本还会将默认的 uevent 处理器从/sbin/路径取消设置。

热插拔。这样做是因为内核不再需要调用外部二进制程序，而是由 udevd 监听内核发出的 uevents 网络套接字。/etc/rc.d/init.d/udev_retry 脚本负责重新触发那些规则可能依赖于 mountfs 脚本运行后才挂载的文件系统（特别是/usr 和/var 可能引发这种情况）的子系统事件。该脚本在 mountfs 脚本之后运行，因此这些规则（如果被重新触发）在第二次尝试时应该会成功。它由/etc/sysconfig/udev_retry 文件配置；该文件中除注释外的任何单词都被视为重试时要触发的子系统名称。要查找设备的子系统，请使用 udevadm info --attribute-walk 命令，其中是/dev 或/sys 中的绝对路径，例如/dev/sr0 或/sys/class/rtc。

关于内核模块加载和 udev 的信息，请参见第 9.3.2.3 节“模块加载”。

9.6.4. 配置系统时钟

setclock 脚本会从硬件时钟（也称为 BIOS 或互补金属氧化物半导体 CMOS 时钟）读取时间。若硬件时钟设置为 UTC 时间，该脚本将利用/etc/localtime 文件（该文件告知 hwclock 程序应使用的时区）将硬件时钟时间转换为本地时间。由于无法自动检测硬件时钟是否设置为 UTC 时间，因此必须手动进行此项配置。

`setclock` 程序会在内核检测到硬件启动能力时通过 `udev` 运行。也可以手动运行该程序并使用 `stop` 参数将系统时间存储到 CMOS 时钟中。

如果不确定硬件时钟是否设置为 UTC 时间，可以通过运行 `hwclock --localtime --show` 命令来确认。这将显示硬件时钟当前的时间。如果该时间与手表显示的时间一致，则硬件时钟设置为本地时间。如果 `hwclock` 的输出不是本地时间，则很可能设置为 UTC 时间。通过对 `hwclock` 显示的时间加减所在时区的适当小时数来验证这一点。例如，如果当前处于 MST 时区（即 GMT -0700），则应在本地时间基础上加 7 小时。

如果硬件时钟未设置为 UTC 时间，请将下方 UTC 变量的值更改为 0（零）。

执行以下命令创建新文件 /etc/sysconfig/clock：

```
cat > /etc/sysconfig/clock << "EOF"
# 开始 /etc/sysconfig/clock 配置

UTC=1

# 此处可设置需要传递给 hwclock 的任何参数,
# 例如 Alpha 架构机器硬件时钟类型的参数。CLOCKPARAMS=

# /etc/sysconfig/clock 文件结束
EOF
```

关于在 LFS 系统中处理时间设置的建议可参考：

<https://www.linuxfromscratch.org/hints/downloads/files/time.txt>。该文档详细说明了时区、协调世界时(UTC)以及 TZ 环境变量等相关问题。

注意

`CLOCKPARAMS` 和 `UTC` 参数也可在 /etc/sysconfig/rc.site 文件中进行设置。

9.6.5. 配置 Linux 控制台

本节讨论如何配置控制台启动脚本，该脚本用于设置键盘映射、控制台字体和控制台内核日志级别。如果不使用非 ASCII 字符（如版权符号、英镑符号和欧元符号）且键盘为美式布局，则可跳过本节大部分内容。若缺少配置文件（或 `rc.site` 中的等效设置），控制台启动脚本将不会执行任何操作。

控制台脚本会读取 /etc/sysconfig/console 文件获取配置信息。首先确定要使用的键盘映射和屏幕字体。各类语言特定的 HOWTO 文档也可提供帮助，参见 <https://tldp.org/HOWTO/HOWTO-INDEX/other-lang.html>。若仍有疑问，可查看 /usr/share/keymaps 和 /usr/share/consolefonts 目录下的有效键盘映射和屏幕字体。通过阅读 `loadkeys(1)` 和 `setfont(8)` 手册页来确定这些程序的正确参数。

/etc/sysconfig/console 文件应包含以下格式的行：VARIABLE=值。以下是可识别的变量：

`LOGLEVEL` 该变量指定通过 `dmesg -n` 命令设置的发送至控制台的内核消息日志级别。有效级别范围为 1（不显示消息）至 8，默认级别为 7（显示详尽信息）。`KEYMAP` 该变量用于指定 `loadkeys` 程序的参数，通常是要加载的键盘映射名称，例如

`int`。若未设置此变量，引导脚本将不会运行 `loadkeys` 程序，系统将使用默认内核键盘映射

请注意，部分关键键位映射存在多个同名版本（例如 `qwerty/` 和 `qwertz/` 目录下的 `cz` 及其变体，`olpc/` 和 `qwerty/` 目录下的 `es`，以及 `fgGioc/` 和 `qwerty/` 目录下的 `trf`）。遇到这种情况时，必须同时指定父目录（例如 `qwerty/es`）以确保加载正确的键位映射。

`KEYMAP_CORRECTIONS` 这个（极少使用的）变量用于指定第二次调用 `loadkeys` 程序时的参数。当标准键位映射不完全符合需求且需要微调时，该功能非常实用。例如，若要在原本不含欧元符号的键位映射中添加该符号，可将此变量设为 `euro2`。

`FONT` 此变量用于指定 `setfont` 程序的参数。通常包括字体名称、`-m` 参数以及要加载的应用字符映射名称。例如，若要同时加载“lat1-16”字体和适用于美国的“8859-1”

应用字符映射，应将该变量设为 `lat1-16 -m 8859-1`。在 UTF-8 模式下，内核会使用应用字符映射将 8 位键码转换为 UTF-8 编码。因此“m”参数的取值应设置为键位映射中组合键码的编码格式。

将 `UNICODE` 变量设置为 1、yes 或 true 可将控制台切换至 UTF-8 模式。这在基于 UTF-8 的区域设置中非常有用，但在其他情况下可能产生问题。

`LEGACY_CHARSET` 对于许多键盘布局，`Kbd` 软件包中并未提供现成的 Unicode 键位映射表。若将此变量设置为现有非 UTF-8 键位映射表的编码，控制台启动脚本会实时将其转换为 UTF-8 格式。

而默认的内核键位映射表会以下是一些示例：

- 我们将在第 9.7 节“配置系统区域设置”中使用 C.UTF-8 作为 Linux 控制台交互会话的区域设置，因此应将 `UNICODE` 设为 1。`Kbd` 软件包提供的控制台字体（包含 C.UTF-8 区域设置中程序消息所需的所有字符字形）是 `LatArCyrHeb*.psfu.gz` 和 `LatGrkCyr*`。

`psfu.gz`、`Lat2-Terminus16.psfu.gz` 和 `pancyrilllic.f16.psfu.gz` 位于 `/usr/share/consolefonts` 目录下（其他随附的控制台字体缺少某些字符的字形（如 Unicode 左右引号和 Unicode 英文破折号）。因此请将其中一个字体（例如 `Lat2-Terminus16.psfu.gz`）设为默认控制台字体：

```
cat > /etc/sysconfig/console << "EOF"
# 开始 /etc/sysconfig/console

UNICODE="1" FONT="Lat2-
Terminus16"

# /etc/sysconfig/console 文件结束
EOF
```

- 对于非 Unicode 配置，通常只需要设置 `KEYMAP` 和 `FONT` 变量。例如，波兰语配置会使用如下设置：

```
cat > /etc/sysconfig/console << "EOF"
# 开始 /etc/sysconfig/console 配置

KEYMAP="pl2" FONT="lat2a-16 -m
8859-2"

# 结束 /etc/sysconfig/console 配置
EOF
```

- 如上所述，有时需要对标准键盘映射进行微调。以下示例演示如何在德语键盘映射中添加欧元符号：

```
cat > /etc/sysconfig/console << "EOF"
# 开始 /etc/sysconfig/console 配置

KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
FONT="lat0-16 -m 8859-15"
UNICODE="1"

# 结束 /etc/sysconfig/console
EOF
```

- 以下是针对保加利亚语启用了 Unicode 的示例（已有现成的 UTF-8 键盘映射）：

```
cat > /etc/sysconfig/console << "EOF"
# 开始 /etc/sysconfig/console

UNICODE="1" KEYMAP="bg_bds-
utf8" FONT="LatArCyrHeb-16"

# /etc/sysconfig/console 文件结束
EOF
```

- 由于前例使用了包含 512 个字符的 LatArCyrHeb-16 字体，除非使用帧缓冲设备，否则 Linux 控制台将无法显示明亮色彩。若用户希望在不使用帧缓冲的情况下保持明亮色彩，且能接受不显示非本语言字符，仍可采用特定语言的 256 字符字体，如下所示：

```
cat > /etc/sysconfig/console << "EOF"
# 开始 /etc/sysconfig/console 配置

UNICODE="1" KEYMAP="bg_bds-
utf8" FONT="cyr-sun16"

# 结束 /etc/sysconfig/console 配置
EOF
```

- 以下示例展示了从 ISO-8859-15 到 UTF-8 的键位映射自动转换，以及在 Unicode 模式下启用死键的功能：

```
cat > /etc/sysconfig/console << "EOF"
# 开始 /etc/sysconfig/console 配置

UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
FONT="LatArCyrHeb-16 -m 8859-15"

# 结束 /etc/sysconfig/console 文件
EOF
```

- 某些键盘映射包含死键（即单独按下时不产生字符，但会对下一个按键产生的字符添加重音符号）或定义了组合规则（例如：在默认键盘映射中“按下 Ctrl+. A E 可得到Æ”）。Linux-6.13.4 仅当需要组合的源字符不是多字节字符时，才能正确解释键盘映射中的死键和组合规则。这一缺陷不会影响欧洲语言的键盘映射，因为其重音符号是添加到无重音的 ASCII 字符上，或是将两个 ASCII 字符组合

组合在一起。但在 UTF-8 模式下这会成为问题；例如希腊语中有时需要在字母上添加重音符号^a。解决方案要么避免使用 UTF-8 编码，要么安装 X 窗口系统——其输入处理不存在此限制。

- 对于中文、日文、韩文等语言，Linux 控制台无法配置显示所需字符。需要使用这些语言的用户应当安装 X Window System、覆盖必要字符范围的字体以及正确的输入法（例如 SCIM 支持多种语言）。

注意

/etc/sysconfig/console 文件仅控制 Linux 文本控制台的本地化设置。它与 X Window System 中的键盘布局/终端字体设置、ssh 会话或串行控制台无关。上述最后两项列出的限制在这些情况下均不适用。

9.6.6. 启动时创建文件

有时需要在系统启动时创建特定文件。例如，经常需要建立`/tmp/.ICE-unix`目录。这可以通过在`/etc/sysconfig/createfiles`配置脚本中添加条目来实现，该文件的格式说明已内置于默认配置文件的注释中。

9.6.7. 配置 Sysklogd 脚本

sysklogd 脚本作为 System V 初始化的一部分会调用 syslogd 程序。参数`-m 0`用于关闭默认情况下 syslogd 每 20 分钟写入日志文件的周期性时间戳标记。若需启用此标记，请编辑`/etc/sysconfig/rc.site`文件，并将变量`SYSKLOGD_PARMS`设置为所需

值。例如，若要清除所有参数，可将该变量设为空值：

```
SYSKLOGD_PARMS=
```

更多选项请参阅 syslogd 手册页。

9.6.8. rc.site 文件

可选的 /etc/sysconfig/rc.site 文件包含为每个 SystemV 启动脚本自动设置的参数。该文件也可替代 /etc/sysconfig/ 目录下 hostname、console 和 clock 文件中指定的值。若相关变量同时存在于这些独立文件和 rc.site 中，则以脚本专属文件中的值为准。

rc.site 还包含可定制启动过程其他方面的参数。设置 IPROMPT 变量将启用启动脚本的选择性运行。其他选项详见文件注释。该文件默认版本如下：

```
# rc.site
# 启动脚本的可选参数

# 发行版信息 # 此处指定的值将覆盖默认设置 #DISTRO="Linux From Scratch" # 发行版名称
#DISTRO_CONTACT="lfs-dev@lists.linuxfromscratch.org" # 错误报告地址 #DISTRO_MINI="LFS" # 用于发行版
# 配置文件的简称

# 定义屏幕消息中使用的自定义颜色

# 更多信息请查阅 `man console_codes` 手册 # 参考"ECMA-48 图形渲染设置"章节 #
```

章节 # # 警告：当从 8 位字体切换至 9 位字体时，# Linux 控制台会将粗体显示
(1;) 重新解释为# 9 位字体的顶部 256 个字形。此情况不

影响帧缓冲控制台

```
# 若在此处指定这些值，将覆盖默认设置 #BRACKET="\033[1;34m" # 蓝色
#FAILURE="\033[1;31m" # 红色 #INFO="\033[1;36m" # 青色
#NORMAL="\033[0;39m" # 灰色
```

```
#SUCCESS="\033[1;32m" # 绿色
#WARNING="\033[1;33m" # 黄色
```

```
# 使用彩色前缀 # 此处指定的值会覆盖默认值 #BMPREFIX=
#SUCCESS_PREFIX="${SUCCESS} * ${NORMAL}"
#FAILURE_PREFIX="${FAILURE}*****${NORMAL}"
```

```
#WARNING_PREFIX="${WARNING} *** ${NORMAL}"
```

手动设置消息输出的右边界（字符数） # 在启动过程中重置控制台字体时非常有用，可覆盖 # 自动屏幕宽度检测

```
#COLUMNS=120
```

```
# 交互式启动 #IPROMPT="yes" # 是否显示交互式启动提示 #itime="3" # 显示提示的持续时间（秒）
# 发行版欢迎字符串的总长度（不含转义码） #wlen=$(echo "欢迎使用 ${DISTRO}" | wc -c )
#welcome_message="欢迎使用 ${INFO} ${DISTRO} ${NORMAL}"
```

```
# 交互式字符串的总长度（不含转义码） #ilen=$(echo "按' I' 键进入交互式启动" | wc -c )
#i_message="按' ${FAILURE} I${NORMAL}' 键进入交互式启动"
```

设置脚本在重启时跳过文件系统检查 #FASTBOOT=yes

跳过从控制台读取 #HEADLESS=yes

```
# 若设为 yes 则显示 fsck 进度
#VERBOSE_FSCK=no
```

加速启动过程，不等待 udev 设备就绪 #OMIT_UDEV_SETTLE=y

加速启动过程，不等待 udev_retry 设备就绪 #OMIT_UDEV_RETRY_SETTLE=yes

```
# 若设为 yes 则跳过清理/tmp 目录
#SKIPTMPCLEAN=no
```

```
# 针对 setclock
#UTC=1
#CLOCKPARAMS=
```

针对 consolelog（注意默认值 7=调试级别，输出较冗长） #LOGLEVEL=7

针对网络配置

```
#HOSTNAME=mylfs
# 关机时 TERM 与 KILL 信号之间的延迟 #KILLDELAY=3
```

```
# 可选的 sysklogd 参数 #SYSKLOGD_PARMS="-m 0"
# 控制台参数 #UNICODE=1 #KEYMAP="de-latin1" #KEYMAP_CORRECTIONS="euro2" #FONT="lat0-16 -m 8859-15" #LEGACY_CHARSET=
```

9.6.8.1. 自定义启动和关机脚本 LFS 启动脚本以相当高效的方式引导和关闭系统，但您可以通过调整 rc.site 文件中的设置来进一步提升速度，并根据个人偏好调整提示信息。为此，请修改上方/etc/sysconfig/rc.site 文件中的配置项。

- 在启动脚本 udev 执行期间，会调用 udev settle 命令，该命令需要一定时间才能完成。根据系统中设备的不同，这个等待时间可能是必要的也可能并非必需。如果您的系统仅包含简单分区和单张网卡，启动过程很可能无需等待该命令。若要跳过此步骤，请设置变量 OMIT_UDEV_SETTLE=y。
- 启动脚本 udev_retry 默认也会执行 udev settle 命令。仅当/var 目录被单独挂载时才需要此命令，因为时钟功能需要访问/var/lib/hwclock/adjtime 文件。其他自定义配置可能也需要等待 udev 完成操作，但在多数安装环境中并无必要。通过设置变量 OMIT_UDEV_RETRY_SETTLE=y 可跳过该命令。
- 默认情况下，文件系统检查会以静默模式运行。这在启动过程中可能表现为延迟现象。若要显示 fsck 的输出信息，请设置变量 VERBOSE_FSCK=y。
- 重启系统时，您可能希望完全跳过文件系统检查（fsck）。要实现这一点，可以创建/f 使用命令 `/sbin/shutdown -f -r now` 快速启动或重启系统。另外，您也可以通过创建 `/forcefsck` 文件或使用 `-F` 参数（而非 `-f`）运行 shutdown 命令来强制检查所有文件系统。
设置变量 `FASTBOOT=y` 将在启动过程中禁用 fsck 检查，直到该变量被移除。不建议长期保持此设置。
- 通常情况下，系统启动时会删除 `/tmp` 目录下的所有文件。根据存在的文件或目录数量，这可能导致明显的启动延迟。若要跳过清理这些文件，可设置变量 `SKIPTMPCLEAN=y`。
- 在关机过程中，init 程序会向每个已启动的程序（如 getty）发送 TERM 信号，等待预设时间（默认为 3 秒），然后向每个进程发送 KILL 信号并再次等待。对于未被自身脚本关闭的进程，sendsignals 脚本会重复此过程。可通过传递参数调整 init 的等待延迟，例如使用 `-t0` 参数（如 `/sbin/shutdown -t0 -r now`）可消除 init 的等待延迟。若要跳过 sendsignals 脚本的延迟，可设置参数 `KILLDELAY=0`。

9.7. 配置系统区域设置

要实现本地语言支持，需要设置一些必要的环境变量。正确配置这些变量将带来以下效果：

- 程序输出内容翻译为您的母语
- 将字符正确分类为字母、数字及其他类别。这对于 bash 在非英语语言环境中正确处理命令行中的非 ASCII 字符是必需的
- 符合该国标准的字母排序顺序
- 合适的默认纸张尺寸
- 货币、时间和日期值的正确格式

将下面的替换为所需语言的两位字母代码（例如 en），替换为相应国家的两位字母代码（例如 GB）。应替换为所选区域设置的规范字符集。

可能还包含可选修饰符，例如@euro。

可通过运行以下命令获取 Glibc 支持的所有区域设置列表：

```
locale -a
```

字符映射表可能拥有多个别名，例如 ISO-8859-1 也可被称为 iso8859-1 或 iso88591。某些应用程序无法正确处理这些同义词变体（例如要求必须将 UTF-8 写作 UTF-8 而非 utf8），因此在多数情况下选择区域设置的规范名称最为稳妥。要确定规范名称，请运行以下命令，其中替换为您首选区域设置（示例中为 en_GB）通过 locale -a 命令输出的名称。

iso88591 (以我们的示例为例)

```
LC_ALL=<区域设置名称> locale charmap
```

对于 en_GB.iso88591 区域设置，上述命令将输出：

```
ISO-8859-1
```

这将生成最终的区域设置 en_GB.ISO-8859-1。关键是要在使用上述启发式方法找到区域设置后，先进行测试再将其添加到 Bash 启动文件中：

```
LC_ALL=<区域设置名称> locale language LC_ALL=<区域设置名称>
locale charmap LC_ALL=<区域设置名称> locale int_curr_symbol
LC_ALL=<区域设置名称> locale int_prefix
```

上述命令应输出语言名称、该区域设置使用的字符编码、本地货币以及拨打国际电话时需要加拨的国家前缀码。如果任何命令出现类似下方所示的错误信息，意味着您所需的区域设置要么未在第 8 章中安装，要么不受当前 Glibc 默认安装支持。

```
locale: 无法将 LC_* 设置为默认区域：没有那个文件或目录
```

若出现此情况，您应使用 localedef 命令安装所需区域设置，或考虑选择其他区域设置。后续操作指南将默认 Glibc 不会产生此类错误信息。

如果区域设置名称不符合预期，其他软件包也可能出现功能异常（但未必会显示错误信息）。这种情况下，研究其他 Linux 发行版如何支持您的区域设置可能会提供有用信息。

Shell 程序/bin/bash（以下简称“shell”）通过一系列启动文件来帮助构建运行环境。每个文件都有特定用途，可能对登录环境和交互环境产生不同影响。/etc 目录中的文件提供全局设置，若用户主目录中存在同名文件，则可能覆盖全局设置。

交互式登录 shell 在成功登录后启动，通过/bin/login 读取/etc/passwd 文件。交互式非登录 shell 则在命令行启动（例如[提示符]\$/bin/bash）。非交互式 shell 通常在运行 shell 脚本时出现，它之所以是非交互的，是因为它正在处理脚本而非在命令之间等待用户输入。

一旦确定了正确的区域设置，就创建/etc/profile 文件来设置所需的区域，但要设置为 C 区域。

如果在 Linux 控制台中运行，则改用 UTF-8 区域设置（以防止程序输出 Linux 控制台无法渲染的字符）：

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

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

if [[ "$TERM" = linux ]]; then export
LANG=C.UTF-8 else

export LANG=<语言代码>_<国家代码>.<字符集><@修饰符> fi

# 结束 /etc/profile 文件
EOF
```

C（默认）和 en_US（美国英语用户推荐使用）这两种区域设置存在差异。C 区域采用 USASCII 7 位字符集，会将高位设置的字节视为无效字符。这就是为什么在该区域设置下，诸如 ls 命令会用问号替代这些字符。此外，若尝试通过 Mutt 或 Pine 发送包含此类字符的邮件，会导致发出的邮件不符合 RFC 标准（外发邮件的字符集会被标记为未知的 8-位）。建议仅在确定永远不需要使用 8 位字符时，才选用 C 区域设置。

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。其工作原理是将键盘输入转换为特定操作。Readline 被 bash 和大多数其他 shell 以及许多其他应用程序所使用。

大多数人不需要用户特定的功能，因此以下命令将创建一个全局的/etc/inputrc 文件，供所有登录用户使用。如果之后您决定需要基于每个用户覆盖默认设置，可以在用户的主目录中创建一个包含修改后映射的.inputrc 文件。

有关如何编辑 inputrc 文件的更多信息，请参阅 info bash 中 Readline Init File 部分的内容。info readline 也是一个很好的信息来源。

以下是一个通用的全局 inputrc 文件配置及其注释说明，用于解释各项功能的作用。请注意，注释不能与命令写在同一行。使用以下命令创建该文件：

```
cat > /etc/inputrc << "EOF"
# 开始 /etc/inputrc # 由 Chris Lynn 修改

# 允许命令提示符自动换行显示
set horizontal-scroll-mode Off

# 启用 8 位输入字符集
set meta-flag On
set input-meta On

# 关闭第 8 位截断功能
set convert-meta Off

# 保留第 8 位用于显示
set output-meta On

# 铃声模式：无/可见/可听
set bell-style none

# 以下所有内容将第一个参数中的值#的转义序列映射到 readline 特定函数 "\eOd": 向后移动一个单词 "\eOc": 向前移动一个单词

# 适用于 Linux 控制台 "\e[1~":" 移动到行首 "\e[4~":" 移动到行尾 "\e[5~":" 历史记录开头 "\e[6~":" 历史记录末尾 "\e[3~":" 删除字符 "\e[2~":" 引用插入

# 适用于 xterm 终端 "\eOH": 移动到行首 "\eOF": 移动到行尾
# 适用于 Konsole 终端 "\e[H": 移动到行首 "\e[F": 移动到行尾

# 结束 /etc/inputrc 文件
EOF
```

9.9. 创建 /etc/shells 文件

shells 文件包含系统上所有登录 shell 的列表。应用程序通过该文件判断某个 shell 是否有效。每个 shell 应独占一行，内容为该 shell 在目录结构根目录 (/) 下的相对路径。

例如，chsh 程序会参考此文件来判断非特权用户能否为自己账户更改登录 shell。若命令名称未列在文件中，用户将被拒绝更改 shell 的权限。

某些应用程序（如 GDM）若找不到/etc/shells 文件就无法显示用户头像浏览器，而传统 FTP 守护程序也会拒绝那些 shell 未列在此文件中的用户访问，因此创建该文件是必需的。

```
cat > /etc/shells << "EOF"
# 以下是/etc/shells 文件内容开始
```

```
/bin/sh
/bin/bash
```

```
# /etc/shells 文件结束
EOF
```

第 10 章 使 LFS 系统可启动

10.1. 简介

现在是时候让 LFS 系统可启动了。本章将讨论创建/etc/fstab 文件、为新 LFS 系统构建内核以及安装 GRUB 引导加载程序，以便在启动时可以选择启动 LFS 系统。

10.2. 创建/etc/fstab 文件 /etc/fstab 文件被某些程序用来确定默认情况下文件系统应该挂载到何处，

顺序，并且在挂载前必须检查（是否存在完整性错误）。创建一个新的文件系统表如下：

```
cat > /etc/fstab << "EOF"
# 开始 /etc/fstab 文件

# 文件系统 挂载点 类型 选项 dump fsck #
                                         顺序

/dev/<xxx> /          默认值           1      1
/dev/<yyy> swap swap      pri=1           0      0
proc /proc proc nosuid,noexec,nodev 0 0 sysfs /sys sysfs nosuid,noexec,nodev 0 0 devpts
/dev/pts

tmpfs /run           devpts gid=5, mode=620   0      0
tmpfs /tmp           tmpfs defaults        0      0
devtmpfs /dev devtmpfs mode=0755, nosuid 0 0 tmpfs /dev/shm
tmpfs /dev/shm       tmpfs nosuid, nodev    0      0
cgroup2 /sys/fs/cgroup cgroup2 nosuid, noexec, nodev 0 0

# /etc/fstab 结束
EOF
```

将、和替换为适用于您系统的值，例如 sda2、sda5 和 ext4。有关此文件中六个字段的详细信息，请参阅 fstab(5) 手册页。

源自 MS-DOS 或 Windows 的文件系统（如 vfat、ntfs、smbfs、cifs、iso9660、udf）需要特殊选项 utf8，才能使文件名中的非 ASCII 字符被正确解析。对于非 UTF-8 区域设置，iocharset 的值应设置为与区域设置的字符集相同，并以内核可理解的方式进行调节。如果相关字符集定义（在内核配置的“文件系统 -> 本地语言支持”下可找到）已编译进内核或作为模块构建，则此方法有效。但是，如果区域设置的字符集是 UTF-8，则对应的 iocharset=utf8 选项会使文件系统区分大小写。要解决此问题，请使用特殊选项

对于使用 UTF-8 区域设置的情况，应使用 utf8 而非 iocharset=utf8。vfat 和 smbfs 文件系统还需要“codepage”选项，该值应设为所在国家 MS-DOS 系统使用的代码页编号。例如，俄语 KOI8-R 用户（ru_RU.KOI8-R）若要挂载 USB 闪存盘，需在/etc/fstab 挂载行的选项部分添加如下参数：

```
noauto, user, quiet, showexec, codepage=866, iocharset=koi8r
```

而俄语 UTF-8 用户（ru_RU.UTF-8）对应的选项片段为：

```
noauto, user, quiet, showexec, codepage=866, utf8
```

请注意，使用 iocharset 是 iso8859-1 编码的默认选项（这会使文件系统保持大小写不敏感），而 utf8 选项则告知内核使用 UTF-8 转换文件名，以便它们能在 UTF-8 区域设置中被正确解析。

在内核配置过程中，也可以为某些文件系统指定默认的代码页和 I/O 字符集参数。相关配置项包括“默认 NLS 选项”(CONFIG_NLS_DEFAULT)、“默认远程 NLS 选项”(CONFIG_SMB_NLS_DEFAULT)、“FAT 默认代码页”(CONFIG_FAT_DEFAULT_CODEPAGE)以及“FAT 默认 I/O 字符集”(CONFIG_FAT_DEFAULT_IOCTLSET)。需要注意的是，NTFS 文件系统在内核编译时无法指定这些参数设置。

对于某些类型的硬盘，可以通过设置使 ext3 文件系统在断电情况下保持数据可靠性。具体实现方式是在/etc/fstab 文件中对应的挂载项添加

barrier=1 挂载选项。要检测磁盘驱动器是否支持此功能，可对目标磁盘执行 hdparm 命令。例如执行以下检测命令：

```
hdparm -I /dev/sda | grep NCQ
```

返回非空输出时，表示支持该选项。

注意：基于逻辑卷管理（LVM）的分区无法使用屏障选项。

10.3. Linux-6.13.4 Linux 软件包包含 Linux 内核。

预计构建时间: 0.4 - 32 SBU (通常约 2.5 SBU)

所需磁盘空间: 所需空间: 1.7 - 14 GB (通常约 2.3 GB)

10.3.1. 内核安装步骤 构建内核包含几个步骤——配置、编译和安装。关于本书所述内核配置方法之外的其他方法，请阅读内核源码树中的 README 文件。

重要提示

首次构建 Linux 内核是 LFS 中最具挑战性的任务之一。其成功与否取决于目标系统的具体硬件配置和您的特定需求。内核提供了近 12,000 个可配置项，但大多数计算机只需配置其中约三分之一。LFS 编辑团队建议不熟悉此过程的用户严格遵循以下步骤。当前目标是在系统重启后（参见第 11.3 节“系统重启”）能够通过命令行登录，此时暂不考虑优化和定制化需求。

关于内核配置的通用信息可参考 <https://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt>。更多内核配置与构建的参考资料详见

<https://anduin.linuxfromscratch.org/LFS/kernel-nutshell/>。这些参考资料虽略显陈旧，但仍能提供合理的过程概览。

若尝试所有方法仍无法解决，您可以在 lfs-support 邮件列表寻求帮助。请注意需先订阅该列表以避免垃圾邮件。

运行以下命令为编译做准备：

```
make mrproper
```

这能确保内核源码树绝对干净。内核开发团队建议在每次编译内核前都执行此命令。不要指望解压后的源码树本身就是干净的。

有多种方式可以配置内核选项。通常通过菜单驱动的界面来完成，例如：

```
make menuconfig
```

可选 make 环境变量的含义：

```
LANG=<host_LANG_value> LC_ALL=
```

这将把区域设置设为主机使用的值。在 UTF-8 Linux 文本控制台上，这对于 menuconfig 的 ncurses 界面正确绘制线条可能是必需的。如果使用此设置，请确保将替换为主机\$LANG 变量的值。您也可以改用主机的\$LC_ALL 或 \$LC_CTYPE 值。make menuconfig 这会启动一个基于 ncurses 的菜单驱动界面。对于其他（图形）界面，请输入 make help。

注意

配置内核的一个良好起点是运行 make defconfig。这将根据您当前的系统架构，将基本配置设置为一个良好的状态。

请确保启用/禁用/设置以下功能，否则系统可能无法正常工作或根本无法启动：

常规设置 ---> [] 将内核编译警告视为错误 [WERROR] CPU/任务时间和统计核算 --->

[*] 压力阻塞信息追踪 [PSI] [] 需要启动参数来启用压力阻塞信息追踪

[PSI_DEFAULT_DISABLED] < > 通过/sys/kernel/kheaders.tar.xz 启用内核头文件 [IKHEADERS] [*] 控制组支持 ---> [CGROUPS] [*] 内存控制器 [MEMCG] [] 配置标准内核功能（专家用户） ---> [EXPERT]

处理器类型与特性 ---> [*] 构建可重定位内核

[*] 随机化内核镜像地址 (KASLR) [RELOCATABLE]

[RANDOMIZE_BASE]

通用架构相关选项 ---> [*] 栈保护缓冲区溢出检测 [STACKPROTECTOR] [*] 强栈保护 [STACKPROTECTOR_STRONG]

设备驱动 ---> 通用驱动选项 --->

[] 支持 uevent 助手 [UEVENT_HELPER]

[*] 维护一个挂载在 /dev 的 devtmpfs 文件系统 [DEVTMPFS]

[*] 在内核挂载 rootfs 后，自动挂载 devtmpfs 到 /dev

... [DEVTMPFS_MOUNT]

固件驱动 ---> [*] 将 VGA/VBE/EFI 帧缓冲标记为通用系统帧缓冲 [SYSFB_SIMPLEFB] 图形支持 ---> <*> 直接渲染管理器（支持 XFree86 4.1.0 及以上版本的 DRI） --->

... [DRM]

[*] 内核恐慌时显示用户友好提示信息

... [DRM_PANIC]

(kmsg) 恐慌屏幕格式化器 [DRM_PANIC_SCREEN] 支持的 DRM 客户端 ---> [*] 为您的模式设置驱动程序启用传统 fbdev 支持

... [DRM_FBDEV_EMULATION]

<*> 简单帧缓冲驱动程序 [DRM_SIMPLEDRM] 控制台显示驱动程序支持 ---> [*] 帧缓冲控制台支持 [FRAMEBUFFER_CONSOLE]

若构建 64 位系统，请启用以下附加功能。若使用 menuconfig 配置工具，请按以下顺序启用：先启用 CONFIG_PCI_MSI，再启用 CONFIG_IRQ_REMAP，最后启用 CONFIG_X86_X2APIC（因为选项仅在其依赖项被选中后才会显示）。

处理器类型与特性 ---> [*] 支持 x2apic

[X86_X2APIC]

设备驱动 ---> [*] PCI 支持 --->

[PCI]

[*] 消息信号中断 (MSI 和 MSI-X) [PCI_MSI] [*] IOMMU 硬件支持 ---> [IOMMU_SUPPORT] [*] 支持中断重映射 [IRQ_REMAP]

如果您正在构建运行于内存超过 4GB 硬件的 32 位系统，请调整配置以使内核能够使用高达 64GB 的物理内存：

处理器类型与特性 ---> 高端内存支持 --->

(X) 64GB

[大内存 64G]

如果 LFS 系统的分区位于 NVME 固态硬盘中（即分区的设备节点为`/dev/nvme*`而非`/dev/sd*`），需启用 NVME 支持，否则 LFS 系统将无法启动：

```
设备驱动程序 -->
  NVME 支持 --> <*> NVM Express 块设备
                                         [BLK_DEV_NVME]
```

根据系统需求，可能还需要其他配置选项。有关 BLFS 软件包所需选项的完整列表，请参阅《BLFS 内核设置索引》。

注意

若宿主机采用 UEFI 固件且希望以 UEFI 方式启动 LFS 系统，即使计划使用宿主发行版的 UEFI 引导加载程序，也应参照 BLFS 页面调整部分内核配置。

上述配置项的设计原理：

`内核镜像地址随机化 (KASLR)`

为内核映像启用地址空间布局随机化(ASLR)，以缓解基于内核中敏感数据或代码固定地址的某些攻击。

`将内核编译警告视为错误`

若编译器及/或配置与内核开发者所用环境不同，可能导致构建失败。

`通过/sys/kernel/kheaders.tar.xz 启用内核头文件`

这需要 cpio 来构建内核。LFS 系统默认不安装 cpio。

`配置标准内核功能 (专家用户)`

这将使某些选项出现在配置界面中，但修改这些选项可能存在风险。

除非你清楚自己在做什么，否则不要使用此功能。

`强栈保护器`

为内核启用 SSP。我们已通过配置 GCC 的`--enable-default-ssp` 参数为整个用户空间启用了该功能，但内核不会使用 GCC 的 SSP 默认设置。此处我们显式启用该功能。

`uevent 助手支持`

启用此选项可能会在使用 Udev 时干扰设备管理。

`维护 devtmpfs 文件系统`

这将创建由内核自动生成的设备节点，即使 Udev 未运行。随后 Udev 会在此基础上运行，管理权限并添加符号链接。该配置项对所有 Udev 用户都是必需的。

`在/dev 目录自动挂载 devtmpfs`

该选项会在启动 init 之前切换到根文件系统时，将内核视角的设备挂载到`/dev` 目录。

`在内核恐慌发生时显示用户友好信息`

该功能使得当内核恐慌发生且运行的 DRM 驱动程序支持时，内核能正确显示提示信息。若无此功能，诊断恐慌将更为困难：若未运行 DRM 驱动，系统会退回到仅能显示 24 行的 VGA 控制台，关键内核信息常被冲刷消失；若 DRM 驱动正在运行，恐慌发生时显示器输出往往完全混乱。截至 Linux-6.12 版本，主流 GPU 型号的专用驱动均未真正支持此功能，但“简易帧缓冲驱动”可实现在专用 GPU 驱动加载前，通过 VESA (或 EFI) 帧缓冲运行。若专用 GPU 驱动以模块形式构建（而非集成至内核映像）且未使用 initramfs，该功能在根文件系统挂载前即可正常工作，已足以为大多数故障场景提供诊断信息。

LFS 配置错误导致内核恐慌（例如第 10.4 节“使用 GRUB 设置启动流程”中错误的 root=参数设置）

恐慌屏幕格式化器

设置此 kmsg 选项可确保内核恐慌发生时显示最后的内核消息行。默认值

user 会使内核仅显示对用户友好的恐慌信息，不利于诊断。第三个选项 qr_code 会让内核将最后的内核消息行压缩成二维码显示。二维码比纯文本能容纳更多消息行，且可通过外部设备（如智能手机）解码，但需要 LFS 未提供的 Rust 编译器支持。

将 VGA/VBE/EFI 帧缓冲标记为通用系统帧缓冲 和 简单帧缓冲驱动

这些选项允许将 VESA 帧缓冲（或通过 UEFI 启动 LFS 系统时的 EFI 帧缓冲）作为 DRM 设备使用。VESA 帧缓冲将由 GRUB 设置（或 EFI 帧缓冲由 UEFI 固件设置），因此在加载特定 GPU 的 DRM 驱动之前，DRM 紧急处理程序就能正常工作。

为您的模式设置驱动启用传统 fbdev 支持 和 帧缓冲控制台支持

这些选项对于在由 DRI（直接渲染基础设施）驱动驱动的 GPU 上显示 Linux 控制台是必需的。由于启用了 CONFIG_DRM（直接渲染管理器），我们也应该启用这两个选项，否则一旦加载 DRI 驱动，屏幕就会显示空白。

支持 x2apic

支持在 x2APIC 模式下运行 64 位 x86 处理器的中断控制器。64 位 x86 系统的固件可能已启用 x2APIC，若内核未启用此选项且固件已开启 x2APIC，系统启动时将发生崩溃。当固件禁用 x2APIC 时，此选项无实际作用但也不会造成损害。

对于提供大多数信息而言已经足够。在某些情况下，使用 make oldconfig 可能更为合适。更多信息请参阅 README 文档。如需跳过内核配置，可将主机系统中的内核配置文件.config 复制到解压后的 linux-6.13.4 目录（假设该文件可用）。但我们不建议采用此方式。通常更好的做法是浏览所有配置菜单并从头创建内核配置。

编译内核映像及模块：

make

若使用内核模块，可能需要在/etc/modprobe.d 中进行模块配置。有关模块及内核配置的信息，请参阅第 9.3 节“设备与模块处理概述”以及 linux-6.13.4/Documentation 目录中的内核文档。此外，modprobe.d(5) 手册页也可能有所帮助。

除非在内核配置中禁用了模块支持，否则请使用以下命令安装模块：

make modules_install

内核编译完成后，需要执行额外步骤来完成安装。部分文件需复制到/boot 目录下。

注意

若您决定为 LFS 系统使用独立的/boot 分区（可能与宿主发行版共享/boot 分区），下文所述待复制的文件应存放于此。最简便的方式是先在/etc/fstab 中创建/条目（详见前一章节说明），随后在 chroot 环境中以 root 用户身份执行以下命令：

挂载 /boot

命令中省略了设备节点路径，因为 mount 可以从 /etc/fstab 中读取该信息。

内核映像的路径可能因使用的平台而异。下方文件名可根据个人偏好修改，但文件名主体应保持为 vmlinuz，以确保与下一节描述的启动过程自动配置兼容。以下命令假设为 x86 架构：

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-6.13.4-1fs-12.3
```

`System.map` 是内核的符号文件。它映射了内核 API 中每个函数的入口点，以及运行中内核数据结构的地址。在排查内核问题时，该文件可作为重要资源。执行以下命令安装映射文件：

```
cp -iv System.map /boot/System.map-6.13.4
```

上述通过 `make menuconfig` 步骤生成的内核配置文件 `.config` 包含了刚刚编译内核的所有配置选项。建议保留此文件以供日后参考：

```
cp -iv .config /boot/config-6.13.4
```

安装 Linux 内核的文档：

```
cp -r Documentation -T /usr/share/doc/linux-6.13.4
```

需要特别注意的是，内核源代码目录中的文件并不属于 `root` 用户所有。当以 `root` 用户身份解压软件包时（例如我们在 `chroot` 环境中所做的操作），这些文件会保留打包者在原始计算机上设置的用户 ID 和组 ID。对于其他大多数需要安装的软件包而言，这通常不会造成问题，因为它们的源代码树在安装完成后就会被移除。然而，Linux 内核源代码树往往会被长期保留。正因如此，打包者使用的用户 ID 有可能与系统中后续创建的某个用户 ID 重合，这将导致该用户意外获得内核源代码的写入权限。

注意

在许多情况下，为了兼容后续在 BLFS 中安装的软件包，需要更新内核配置。与其他软件包不同，安装新编译的内核后，无需删除内核源代码树。

若计划保留内核源代码树，请对 `linux-6.13.4` 目录执行 `chown -R 0:0` 命令，以确保所有文件归属于 `root` 用户。

如果您是从保留的内核源码树更新配置并重新构建内核，通常不应运行 `make mrproper` 命令。该命令会清除 `.config` 文件及所有相关配置。

- o 前一次构建生成的文件。虽然从`/boot` 中的备份恢复 `.config` 很容易，但清除所有 `.o` 文件仍然是种浪费：对于简单的配置变更，通常只需（重新）构建少数几个 `.o` 文件，若未清除其他 `.o` 文件，内核构建系统会正确跳过它们。

另一方面，如果您升级了 `GCC`，应当运行 `make clean` 命令清除之前构建生成的所有 `.o` 文件，否则新构建可能会失败。

警告

部分内核文档建议创建一个从`/usr/src/linux` 指向内核源码目录的符号链接。这一操作仅适用于 2.6 系列之前的内核版本，在 LFS 系统中切勿创建此类链接，因为它可能导致基础 LFS 系统完成后需要构建的软件包出现问题。

10.3.2. 配置 Linux 模块加载顺序

大多数情况下 Linux 模块会自动加载，但有时需要特定引导。加载模块的程序（modprobe 或 insmod）会使用 `/etc/modprobe.d/usb.conf` 文件实现这一目的。需要创建该文件以确保当 USB 驱动（ehci_hcd、ohci_hcd 和 uhci_hcd）以模块形式构建时，它们能按正确顺序加载——为避免系统启动时输出警告信息，ehci_hcd 必须在 ohci_hcd 和 uhci_hcd 之前加载。

通过以下命令创建新文件 `/etc/modprobe.d/usb.conf`：

```
install -v -m755 -d /etc/modprobe.d
cat > /etc/modprobe.d/usb.conf << "EOF"
# 开始 /etc/modprobe.d/usb.conf

安装 ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true 安装 uhci_hcd /sbin/modprobe
ehci_hcd ; /sbin/modprobe -i uhci_hcd ; true

# /etc/modprobe.d/usb.conf 文件结束
EOF
```

10.3.3. Linux 已安装文件内容：

config-6.13.4, vmlinuz-6.13.4-lfs-12.3, 以及 System.map-6.13.4
已安装的目录：
/lib/modules, /usr/share/doc/linux-6.13.4

简要描述

config-6.13.4 包含内核的所有配置选项

vmlinuz-6.13.4-lfs-12.3 Linux 系统的引擎。当启动计算机时，内核是操作系统最先加载的部分。它会检测并初始化计算机硬件的所有组件，然后将这些组件以文件树的形式提供给软件使用，并将单个 CPU 转变为能够同时运行众多程序的多任务处理机器。

System.map-6.13.4 地址与符号的对应列表；它记录了内核中所有函数和数据结构的入口点及地址映射关系

10.4. 使用 GRUB 设置启动流程注意事项

若您的系统支持 UEFI 且希望以 UEFI 方式启动 LFS，应跳过本页说明，但仍需学习本页提供的 grub.cfg 语法规则及文件中指定分区的方法，并参照 BLFS 手册页的指导配置支持 UEFI 的 GRUB。

10.4.1. 重要警告说明

错误配置 GRUB 可能导致系统无法启动，且需依赖光盘/USB 启动盘等备用引导设备才能恢复。本节内容并非 LFS 系统启动的必要步骤。您也可以选择仅修改现有引导加载程序（如 Grub-Legacy、GRUB2 或 LILO）。

确保准备好一张应急启动盘，以便在计算机无法使用（无法启动）时进行“救援”。如果还没有启动设备，可以创建一个。为了使以下步骤生效，需要提前参考 BLFS 并安装 libisoburn 软件包中的 xorriso 工具。

```
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 命名规范

GRUB 采用独特的驱动器与分区命名结构，格式为(hdn,m)，其中 n 代表硬盘序号，m 代表分区编号。硬盘序号从 0 开始计数，而普通分区编号从 1 开始（扩展分区则从 5 开始计数）。请注意这与早期版本不同——旧版本中两个数字都从 0 开始计数。例如，分区 sda1 在 GRUB 中表示为(hd0,1)，sdb3 则对应(hd1,3)。与 Linux 系统不同，GRUB 不将 CD-ROM 驱动器视为硬盘设备。举例来说，若在 hdb 使用光盘驱动器，第二块硬盘安装在

hdc，那么第二个硬盘驱动器仍将显示为(hd1)。

10.4.3. 配置 GRUB 设置

GRUB 通过向硬盘的第一个物理磁道写入数据来工作。该区域不属于任何文件系统。该区域的程序会访问启动分区中的 GRUB 模块，默认位置是/boot/grub/。

启动分区的位置由用户选择，这将影响配置。建议为启动信息单独划分一个小分区（建议大小为 200MB）。这样无论是 LFS 还是其他商业发行版的每次构建，都可以访问相同的启动文件，并且可以从任何已启动的系统进行访问。如果选择这样做，需要挂载这个独立分区，将当前/boot 目录中的所有文件（例如前一节中刚构建的 Linux 内核）移动到新分区。然后需要卸载该分区，并重新挂载为/boot。如果执行此操作，请确保更新/etc/fstab。

将/boot 保留在当前 LFS 分区上也可以工作，但配置多个系统会更加困难。

根据上述信息，确定根分区（若使用独立启动分区则为启动分区）的正确标识符。以下示例假设根分区（或独立启动分区）为 sda2。

将 GRUB 文件安装到/boot/grub 目录并设置启动引导：

警告

以下命令将覆盖当前的引导加载程序。如果不希望执行此操作（例如正在使用第三方引导管理器来管理主引导记录 MBR），请不要运行该命令。

```
grub-install /dev/sda
```

注意

若系统通过 UEFI 方式启动，grub-install 会尝试安装 x86_64-efi 目标文件，但这些文件在第 8 章中尚未安装。遇到这种情况时，请在上述命令后添加--target i386-pc 参数。

10.4.4. 创建 GRUB 配置文件 生成/boot/grub/grub.cfg：

```
cat > /boot/grub/grub.cfg << "EOF"
# 开始 /boot/grub/grub.cfg 配置
set default=0
set timeout=5

insmod part_gpt
insmod ext2
设置 root=(hd0, 2)
设置 gfxpayload=1024x768x32

菜单项 "GNU/Linux, Linux 6.13.4-lfs-12.3" {
    linux /boot/vmlinuz-6.13.4-lfs-12.3 root=/dev/sda2 ro
}
EOF
```

insmod 命令加载了名为 part_gpt 和 ext2 的 GRUB 模块。尽管名称如此，ext2 实际上支持 ext2、ext3 和 ext4 文件系统。grub-install 命令已将部分模块嵌入主 GRUB 镜像（安装到 MBR 或 GRUB BIOS 分区），以便在不存在“先有鸡还是先有蛋”问题的情况下访问其他模块（位于/boot/grub/i386-pc 目录中）。因此在典型配置下，这两个模块已被预先嵌入，上述两条 insmod 命令不会产生实际效果。不过它们也不会造成任何损害，且在某些特殊配置中可能会需要这些命令。

set gfxpayload=1024x768x32 命令设置要传递给内核的 VESA 帧缓冲分辨率和色深。这是内核 SimpleDRM 驱动程序使用 VESA 帧缓冲的必要设置。您可以根据显示器适配情况调整不同的分辨率或色深值。

注意

从 GRUB 的角度看，内核文件是相对于所用分区的路径。如果您使用了独立的/boot 分区，请删除上述 linux 行中的/boot 路径。同时需要修改 set root 行使其指向启动分区。

注意

当您添加或移除某些磁盘（包括 U 盘等可移动设备）时，GRUB 对分区的标识符可能会发生变化。这种变化可能导致启动失败，因为 grub.cfg 文件中引用了“旧”的标识符。若要避免此类问题，您可以使用分区 UUID 和文件系统 UUID 来代替 GRUB 标识符指定设备。运行 lsblk -o UUID,PARTUUID,PATH,MOUNTPOINT 命令可显示文件系统（UUID 列）和分区（PARTUUID 列）的 UUID 值。随后将 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 所在分区的 UUID> .

请注意，分区的 UUID 与该分区内文件系统的 UUID 完全不同。某些网络资源可能会建议你使用 root=UUID=<文件系统 UUID> 而非

root=PARTUUID=<分区 UUID>，但这样做将需要 initramfs，这已超出 LFS 的范围。

/dev 目录下分区对应的设备节点名称也可能发生变化（虽然比 GRUB 标识符变更的概率更低）。你还可以在/

etc/fstab 文件中~~以防设备节点名称变更导致潜在的启动故障~~ UUID>

GRUB 是一款极其强大的程序，它为从各类设备、操作系统及分区类型启动提供了海量选项。此外还支持图形启动画面、音效播放、鼠标输入等多种自定义功能，这些选项的详细说明已超出本入门指南的范畴。

注意

存在一个名为 grub-mkconfig 的命令可自动生成配置文件。该命令会调用 /etc/grub.d/ 目录下的脚本集，但会覆盖所有手动定制内容。这些脚本主要针对非源码发行版设计，不推荐在 LFS 中使用。若您安装了商业 Linux 发行版，该程序很可能会自动执行，请务必提前备份 grub.cfg 文件。

第 11 章 完结篇

11.1. 圆满竣工 恭喜！全新的 LFS 系统已安装完成！祝愿您这套精心打造的自定义 Linux 系统运行顺利。

建议创建/etc/lfs-release 文件。该文件能帮助您（以及在我们需要为您提供技术支持时）快速识别当前系统安装的 LFS 版本。通过以下命令创建该文件：

```
echo 12.3 > /etc/lfs-release
```

系统后续安装的软件包（无论是二进制形式还是源码编译安装）可能会用到两个描述已安装系统的文件。

第一条信息展示了新系统与 Linux 标准规范(LSB)的兼容状态。创建该文件请执行：

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="12.3"
DISTRIB_CODENAME=""
DISTRIB_DESCRIPTION="Linux From Scratch" EOF
```

第二条信息包含大致相同的内容，主要供 systemd 和部分图形桌面环境使用。创建该文件请执行：

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="12.3"
ID=lfs
PRETTY_NAME="Linux From Scratch 12.3"
VERSION_CODENAME=<在此填写你的名称>
HOME_URL="https://www.linuxfromscratch.org/lfs/"
RELEASE_TYPE="stable"
EOF
```

请务必自定义'DISTRIB_CODENAME'和'VERSION_CODENAME'字段，使系统具有你的个人特色。

11.2. 用户统计

既然已完成本书的实践，你希望被统计为 LFS 用户吗？访问 <https://www.linuxfromscratch.org/cgi-bin/lfscounter.php>，输入你的姓名和使用的首个 LFS 版本号即可完成注册。

现在让我们重启进入 LFS 系统。

11.3. 重启系统 现在所有软件都已安装完毕，是时候重启计算机了。不过仍有几项

需要检查的事项。以下是一些建议：

- 如果内核驱动程序需要某些固件文件才能正常工作，请安装所需的固件。
- 确保为 root 用户设置了密码。

- 此时也适合检查以下配置文件。

- /etc/bashrc
- /etc/dircolors
- /etc/fstab
- /etc/hosts
- /etc/inputrc
- /etc/profile
- /etc/resolv.conf
- /etc/vimrc
- /root/.bash_profile
- /root/.bashrc
- /etc/sysconfig/ifconfig.eth0

既然已经说到这，现在让我们首次启动闪亮的新 LFS 系统吧！首先退出 chroot 环境：

```
logout
```

然后卸载虚拟文件系统：

```
卸载 -v $LFS/dev/pts
挂载点 -q $LFS/dev/shm && 卸载 -v $LFS/dev/shm
卸载 -v $LFS/dev
卸载 -v $LFS/run
卸载 -v $LFS/proc
卸载 -v $LFS/sys
```

如果创建了多个分区，在卸载主分区前需先卸载其他分区，操作如下：

```
卸载 $LFS/home 目录:
卸载 $LFS 目录:
```

卸载 LFS 文件系统本身：

```
umount -v $LFS
```

现在，重启系统。

若之前已按照说明设置好 GRUB 引导程序，系统将自动选择启动 LFS 12.3 的菜单项。

重启完成后，LFS 系统即可投入使用。此时您将看到简单的"login: "提示符。接下来您可以参考 BLFS 手册，根据需求安装更多软件。

若重启失败，请进行故障排查。解决初始启动问题的提示可参阅：

<https://www.linuxfromscratch.org/lfs/troubleshooting.html>

11.4. 附加资源 感谢您阅读这本 LFS 书籍。我们希望您觉得本书有所帮助，并对系统构建过程有了更深入的了解。

现在 LFS 系统已安装完成，您可能在想"接下来该做什么？"为此，我们为您整理了一份资源清单。

· 维护

所有软件都会定期报告错误和安全通知。由于 LFS 系统是从源代码编译的，因此需要您自行跟进这些报告。以下列出了一些跟踪此类报告的在线资源：

- LFS 安全公告

这是在 LFS 书籍发布后发现的安全漏洞列表。

- 开源安全邮件列表

这是一个用于讨论开源社区中安全漏洞、概念和实践的邮件列表。

- LFS 技巧集

LFS 技巧集是由 LFS 社区志愿者提交的教育文档合集，可通过

<https://www.linuxfromscratch.org/hints/downloads/files/> 获取。

- 邮件列表

如果您需要帮助、希望了解最新动态、或想为项目贡献力量，可以订阅多个 LFS 邮件列表。更多详情请参阅第 1 章-邮件列表部分。

- Linux 文档计划

Linux 文档计划(TLDP)旨在协作解决所有 Linux 文档相关事宜。该计划收录了大量 HOWTO 指南、使用手册和 man 页面，官网地址为 <https://tldp.org/>。

11.5. LFS 完成后的起步指南

11.5.1. 确定后续方向

既然 LFS 已经完成且拥有了可启动系统，接下来该做什么？下一步是规划如何使用它。

通常有两个主要方向需要考虑：工作站或服务器。实际上这两个类别并非互斥。每个类别所需的应用程序可以组合在同一个系统中，但让我们先分别探讨它们。

服务器是较为简单的类别。通常包含网页服务器（如 Apache HTTP Server）和数据库服务器（如 MariaDB）。当然也可能需要其他服务。嵌入式操作系统也属于这一类别。

另一方面，工作站系统则复杂得多。它通常需要基于基础图形环境的图形用户界面（如 LXDE、XFCE、KDE 或 Gnome），以及若干图形应用程序（例如 Firefox 网页浏览器、Thunderbird 邮件客户端或 LibreOffice 办公套件）。这些应用程序需要数百个（具体数量取决于功能需求）额外的支持软件包和依赖库。

除上述内容外，各类系统还需要一套系统管理应用程序。这些应用程序均收录于 BLFS 手册中。并非所有软件包都适用于每种环境，例如 dhcpcd 通常不适用于服务器，而 wireless_tools 通常仅对笔记本电脑系统有用。

11.5.2. 基础 LFS 环境下的工作

初次启动 LFS 系统时，虽然已具备构建额外软件包的所有基础工具，但用户操作环境仍非常简陋。可通过以下几种方式进行改善：

环境相当简陋。有几种方法可以改善这种情况：

11.5.2.1. 在 chroot 环境下使用 LFS 宿主系统工作 该方法提供完整的图形环境，支持功能齐全的浏览器和复制/粘贴操作。通过此方法可利用宿主系统的应用程序（如 wget）将软件包源码下载到 chroot 环境中可访问的位置。

为了在 chroot 环境中正确构建软件包，还需确保虚拟文件系统已挂载（若尚未挂载）。可在宿主系统创建如下脚本实现：

```
cat > ~/mount-virt.sh << "EOF"
#!/bin/bash

function mountbind
{ 如果 ! mountpoint $LFS/$1 >/dev/null; 那么
```

```
$SUDO mount --bind /$1 $LFS/$1 回显 $LFS/$1  
已挂载 否则 回显 $LFS/$1 已经挂载 结束 }
```

函数 挂载类型

```
{ 如果 ! mountpoint $LFS/$1 >/dev/null; 那么  
$SUDO mount -t $2 $3 $4 $5 $LFS/$1 echo $LFS/$1 已  
挂载 else echo $LFS/$1 已挂载 fi }
```

```
if [ $EUID -ne 0 ]; then  
SUDO=sudo else SUDO="" fi
```

```
if [ x$LFS == x ]; then echo "未  
设置 LFS 变量" 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
```

```
安装 -v -d -m 1777 $LFS$(realpath /dev/shm) 否则挂载类型 dev/shm
tmpfs tmpfs -o nosuid,nodev 结束
```

```
#挂载绑定 usr/src
#挂载绑定 boot
#挂载绑定 home
EOF
```

请注意，脚本中的最后三条命令已被注释掉。如果这些目录在主机系统上作为独立分区挂载，并且在启动完成的 LFS/BLFS 系统时需要挂载，这些命令会很有用。

该脚本可以通过`bash ~/mount-virt.sh`命令以普通用户（推荐）或 root 身份运行。若以普通用户身份运行，则宿主机系统需具备 sudo 权限。

脚本指出的另一个问题是下载包文件的存储位置。该位置可任意选择，可以是普通用户的主目录（如`~/sources`），也可以是全局路径（如`/usr/src`）。我们建议不要将 BLFS 源码与 LFS 源码混放在（从 chroot 环境视角的）`~/sources` 目录中。无论选择何种路径，这些软件包必须能在 chroot 环境中访问。

最后介绍一个简化进入 chroot 环境流程的便捷功能。可通过在宿主机系统的用户`~/.bashrc`文件中添加别名实现：

```
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'
```

由于引号和反斜杠字符的多层嵌套，这个别名设置起来有点棘手。它必须全部写在一行内。上面的命令为了展示方便被分成了两行。

11.5.2.2. 通过 SSH 远程工作 这种方法同样提供完整的图形环境，但首先需要在 LFS 系统（通常在 chroot 环境下）安装 sshd 服务。该方法还需要另一台计算机配合使用。其优势在于避免了 chroot 环境的复杂性，操作更为简单。此外，所有额外软件包都将使用您构建的 LFS 内核，同时仍为软件包安装提供完整的系统环境。

您可以使用 scp 命令将需要编译的软件包源码上传至 LFS 系统。若希望直接在 LFS 系统上下载源码，则需在 chroot 环境中先安装 libtasn1、p11-kit、make-ca 和 wget（或在启动 LFS 系统后使用 scp 上传这些软件的源码）。

11.5.2.3. 在 LFS 命令行环境下工作 此方法需要在 chroot 环境中安装 libtasn1、p11-kit、make-ca、wget、gpm 以及 links（或 lynx）软件包，然后重启进入新构建的 LFS 系统。此时默认系统会提供六个虚拟控制台。切换控制台只需使用 Alt+Fx 组合键（其中 Fx 代表 F1 至 F6 功能键）。Alt+
← 与 Alt+ → 组合键同样可以

实现控制台切换。

此时，您可以登录两个不同的虚拟控制台，在一个控制台中运行 links 或 lynx 浏览器，在另一个控制台中运行 bash。随后，GPM（通用鼠标服务）允许用鼠标左键从浏览器复制命令，切换控制台，并粘贴到另一个控制台中。

注意

另外需要注意的是，虚拟控制台的切换也可以通过 X Window 实例使用 Ctrl+Alt+Fx 组合键完成，但鼠标复制操作无法在图形界面和虚拟控制台之间进行。您可以通过 Ctrl+Alt+Fx 组合键返回 X Window 显示界面，其中 Fx 通常是 F1 键，但也可能是 F7 键。

第五部分 附录

附录 A. 缩写词与术语表

ABI 应用程序二进制接口

ALFS 自动化 Linux 从头构建系统

API 应用程序编程接口

ASCII 美国信息交换标准代码 BIOS 基本输入输出系统 BLFS Linux 进阶指南 BSD 伯克利软件发行版 chroot 改变根目录
CMOS 互补金属氧化物半导体

COS 服务等级

CPU 中央处理器

CRC 循环冗余校验 CVS 并发版本控制系统 DHCP 动态主机配置协议 DNS 域名服务 EGA 增强型图形适配器

ELF 可执行与可链接格式

EOF 文件结束符

EQN 方程式

ext2 第二代扩展文件系统

ext3 第三代扩展文件系统

ext4 第四代扩展文件系统 FAQ 常见问题解答 FHS 文件系统层次结构标准

FIFO 先进先出

FQDN 完全限定域名

文件传输协议(FTP)

GB 千兆字节 GCC GNU 编译器集合 GID 组标识符 GMT 格林尼治标准时间 HTML 超文本标记语言

IDE 集成驱动电子设备

IEEE 电气与电子工程师协会

输入/输出 输入/输出

IP 互联网协议

IPC 进程间通信

IRC 互联网中继聊天

ISO 国际标准化组织

互联网服务提供商

KB 千字节

发光二极管

LFS 从零开始构建 Linux 系统 LSB Linux 标准规范

MB 兆字节 MBR 主引导记录

MD5 信息摘要算法 5

网络接口卡 (NIC)

本地语言支持 (NLS)

网络新闻传输协议 (NNTP)

原生 POSIX 线程库 (NPTL)

开放声音系统 (OSS)

预编译头文件 (PCH)

Perl 兼容正则表达式 (PCRE)

进程标识符 (PID)

伪终端 (PTY)

服务质量 (QOS)

随机存取存储器 (RAM)

RPC 远程过程调用

RTC 实时时钟

SBU 标准构建单位

SCO Santa Cruz 操作系统公司

SHA1 安全散列算法 1

TLDL Linux 文档项目

TFTP 简单文件传输协议

TLS 线程本地存储

UID 用户标识符

umask 用户文件创建掩码

USB 通用串行总线

协调世界时

通用唯一识别码 (UUID)

虚拟控制台 (VC)

视频图形阵列 (VGA)

虚拟终端 (~~虚拟~~终端)

附录 B. 鸣谢

我们衷心感谢以下个人及组织对《Linux From Scratch》项目的贡献。

- 杰拉德·比克曼斯 —— LFS 创始人
- 布鲁斯·杜布斯 —— LFS 执行编辑
- 吉姆·吉福德 – CLFS 项目联合负责人
- 皮埃尔·拉巴斯蒂 – BLFS 编辑及 ALFS 负责人
- DJ·卢卡斯 – LFS 与 BLFS 编辑
- 肯·莫法特 – BLFS 编辑
- 无数来自 LFS 和 BLFS 邮件列表的贡献者通过提供建议、测试手册、提交错误报告、操作指南以及分享各类软件包安装经验，使本书得以问世。

翻译人员

- Manuel Canales Esparcia – 西班牙语 LFS 翻译项目
- Johan Lenglet – 2008 年前法语 LFS 翻译项目
- 让-菲利普·门古阿尔 – 法语 LFS 翻译项目 2008-2016
- 朱利安·勒皮耶 – 法语 LFS 翻译项目 2017 至今
- 安德森·利扎多 – 葡萄牙语 LFS 翻译项目历史成员
- 雅门森·埃斯平杜拉 – 葡萄牙语 LFS 翻译项目 2022 至今
- 托马斯·赖特尔巴赫 – 德语版 LFS 翻译项目

镜像站点维护者

北美镜像站点

- 斯科特·克维顿 – lfs.oregonstate.edu 镜像站
- 威廉·阿斯特尔 – ca.linuxfromscratch.org 镜像站点
- 尤金·塞勒斯 – lfs.introspeed.com 镜像站点
- 贾斯汀·尼里姆 – lfs-matrix.net 镜像站点

南美洲镜像站点

- 曼努埃尔·卡纳莱斯·埃斯帕西亚 – lfsmirror.lfs-es.info 镜像站点
- 路易斯·法尔孔 – torredehanoi.org 镜像站点

欧洲镜像站点

- 吉多·帕塞特 – nl.linuxfromscratch.org 镜像站点
- 巴斯蒂安·雅克 – lfs.pagefault.net 镜像站点
- 斯文·克兰斯霍夫 – lfs.lineo.be 镜像站点

- 猩红比利时 – lfs.scarlet.be 镜像站点
- 塞巴斯蒂安·福尔伯恩 – lfs.aliensoft.org 镜像站点
- 斯图尔特·福克斯 – lfs.dontuse.ms 镜像站点
- 拉尔夫·乌勒曼 – lfs.oss-mirror.org 镜像站点
- 安东宁·斯普林茨尔 – at.linuxfromscratch.org 镜像站点
- 弗雷德里克·达纳克林特 – se.linuxfromscratch.org 镜像站点
- 弗兰克 – lfs.linuxpourtous.com 镜像站点
- 菲利普·巴克 – lfs.cict.fr 镜像站点
- 维塔利·切卡辛 – lfs.pilgrims.ru 镜像站点
- 本杰明·海尔 – lfs.wankoo.org 镜像站点
- 安东·迈萨克 – linuxfromscratch.org.ru 镜像站点

亚洲镜像站点

- 萨提特·佩姆萨旺 – lfs.phayoune.org 镜像站点
- 静冈网络有限公司 – lfs.mirror.shizu-net.jp 镜像站点

澳大利亚镜像站点

- 杰森·安德拉德 – au.linuxfromscratch.org 镜像站点

前项目团队成员

- 克里斯汀·巴尔恰克 – LFS 手册编辑
- Archaic – LFS 技术文档作者/编辑、HLFS 项目负责人、BLFS 编辑、技巧与补丁项目维护者
- 马修·伯吉斯 – LFS 项目负责人、LFS 技术文档作者/编辑
- 内森·库尔森 – LFS 启动脚本维护者
- 蒂莫西·鲍舍
- 罗伯特·布里格斯
- 伊恩·奇尔顿
- 杰伦·库曼斯 – 网站开发人员，常见问题维护者
- 曼努埃尔·卡纳莱斯·埃斯帕西亚 – LFS/BLFS/HLFS XML 与 XSL 维护者
- 亚历克斯·赫鲁内沃德 – LFS 技术文档撰写者
- 马克·赫尔丁克
- 杰里米·亨特沃克 – LFS 技术文档作者，LFS LiveCD 维护者
- 布莱恩·卡兹班 – LFS 技术文档作者
- 马克·海默斯
- 塞思·W·克莱因 – 常见问题解答维护者
- Nicholas Leippe – 维基维护者

- 安德森·利扎多 – Wiki 维护者
- 兰迪·麦克默奇 – BLFS 项目负责人, LFS 编辑
- 丹·尼科尔森 – LFS 与 BLFS 编辑
- 亚历山大·E·帕特拉科夫 – LFS 技术文档撰写人, LFS 国际化编辑, LFS Live CD 维护者
- 西蒙·佩罗
- 斯科特·麦克弗森 – LFS NNTP 网关维护者
- 道格拉斯·R·雷诺 – Systemd 编辑
- 瑞安·奥利弗 – CLFS 项目联合负责人
- 格雷格·谢弗 – LFS 技术文档作者及下一代 64 位构建方法架构师
- 杰西·泰-腾-奎 – LFS 技术文档作者
- 詹姆斯·罗伯逊 – Bugzilla 维护者
- 图沙尔·特雷德赛 – BLFS 手册编辑、技巧与补丁项目负责人
- 杰里米·尤特利 – LFS 技术文档作者、Bugzilla 维护者、LFS 启动脚本维护者
- 扎克·温克尔斯 – LFS 技术文档作者

附录 C. 依赖关系

com> - LFS 技术文档撰稿人

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编译和安装。某些软件包甚至存在循环依赖关系，即第一个软件包依赖第二个，而第二个又反过来依赖第一个。由于这些依赖关系，LFS 中软件包的构建顺序至关重要。本附录旨在记录 LFS 中每个构建软件包的依赖关系。

对于每个构建的软件包，下方列出了三到五种依赖类型（有时多达五种）。第一种列出了编译和安装该软件包所需的其他软件包。第二种列出了在运行时使用该软件包中的程序或库时必须存在的软件包。第三种列出了除第一种列表外，运行测试套件所需的额外软件包。第四种依赖列表则包含那些需要在该软件包完成构建并安装到最终位置后，才能进行构建和安装的其他软件包。

最后的依赖项列表是 LFS 未涉及的可选软件包，但对用户可能很有用。这些软件包可能还有额外的必需或可选依赖项。对于这些依赖项，建议的做法是在完成 LFS 手册后安装它们，然后返回并重新构建 LFS 软件包。

在 BLFS 中会涉及多次重新安装的情况。

Acl

安装依赖于：Attr、Bash、Binutils、Coreutils、GCC、Gettext、Grep、M4、Make、Perl、Sed 和 Texinfo
运行时必需：Attr 和 Glibc

测试套件依赖：Automake、Diffutils、Findutils 和 Libtool

必须预先安装：Coreutils、Sed、Tar 和 Vim

可选依赖项：无

Attr

安装依赖项：Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、M4、Make、Perl、Sed 和 Texinfo
运行时必需：Glibc

测试套件依赖：Automake、Diffutils、Findutils 和 Libtool

必须预先安装：Acl、Libcap 和 Patch

可选依赖项：无

Autoconf

安装依赖项：Bash、Coreutils、Grep、M4、Make、Perl、Sed 和 Texinfo

运行时必需：Bash、Coreutils、Grep、M4、Make、Sed 和 Texinfo

测试套件依赖：Automake、Diffutils、Findutils、GCC 和 Libtool

必须在以下软件之前安装：Automake 和 Coreutils

可选依赖项：Emacs

Automake

安装依赖项：Autoconf、Bash、Coreutils、Gettext、Grep、M4、Make、Perl、Sed 和 Texinfo

运行时必需：Bash、Coreutils、Grep、M4、Sed 和 Texinfo 测试套件依赖：Binutils、Bison、Bzip2、DejaGNU、Diffutils、Expect、Findutils、Flex、GCC、Gettext、Gzip

Libtool 和 Tar

必须预先安装：Coreutils 可选依赖

项：无

Bash

安装依赖项: Bash、Binutils、Bison、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Make、Ncurses、Patch、Readline、Sed 和 Texinfo

运行时依赖项: Glibc、Ncurses 和 Readline

测试套件依赖项: Expect 和 Shadow

必须预先安装: 无

可选依赖项: Xorg

Bc

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Grep、Make 和 Readline

运行时必需: Glibc、Ncurses 和 Readline

测试套件依赖项: Gawk 必须在此之

前安装: Linux 可选依赖项: 无

Binutils

安装依赖项: Bash、Binutils、Coreutils、Diffutils、File、Flex、Gawk、GCC、Glibc、Grep、Make、Perl、Pkgconf、Sed、Texinfo、Zlib 和 Zstd

运行时依赖: Glibc、Zlib 和 Zstd

测试套件依赖: DejaGNU 和 Expect

必须预先安装: 无

可选依赖: Elfutils 和 Jansson

Bison

安装依赖: Bash、Binutils、Coreutils、Diffutils、GCC、Gettext、Glibc、Grep、M4、Make、Perl 和 Sed

运行时必需: Glibc

测试套件依赖: Diffutils、Findutils 和 Flex

必须预先安装: Kbd 和 Tar 可选依赖项:

Doxxygen

Bzip2

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Glibc、Make 和 Patch

运行时需求: Glibc

测试套件依赖项: 无

必须在此之前安装: File 和 Libelf

可选依赖项: 无

检查

安装依赖项: Gawk、GCC、Grep、Make、Sed 和 Texinfo

运行时必需: Bash 和 Gawk

测试套件依赖: 无

必须在之前安装: 无

可选依赖项: libsubunit 和 patchutils

Coreutils

安装依赖项: Autoconf、Automake、Bash、Binutils、Coreutils、GCC、Gettext、Glibc、GMP、Grep、Libcap、Make、OpenSSL、Patch、Perl、Sed 和 Texinfo

运行时依赖: Glibc

测试套件依赖: Diffutils、E2fsprogs、Findutils、Shadow 和 Util-linux

必须在以下软件之前安装: Bash、Diffutils、Findutils、Man-DB 和 Udev

可选依赖项: Expect.pm 和 IO::Tty

DejaGNU

安装依赖: Bash、Coreutils、Diffutils、Expect、GCC、Grep、Make、Sed 和 Texinfo

运行时必需: Expect 和 Bash

测试套件依赖: 无

必须在以下软件之前安装: 无

可选依赖项: 无

Diffutils

安装依赖项: Bash、Binutils、Coreutils、Gawk、GCC、Gettext、Glibc、Grep、Make、Sed 和 Texinfo

运行时必需: Glibc

测试套件依赖: Perl

必须在此之前安装: 无

可选依赖项: 无

E2fsprogs

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Gzip、Make、Pkgconf、Sed、Texinfo 以及 Util-linux

运行时依赖: Glibc 和 Util-linux

测试套件依赖: Procps-ng 和 Psmisc

必须在此之前安装: 无

可选依赖项: 无

Expat

安装依赖项: Bash、Binutils、Coreutils、Gawk、GCC、Glibc、Grep、Make 和 Sed

运行时依赖: Glibc

测试套件依赖: 无

必须在此之前安装: Python 和 XML::Parser

可选依赖项: 无

Expect

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Glibc、Grep、Make、Patch、Sed 和 Tcl

运行时必需: Glibc 和 Tcl 测试套件依赖:

无 必须在此之前安装: 无

可选依赖项: Tk

文件

安装依赖项: Bash、Binutils、Bzip2、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Make、Sed、Xz 以及 Zlib

运行时必需: Glibc、Bzip2、Xz 和 Zlib

测试套件依赖项: 无 必须在此之前安装:

无可选依赖项: libseccomp

查找工具集

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、Make、Sed 和 Texinfo

运行时必需: Bash 和 Glibc

测试套件依赖: DejaGNU、Diffutils 和 Expect

必须在此之前安装: 无可选依赖

项: 无

Flex

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、M4、Make、Patch、Sed 和 Texinfo

运行时依赖: Bash、Glibc 和 M4

测试套件依赖: Bison 和 Gawk

必须预先安装: Binutils、IProute2、Kbd、Kmod 和 Man-DB

可选依赖项: 无

Flit-Core

安装依赖: Python

运行时依赖: Python

测试套件依赖: 无可用测试套件

必须预先安装: Wheel

可选依赖项: pytest 和 testpath

Gawk

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、GMP、Grep、Make、MPFR、Patch、Readline、Sed 和 Texinfo

运行时依赖: Bash、Glibc 和 Mpfr

测试套件依赖项: Diffutils 必须在此之前

安装: 无可选依赖项: libsigsegv

GCC

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Findutils、Gawk、GCC、Gettext、Glibc、GMP、Grep、M4、Make、MPC、MPFR、Patch、Perl、Sed、Tar、Texinfo 和 Zstd

运行时必需: Bash、Binutils、Glibc、Mpc 和 Python

测试套件依赖: DejaGNU、Expect 和 Shadow

必须在此之前安装: 无

可选依赖项: GDC、GNAT 和 ISL

GDBM

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Grep、Make 和 Sed

运行时依赖项: Bash、Glibc 和 Readline

测试套件依赖项: 无

必须在此之前安装: 无

可选依赖项: 无

Gettext

安装依赖项: Bash、Binutils、Coreutils、Gawk、GCC、Glibc、Grep、Make、Ncurses、Sed 和 Texinfo

运行时依赖: Acl、Bash、Gcc 和 Glibc; 测试套件依

赖: Diffutils、Perl 和 Tcl; 必须在以下软件之前安装:

Automake 和 Bison; 可选依赖项: libunistring 和

libxml2

Glibc

安装依赖项: Bash、Binutils、Bison、Coreutils、Diffutils、Gawk、GCC、Gettext、Grep、Gzip、Linux API 头文件、Make、Perl、Python、Sed 和 Texinfo

运行时需求: 无

测试套件依赖: File

必须在此之前安装: 无

可选依赖项: 无

GMP

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、M4、Make、Sed 以及 Texinfo

运行时必需: GCC 和 Glibc

测试套件依赖: 无

必须在以下软件之前安装: MPFR 和 GCC

可选依赖项: 无

Gperf

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc 和 Make

运行时需求: GCC 和 Glibc

测试套件依赖: Diffutils 和 Expect

必须在此之前安装: 无

可选依赖项: 无

Grep

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Gettext、Glibc、Grep、Make、Patch、Sed 和 Texinfo

运行时必需: Glibc

测试套件依赖: Gawk

必须在以下软件前安装: Man-DB

可选依赖项: PCRE2 和 libsigsegv

Groff

安装依赖项: Bash、Binutils、Bison、Coreutils、Gawk、GCC、Glibc、Grep、Make、Patch、Sed 以及 Texinfo

运行时依赖: GCC、Glibc 和 Perl

无可用测试套件

必须在安装前完成: Man-DB 可选依赖项: ghostscript 和 Uchardet

GRUB

安装依赖项: Bash、Binutils、Bison、Coreutils、Diffutils、GCC、Gettext、Glibc、Grep、Make、Ncurses Sed、Texinfo 和 Xz

运行时依赖: Bash、GCC、Gettext、Glibc、Xz 和 Sed

测试套件依赖: 无

必须预先安装: 无

可选依赖项: 无

Gzip

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Grep、Make、Sed 和 Texinfo

运行时必需: Bash 和 Glibc

测试套件依赖: Diffutils 和 Less

必须在此之前安装: Man-DB

可选依赖项: 无

Iana-Etc

安装依赖项: Coreutils

运行时必需: 无

测试套件依赖: 无可用测试套件

必须在此之前安装: Perl

可选依赖项: 无

Inetutils

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Grep、Make、Ncurses、Patch、Sed、Texinfo 以及 Zlib

运行时依赖项: GCC、Glibc、Ncurses 和 Readline

测试套件依赖项: 无 必须在此之前

安装: Tar 可选依赖项: 无

Intltool

安装依赖项: Bash、Gawk、Glibc、Make、Perl、Sed 和 XML::Parser

运行时需求: Autoconf、Automake、Bash、Glibc、Grep、Perl 和 Sed

测试套件依赖: Perl 必须预先安

装: 无 可选依赖项: 无

IProute2

安装依赖项: Bash、Bison、Coreutils、Flex、GCC、Glibc、Make、Libcap、Libelf、Linux API 头文件
Pkgconf 和 Zlib

运行时必需: Bash、Coreutils、Glibc、Libcap、Libelf 和 Zlib

测试套件依赖: 无可用测试套件

必须在此之前安装: 无

可选依赖项: Berkeley DB、iptables、libbpf、libmnl 和 libtirpc

Jinja2

安装依赖项: MarkupSafe、Python、Setuptools 和 Wheel

运行时必需: MarkupSafe 和 Python

测试套件依赖: 无可用测试套件

必须预先安装: Udev

可选依赖项: 无

Kbd

安装依赖项: Bash、Binutils、Bison、Coreutils、Flex、GCC、Gettext、Glibc、Gzip、Make、Patch 和 Sed

运行时必需: Bash、Coreutils 和 Glibc

测试套件依赖: 无

必须预先安装: 无

可选依赖项: Linux-PAM

Kmod

安装依赖项: Bash、Binutils、Bison、Coreutils、Flex、GCC、Gettext、Glibc、Gzip、Make、OpenSSL
Pkgconf、Sed、Xz 和 Zlib

运行时依赖: Glibc、Xz 和 Zlib

测试套件依赖: 无可用测试套件

必须预先安装: Udev

可选依赖项: scdoc (用于生成手册页)

更少

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Glibc、Grep、Make、Ncurses 和 Sed

运行时依赖: Glibc 和 Ncurses

测试套件依赖: 无可用测试套件

必须在以下软件之前安装: Gzip

可选依赖项: PCRE2 或 PCRE

Libcap

安装依赖项: Attr、Bash、Binutils、Coreutils、GCC、Glibc、Perl、Make 和 Sed

运行时必需: Glibc

测试套件依赖项: 无

必须在以下软件之前安装: IProute2 和 Shadow

可选依赖项: Linux-PAM

Libelf

安装依赖项: Bash、Binutils、Bzip2、Coreutils、GCC、Glibc、Make、Xz、Zlib 和 Zstd

运行时依赖项: Bzip2、Glibc、Xz、Zlib 和 Zstd

测试套件依赖项: 无

必须在此之前安装: IProute2 和 Linux

可选依赖项: 无

Libffi

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Make 和 Sed

运行时依赖: Glibc; 测试套件依赖:

DejaGnu; 必须在此之前安装:

Python; 可选依赖项: 无

Libpipeline

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Make、Sed 和 Texinfo

运行时需求: Glibc 测试套件依赖: Check 和

Pkgconf 必须在以下软件之前安装: Man-DB 可

选依赖项: 无

Libtool

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Make、Sed 和 Texinfo

运行时必需: Autoconf、Automake、Bash、Binutils、Coreutils、File、GCC、Glibc、Grep、Make 和 Sed

测试套件依赖: Autoconf、Automake 和 Findutils

必须预先安装: 无

可选依赖项: 无

Libxcrypt

安装依赖: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Make、Perl 和 Sed

运行时必需: Glibc

测试套件依赖: 无

必须在以下软件之前安装: Perl、Python、Shadow 和 Udev

可选依赖项: 无

Linux

安装依赖项: Bash、Bc、Binutils、Coreutils、Diffutils、Findutils、GCC、Glibc、Grep、Gzip、Kmod、Libelf、Make、Ncurses、OpenSSL、Perl 和 Sed

运行时必需: 无

测试套件依赖: 无可用测试套件

必须预先安装: 无

可选依赖项: cpio、LLVM (含 Clang) 及 Rust-bindgen

Linux API 头文件

安装依赖项: Bash、Binutils、Coreutils、Findutils、GCC、Glibc、Grep、Gzip、Make、Perl 和 Sed
 运行时依赖: 无
 测试套件依赖: 无 可用测试套件
 安装前提条件: 无 可选依赖项: 无

Lz4

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc 和 Make
 运行时必需: Glibc
 测试套件依赖: Python
 安装前提条件: Zstd
 可选依赖项: 无

M4

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Grep、Make、Sed 和 Texinfo
 运行时必需: Bash 和 Glibc
 测试套件依赖: Diffutils
 必须在以下软件之前安装: Autoconf 和 Bison
 可选依赖项: libsigsegv

Make

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、Make、Sed 和 Texinfo
 运行时需求: Glibc
 测试套件依赖: Perl 和 Procps-ng
 必须在此之前安装: 无
 可选依赖项: Guile

Man-DB

安装依赖项: Bash、Binutils、Bzip2、Coreutils、Flex、GCC、GDBM、Gettext、Glibc、Grep、Groff、Gzip
 Less、Libpipeline、Make、Pkgconf、Sed 和 Xz
 运行时依赖: Bash、GDBM、Groff、Glibc、Gzip、Less、Libpipeline 和 Zlib
 测试套件依赖: Util-linux
 必须预先安装: 无
 可选依赖项: libseccomp 和 po4a

Man-Pages

安装依赖项: Bash、Coreutils、Make 和 Sed
 运行时需求: 无
 测试套件依赖: 无 可用测试套件
 必须在此之前安装: 无 可选依赖项: 无

MarkupSafe

安装依赖项: Python、Setuptools 和 Wheel

运行时依赖: Python

测试套件依赖: 无可用测试套件

必须在安装前安装: Jinja2 可选依赖

项: 无

Meson

安装依赖项: Ninja、Python、Setuptools 和 Wheel

运行时必需: Python

无可用测试套件

必须在以下组件之前安装: Udev

可选依赖项: 无

MPC

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、GMP、Make、MPFR、Sed 以及 Texinfo

运行时依赖: Glibc、GMP 和 MPFR

测试套件依赖: 无

必须在以下软件之前安装: GCC

可选依赖项: 无

MPFR

安装依赖: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、GMP、Make 和 Sed Texinfo

运行时必需: Glibc 和 GMP

测试套件依赖: 无

必须在以下软件之前安装: Gawk 和 GCC

可选依赖项: 无

Ncurses

安装依赖项: Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Grep、Make、Patch 和 Sed

运行时必需: Glibc

测试套件依赖项: 无可用测试套件

必须在此之前安装: Bash、GRUB、Inetutils、Less、Procps-ng、Psmisc、Readline、Texinfo、Util-linux 和 Vim

Ninja

安装依赖项: Binutils、Coreutils、GCC 和 Python

运行时必需: GCC 和 Glibc

测试套件依赖项: cmake

必须在此之前安装: Meson

可选依赖项: Asciidoc、Doxygen、Emacs 和 re2c

OpenSSL

安装依赖项: Binutils、Coreutils、GCC、Make 和 Perl

运行时依赖: Glibc 和 Perl；测试套件依赖: 无

必须在此之前安装: Coreutils、Kmod、Linux 和 Udev

可选依赖项: 无

补丁

安装依赖项: Attr、Bash、Binutils、Coreutils、GCC、Glibc、Grep、Make 和 Sed

运行时必需: Attr 和 Glibc

测试套件依赖: Diffutils

必须在之前安装: 无

可选依赖项: Ed

Perl

安装依赖项: Bash、Binutils、Coreutils、Gawk、GCC、GDBM、Glibc、Grep、Libxcrypt、Make、Sed 和 Zlib

运行时依赖: GDBM、Glibc 和 Libxcrypt

测试套件依赖: Iana-Etc、Less 和 Procps-ng

必须在此之前安装: Autoconf

可选依赖: Berkeley DB

Pkgconf

安装依赖项: Bash、Binutils、Coreutils、Gawk、GCC、Glibc、Grep、Make 和 Sed

运行时依赖: Glibc

测试套件依赖: 无

必须在此之前安装: Binutils、E2fsprogs、IProute2、Kmod、Man-DB、Procps-ng、Python、Udev 和 Util-linux

可选依赖项: 无

Procps-ng

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Make、Ncurses 和 Pkgconf

运行时需求: Glibc

测试套件依赖: DejaGNU

必须在此之前安装: 无

可选依赖项: elogind

Psmisc

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、Make、Ncurses 和 Sed

运行时需求: Glibc 和 Ncurses

无可用测试套件

无需预先安装的依赖项: 无

可选依赖项: 无

Python

安装依赖项: Bash、Binutils、Coreutils、Expat、GCC、Gdbm、Gettext、Glibc、Grep、Libffi、Libxcrypt、Make、Ncurses、OpenSSL、Pkgconf、Sed 和 Util-linux

运行时依赖项: Bzip2、Expat、Gdbm、Glibc、Libffi、Libxcrypt、Ncurses、OpenSSL 和 Zlib

测试套件依赖项: GDB 和 Valgrind 必须预先安装: Ninja 可选依

赖项: Berkeley DB、libnsl、SQLite 和 Tk

Readline

安装依赖项: Bash、Binutils、Coreutils、Gawk、GCC、Glibc、Grep、Make、Ncurses、Patch、Sed 和 Texinfo

运行时必需项: Glibc 和 Ncurses

测试套件依赖项: 无可用测试套件

必须在以下软件之前安装: Bash、Bc 和 Gawk

可选依赖项: 无

Sed

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、Make、Sed 和 Texinfo

运行时必需: Acl、Attr 和 Glibc

测试套件依赖项: Diffutils 和 Gawk

必须在以下软件之前安装: E2fsprogs、File、Libtool 和 Shadow

可选依赖项: 无

SetupTools

安装依赖项: Python 和 Wheel (运行时必需)

测试套件依赖: 无可用测试套件

必须提前安装: Jinja2、MarkupSafe 和 Meson

可选依赖项: 无

影子密码

安装依赖项: Acl、Attr、Bash、Binutils、Coreutils、Diffutils、Findutils、Gawk、GCC、Gettext、Glibc、Grep、Libcap、Libxcrypt、Make 和 Sed

运行时必需: Glibc 和 Libxcrypt

测试套件依赖: 无可用测试套件

必须在此之前安装: Coreutils 可选依赖项: CrackLib

和 Linux-PAM

系统日志守护进程

安装依赖项: Binutils、Coreutils、GCC、Glibc、Make 和 Patch

运行时必需: Glibc

测试套件依赖: 无可用测试套件

必须预先安装: 无 可选依赖项: 无

SysVinit

安装依赖项: Binutils、Coreutils、GCC、Glibc、Make 和 Sed

运行时依赖: Glibc

测试套件依赖: 无可用测试套件

必须在此之前安装: 无

可选依赖项: 无

Tar

安装依赖项: Acl、Attr、Bash、Binutils、Bison、Coreutils、GCC、Gettext、Glibc、Grep、Inetutils、Make
Sed 和 Texinfo

运行时必需: Acl、Attr、Bzip2、Glibc、Gzip 和 Xz

测试套件依赖: Autoconf、Diffutils、Findutils、Gawk 和 Gzip

必须先安装: 无

可选依赖项: 无

Tcl

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Glibc、Grep、Make 和 Sed

运行时依赖: Glibc 和 Zlib

测试套件依赖: 无

必须在此之前安装: 无

可选依赖项: 无

Texinfo

安装依赖项: Bash、Binutils、Coreutils、GCC、Gettext、Glibc、Grep、Make、Ncurses、Patch 和 Sed

运行时依赖: Glibc 和 Ncurses

测试套件依赖: 无

必须在此之前安装: 无

可选依赖项: 无

Udev

安装依赖项: Acl、Bash、Binutils、Coreutils、Diffutils、Gawk、GCC、Glibc、Gperf、Grep、Jinja2、Libcap
Libxcrypt、Meson、OpenSSL、Pkgconf、Sed、Util-linux 和 Zstd

运行时必需: Acl、Glibc、Libcap、OpenSSL 和 Util-linux

测试套件依赖项: 无 必须在之前安装:

Util-linux 可选依赖项: 无

工具集-util-linux

安装依赖项: Bash、Binutils、Coreutils、Diffutils、File、Findutils、Gawk、GCC、Gettext、Glibc、Grep、
Make、Ncurses、Pkgconf、Sed、Udev 和 Zlib

运行时依赖项: Glibc、Ncurses、Readline、Udev 和 Zlib

测试套件依赖项: 无

必须预先安装: 无

可选依赖项: Asciidoctor、Libcap-NG、libeconf、libuser、libutempter、Linux-PAM、smartmontools、
po4a 以及 slang

Vim

安装依赖项: Acl、Attr、Bash、Binutils、Coreutils、Diffutils、GCC、Glibc、Grep、Make、Ncurses 和 Sed

运行时依赖: Acl、Attr、Glibc、Python、Ncurses 和 Tcl

测试套件依赖: 无

必须预先安装: 无

可选依赖项: Xorg、GTK+2、LessTif、Ruby 和 GPM

轮子

安装依赖项: Python 和 Flit-core (运行时必需)

测试套件依赖: 无可用测试套件

必须提前安装: Jinja2、MarkupSafe、Meson 和 Setuptools

可选依赖项: 无

XML 解析器

安装依赖项: Bash、Binutils、Coreutils、Expat、GCC、Glibc、Make 和 Perl

运行时必需: Expat、Glibc 和 Perl

测试套件依赖: Perl

必须在安装前完成: Intltool

可选依赖项: 无

Xz

安装依赖项: Bash、Binutils、Coreutils、Diffutils、GCC、Glibc 和 Make

运行时必需: Glibc 测试套件依赖项: 无

必须在此之前安装: File、GRUB、Kmod、Libelf、Man-DB 和 Udev

可选依赖项: 无

Zlib

安装依赖项: Bash、Binutils、Coreutils、GCC、Glibc、Make 和 Sed

运行时必需: Glibc 测试套件依赖项: 无

必须在以下软件之前安装: File、Kmod、Libelf、Perl 和 Util-linux

可选依赖项: 无

Zstd

安装依赖项: Binutils、Coreutils、GCC、Glibc、Gzip、Lz4、Make、Xz 和 Zlib

运行时必需: Glibc

测试套件依赖项: 无

必须在以下软件之前安装: Binutils、GCC、Libelf 和 Udev

可选依赖项: 无

附录 D. 启动与系统配置脚本版本-20240825

本附录所列脚本按其常规存放目录排序，依次为 /etc/rc.d/init.d、/etc/sysconfig、/etc/sysconfig/network-devices 以及 /etc/sysconfig/network-devices/services。各章节内文件按常规调用顺序排列。

D.1. /etc/rc.d/init.d/rc

rc 脚本是 init 调用的首个脚本，负责启动引导流程。

```
#!/bin/bash
#####
##### 开始 rc 脚本 #####
##### 功能描述
: 主运行级别控制脚本 ## 作者 : Gerard Beekmans - gerard@linuxfromscratch.org ## : DJ Lucas -
dj@linuxfromscratch.org ## 更新维护 : Bruce Dubbs - bdubbs@linuxfromscratch.org ## : Pierre Labastie -
pierre@linuxfromscratch.org ## 版本 : LFS 7.0 ## 更新说明 : 2022 年 3 月 24 日更新: S/K 文件新语义
# - 不再检测 S 脚本在上个运行级别是否为 K 脚本 # 改为检测它们在上个运行级别是否非 S 脚本 # - 不再检测
K 脚本在上个运行级别是否为 S 脚本 # 改为检测它们在上个运行级别是否非 K 脚本 # - 运行级别 0 或 6
中的 S 脚本现在以 # "script start" 方式运行 (原为"script stop")



#####
. /lib/lsb/init-functions

打印错误信息()
{
    记录失败信息 # 调用时设置 $i 变量 MSG="失败: \n\n 您不应该看到这条错误信息。 \n\n" MSG="${MSG}"
    这意味着在 \n" MSG="${MSG}${i}, \n" MSG="${MSG} 中发生了未预见的错误，返回值为 ${error_value}。
    \n"
}

MSG="${MSG}" 如果您能追踪到这个错误是由 \n" MSG="${MSG}${DISTRO_MINI} 手册提供的某个文件中的 bug 引起
的， \n" MSG="${MSG} 请通过 ${DISTRO_CONTACT} 联系我们。 \n" 记录失败信息 "${MSG}"

记录提示信息 "按回车键继续..." 等待用户输入 }

检查脚本状态() { # 调用时设置 $i 参数
if [ ! -f ${i} ]; then
    记录警告信息 "${i} 不是有效的符号链接。"
    脚本状态="1"
```

如果

```
if [ ! -x ${i} ]; then log_warning_msg "${i} 不可执行, 跳过."
SCRIPT_STAT="1" fi }
```

运行()

```
{ if [ -z $interactive ]; then
    ${1} ${2}
    返回 $?
fi

while true; do read -p "运行 ${1} ${2} (是/否/继续)? " -n 1 runit echo
```

```
case ${runit} in
c | C)
    交互式=""
    ${1} ${2}
    返回值=${?}
    中断;
;;
n | N)
    返回 0
;;
y | Y)
    ${1} ${2}
    ret=${?}
    中断
;;
esac
```

完成

返回 \$ret

}

```
# 读取本地设置/覆盖项 [ -r /etc/sysconfig/rc.site ] && source /etc/sysconfig/rc.site
```

```
DISTRO=${DISTRO:-"Linux From Scratch"} DISTRO_CONTACT=${DISTRO_CONTACT:-"lfs-dev@lists.linuxfromscratch.org (需注册)"}
DISTRO_MINI=${DISTRO_MINI:-"LFS"} IPROMPT=${IPROMPT:-"no"}
```

```
# 以下 3 个信号不会导致脚本退出 trap "" INT QUIT TSTP
```

```
[ "${1}" != "" ] && runlevel=${1}
```

如果 ["\${runlevel}" == ""]; 那么 echo "用法:
\${0} <运行级别>" >&2 退出 1 结束

```
前次=${PREVLEVEL} [ "${前次}" == "" ] && 前次=N
```

```

] && previous=N if [ ! -d /etc/rc.d/rc${runlevel}.d ]; then log_info_msg
"/etc/rc.d/rc${runlevel}.d does not exist.\n" exit 1 fi

if [ "$runlevel" == "6" -o "$runlevel" == "0" ]; then IPROMPT="no"; fi

# 注意: 在${LOGLEVEL:-7}中, 是冒号':'短横线'-'和数字 7, 而非减号 7 if [ "$runlevel" ==
"S" ]; then [ -r /etc/sysconfig/console ] && source /etc/sysconfig/console dmesg -n
"${LOGLEVEL:-7}" fi

if [ "${IPROMPT}" == "yes" -a "$runlevel" == "S" ]; then # 发行版欢迎字符串的总长度(不包含转义码)
wlen=${wlen:-$(echo "Welcome to ${DISTRO}" | wc -c)} welcome_message=${welcome_message:-"Welcome to
${INFO} ${DISTRO} ${NORMAL}"}

# 交互式字符串总长度(不含转义码) ilen=${ilen:-$(echo "按'I'键进入交互式启动" | wc -c)}
i_message=${i_message:-"按'${FAILURE} I${NORMAL}'键进入交互式启动"}

# dcol 和 icol 是消息前的空格, 用于将消息居中显示 # itime 是等待用户按键的时间 wcol=$(( ( ${COLUMNS}
- ${wlen} ) / 2 )) icol=$(( ( ${COLUMNS} - ${ilen} ) / 2 ))

itime=${itime:-"3"}

echo -e "\n\n" echo -e "\033[${wcol}G${welcome_message}" echo -e
"\033[${icol}G${i_message}${NORMAL}" echo "" read -t "${itime}" -n 1
interactive 2>&1 > /dev/null fi

# 将变量转为小写 [ "${interactive}" == "I" ] && interactive="i"
[ "${interactive}" != "i" ] && interactive=""

# 如果存在运行级别 S 的状态文件则读取 [ -r /run/interactive ] &&
source /run/interactive

# 停止所有标记为 K 的服务, 除非该服务在上个运行级别已被标记为 K: # 脚本需自行确保不会尝试终止
未运行的服务 if [ "${previous}" != "N" ]; then for i in $(ls -v /etc/rc.d/rc${runlevel}.d/K* 2>
/dev/null) do

check_script_status if [ "${SCRIPT_STAT}" == "1"
]; then SCRIPT_STAT="0" continue fi

suffix=${i##/etc/rc.d/rc${runlevel}.d/K[0-9][0-9]} [ -e /etc/rc.d/rc${previous}.d/K[0-
9][0-9]$suffix ] && continue

运行 ${i} stop
错误码=${?}

if [ "${错误码}" != "0" ]; then 打印错误信息; fi done

```

fi

```
if [ "${previous}" == "N" ]; then export IN_BOOT=1; fi

if [ "$runlevel" == "6" -a -n "${FASTBOOT}" ]; then touch /fastboot fi
```

```
# 启动当前运行级别中标记为 S 的所有服务（除非在上个运行级别中已被标记为 S）
# 脚本需自行确保不会重复启动已运行的服务
for i in $( ls -v /etc/rc.d/rc${runlevel}.d/S* 2> /dev/null )
do
```

```
如果 [ "${previous}" != "N" ]; 那么 suffix=${i#/etc/rc.d/rc${runlevel}.d/S[0-9][0-9]} [ -e
/etc/rc.d/rc${previous}.d/S[0-9][0-9]$suffix ] && 继续 fi
```

检查脚本状态

```
如果 [ "${SCRIPT_STAT}" == "1" ]; 那么
SCRIPT_STAT="0" 继续 fi
```

运行 \${i} 启动

错误值=\${?}

```
如果 [ "${error_value}" != "0" ]; then 打印错误信息; fi 完成
```

```
# 当从运行级别 S 切换时存储交互变量，否则删除
如果 [ "${runlevel}" == "S" -a "${interactive}" == "i" ]; then
echo "interactive=\"$i\"" > /run/interactive 否则
```

```
rm -f /run/interactive 2> /dev/null fi
```

仅在首次启动时复制启动日志

```
if [ "${previous}" == "N" -a "${runlevel}" != "S" ]; then
cat $BOOTLOG >> /var/log/boot.log
```

标记启动结束

```
echo "-----" >> /var/log/boot.log
```

删除临时文件

```
rm -f $BOOTLOG 2> /dev/null
fi
```

rc 结束

D.2. /lib/lsb/init-functions

```
#!/bin/sh
#####
##### # # 开始 /lib/lsb/init-
funtions # # 描述：运行级别控制函数
```

```
#  
# 作者 : Gerard Beekmans - gerard@linuxfromscratch.org # : DJ Lucas - dj@linuxfromscratch.org  
# 更新 : Bruce Dubbs - bdubbs@linuxfromscratch.org # # 版本 : LFS 7.0 # # 备注 : 代码基于  
Matthias Benkmann 的 simpleinit-msb # http://winterdrache.de/linux/newboot/index.html # # 该文  
件应存放于/lib/lsb 目录下 #
```

```
#####
#####
```

```
## 环境设置 # 设置环境变量的默认值 umask 022 export  
PATH="/bin:/usr/bin:/sbin:/usr/sbin"
```

```
## 设置颜色命令 (通过 echo 使用) # 更多信息请查阅`man console_codes`手册 #  
参见"ECMA-48 图形渲染设置"章节 # # 警告: 当从 8 位字体切换到 9 位字体时 #  
Linux 控制台会将加粗标记(1;)重新解释为 # 9 位字体的顶部 256 个字形。此情况不
```

```
# 影响帧缓冲控制台
```

```
NORMAL="\033[0;39m" # 标准控制台灰色  
SUCCESS="\033[1;32m" # 成功提示为绿色  
WARNING="\033[1;33m" # 警告提示为黄色  
FAILURE="\033[1;31m" # 失败提示为红色  
INFO="\033[1;36m" # 信息提示为浅青色  
BRACKET="\033[1;34m" # 括号为蓝色
```

```
# 使用彩色前缀
```

```
BMPREFIX=""  
SUCCESS_PREFIX="$${SUCCESS} * ${NORMAL}"  
FAILURE_PREFIX="$${FAILURE} *****${NORMAL}"  
WARNING_PREFIX="$${WARNING} *** ${NORMAL}"  
SKIP_PREFIX="$${INFO} S ${NORMAL}"
```

```
SUCCESS_SUFFIX="$${BRACKET} [$${SUCCESS} OK ${BRACKET}]${NORMAL}"  
FAILURE_SUFFIX="$${BRACKET} [$${FAILURE} FAIL ${BRACKET}]${NORMAL}"  
WARNING_SUFFIX="$${BRACKET} [$${WARNING} WARN ${BRACKET}]${NORMAL}"  
SKIP_SUFFIX="$${BRACKET} [$${INFO} SKIP ${BRACKET}]${NORMAL}"
```

```
BOOTLOG=/run/bootlog  
KILLDELAY=3  
SCRIPT_STAT="0"
```

```
# 设置用户指定的环境变量, 例如 HEADLESS [ -r /etc/sysconfig/rc.site ] && .  
/etc/sysconfig/rc.site
```

```
# 如果设置了 HEADLESS 变量, 则使用该值  
# 如果文件描述符 1 或 2 (标准输出和标准错误) 未打开 # 或不指向终端设备, 则认为脚本处  
于无头模式 [ ! -t 1 -o ! -t 2 ] && HEADLESS=${HEADLESS:-yes}
```

```
if [ "x$HEADLESS" != "xyes" ] then ## 屏  
幕尺寸 # 获取当前屏幕尺寸
```

```

如果 [ -z "${COLUMNS}" ]; then
COLUMNS=$(stty size) COLUMNS=${COLUMNS##*}
} fi else COLUMNS=80 fi

] 那么 ## 屏幕尺寸 # 获取当前屏幕尺寸 # 使用远程连接（如串口）时，stty size 会返回 0 if [ "${COLUMNS}"
= "0" ]; then COLUMNS=80 fi

```

```

## 用于定位结果消息的测量值 COL=$(( ${COLUMNS} - 8 ))
WCOL=$(( ${COL} - 2 ))

```

```

## 设置光标位置命令，通过 echo 使用 SET_COL="\033[$COLG" # 定位到第$COL 个字符
SET_WCOL="\033[$WCOLG" # 定位到第$WCOL 个字符 CURS_UP="\033[1A\033[OG" # 上移
一行，定位到第 0 个字符 CURS_ZERO="\033[OG"

```

```

#####
# start_daemon()
#
# 用法: start_daemon [-f] [-n 优先级] [-p pid 文件] 程序路径 [参数...]
#
#
# 功能: 将指定程序作为守护进程运行
#
#
# 输入参数: -f: (强制) 即使程序已在运行也强制执行
#
# -n 优先级: 指定 nice 优先级等级, 参见'man nice(1)'手册
#
# -p pid 文件: 使用指定文件记录进程 ID
#
# 程序路径: 目标程序的完整路径
#
# 参数: 传递给程序(程序路径)的附加参数
#
#
# 返回值 (遵循 LSB 退出代码规范):
#   local force=""
#   local nice="0"
#   本地进程 ID 文件=""
#   本地进程 ID 列表=""
#   本地返回值=""
#
# 处理参数
当条件为真
执行
判断参数 "${1}" 属于以下哪种情况:

```

```

-f)
force="1"
位移 1
;;
-n)
优先级="${2}"
移动 2
;;

```

```

-p)
pidfile="${2}"
shift 2
;;

*)
return 2
;;

*)
程序="${1}"
中断
;;
esac
完成

# 检查程序是否有效
if [ ! -e "${program}" ]; then
return 5
#i执行
if [ -z "${force}" ]; then
if [ -z "${pidfile}" ]; then
# 通过发现进程确定 PID
pidlist=`pidofproc "${1}"`  

retval="$?"
else
# PID 文件包含所需的进程 ID
# 注意根据 LSB 规范要求，必须向 pidofproc 提供路径参数，
# 但当前实现或标准并未实际使用该路径参数
pidlist=`pidofproc -p "${pidfile}" "${1}"`  

retval="$?"
如果

# 仅返回状态值
# 记录日志消息应由初始化脚本（或发行版函数）负责处理！
case "${retval}" in

0)
# 程序已在正常运行，这是一个成功的启动。return 0 ;;

1)
# 程序未运行，但存在无效的 pid 文件
# 移除该 pid 文件并继续执行
rm -f "${pidfile}" ;;

3)
# 程序未运行且不存在 pid 文件
# 此处无需操作，交由 start_deamon 继续处理
;;

*)
# 其他由状态值返回的情况不应被解析
# 将作为未指定错误返回
return 1 ;;

esac
如果

```

```
# 开始执行! nice -n "${nice}"
"${@}" }

#####
##### killproc() # # 用法:
killproc [-p pid 文件] 路径名 [信号] # # # 用途: 向运行中的进程发送控制信号 # # # 输入参数: -p pid 文
件, 使用指定的 pid 文件 # # 路径名, 指定程序的路径 # # 信号, 向路径名指定的程序发送该信号 # # # 返回值
(依据 LSB 退出代码定义): # # 0 - 程序(路径名)已停止/已经停止, 或 # # 运行中的程序已成功接收指定信号并
停止 # # 1 - 通用或未指定错误 # # 2 - 无效或过多的参数 # # 5 - 程序未安装 # # 7 - 程序未运行且提供了信号 #
#####
##### killproc() {
```

local pidfile
本地程序
本地前缀
本地程序名 本地信号="-TERM" 本
地后备信号="-KILL" 本地无信号
本地进程列表 本地返回值 本地进
程 ID 本地延迟="30" 本地进程终
止 本地死亡时间

```
# 处理参数
while true; do
    case "${1}" in
        -p)
            pidfile="${2}"
            移动 2
            ;;
        *)
            程序="${1}" 如果 [ -n "${2}" ];
            那么
                信号="${2}"
                回退=""
            否则
                nosig=1
            如果
                # 若存在额外参数则报错
                if [ -n "${3}" ]; then
                    return 2
                否则
                    中断
                如果
```

;;

esac

完成

```
# 检查程序是否有效
if [ ! -e "${program}" ]; then
    return 5
#i 检查信号是否有效
check_signal "${signal}"
if [ "${?}" -ne "0" ]; then

# 获取进程 ID 列表
if [ -z "${pidfile}" ]; then
# 通过探测方式确定进程 ID
pidlist=`pidofproc "${1}"`  

retval="${?}"
else
    # PID 文件包含所需的进程 ID # 注意根据 LSB 规范要求，必须向 pidofproc 提供路径参数 # 但当前实现和标准并未实际使用该路径参数
    pidlist=`pidofproc -p "${pidfile}" "${1}"`  

    retval="${?}"
```

如果

```
# 仅返回状态值 # 记录日志消息应由初始化脚本（或发行版函数）负责处理
case "${retval}" in
```

0)

```
# 程序正常运行中 # 此处无需操作，交由 killproc 继续处理
;;
```

1)

```
# 程序未运行，但存在无效的 pid 文件 # 请删除该 pid 文件
```

```
progname=${program##*/}
```

```
if [[ -e "/run/${progname}.pid" ]]; then
    pidfile="/run/${progname}.pid" rm -f  

    "$pidfile" fi
```

```
# 仅当未传递信号时才算成功
```

```
if [ -n "${nosig}" ]; then
    返回 0
else
    return 7
如果
;;
```

3)

```
# 程序未运行且不存在 pid 文件
```

```
# 仅当未传递信号时才算成功
if [ -n "${nosig}" ]; then
```

```
    返回 0
否则
    返回 7
如果
;;
```

```

*) # 其他状态返回值不应被解析 # 而应作为未指定错误返回。返回 1 ;;

esac

# 对退出信号和控制信号执行不同操作 check_sig_type "${signal}"

if [ "${?}" -eq "0" ]; then # 该信号用于终止程序

    # 处理空进程列表的情况 (pid 文件仍存在但未给出信号) if [ "${pidlist}" != "" ];
    ]; then

        # 终止进程 ID 列表中的进程
        for pid in ${pidlist}; do

            kill -0 "${pid}" 2> /dev/null

            if [ "${?}" -ne "0" ]; then
                # 进程已终止, 继续处理下一个并假设一切正常
                continue
            else
                kill "${signal}" "${pid}" 2> /dev/null

                # 最多等待 ${delay}/10 秒 (以 0.1 秒为单位) 让 "${pid}" 终止

                while [ "${delay}" -ne "0" ]; do
                    kill -0 "${pid}" 2> /dev/null || piddead="1" if [
                        "${piddead}" = "1" ]; then break; fi sleep 0.1 delay=$(((
                            ${delay} - 1 ))" done

            fi
        done
    fi
done
fi

# 检查并移除陈旧的 PID 文件
if [ -z "${pidfile}" ]; then
    # 获取$program 的基本名称
    prefix=`echo "${program}" | sed 's/[^\-]*$//'
    progname=`echo "${program}" | sed "s@${prefix}@@"`"

    如果 [ -e "/run/${progname}.pid" ]; 那么
    rm -f "/run/${progname}.pid" 2> /dev/null 否则

    如果 [ -e "${pidfile}" ]; 那么 rm -f "${pidfile}" 2> /dev/null; 结束 结束

# 对于不期望程序退出的信号 日志

```

```

# 让 kill 命令执行其任务，并检查 kill 的返回值

else # check_sig_type - 信号并非用于终止程序
    for pid in ${pidlist}; do
        kill "${signal}" "${pid}" if [ "${?}" -ne "0" ]; then return 1; fi
    done fi }

#####
# pidofproc() 函数 #
#
# 用法: pidofproc [-p pidfile] pathname
#
#
# 功能: 该函数返回特定守护进程的一个或多个 PID
#
#
# 输入参数: -p pidfile, 使用指定的 pidfile 而非 pidof
# pathname, 指定程序的路径
#
#
# 返回值 (遵循 LSB 状态码定义):
# 0 - 成功 (PID 输出到标准输出)
# 1 - 程序已终止但 PID 文件仍存在 (输出残留 PID)
# 3 - 程序未运行 (无输出)
local pidfile
local program
本地前缀
本地程序名
本地进程列表
本地进程标识符 本地退出状态
="0"

# 处理参数
while true; do
    判断参数 "${1}" 属于以下哪种情况:

    -p)
        pidfile="${2}"
        shift 2
        ;;

    *)
        program="${1}" if [ -n "${2}" ]
        ]; then
            # 参数过多 # 由于这是状态检查, 返回未知状态 return 4
            else

                中断
                fi
                ;;

        esac
    完成

    # 如果未指定 PID 文件, 尝试自动查找 if [ -z "${pidfile}" ]; then

        # 获取程序的基本名称前缀= echo "${program}" | sed
        's/[^\/*$//`'

        if [ -z "${prefix}" ]; then
            progname="${program}"

```

```

否则 程序名=`echo "${program}" | sed "s@${prefix}@@@"` fi

# 如果存在同名 PID 文件，则假定该文件有效
if [ -e "/run/${progname}.pid" ]; then
pidfile="/run/${progname}.pid" fi fi

# 若 PID 文件已设置且存在，则使用该文件
if [ -n "$pidfile" -a -e "$pidfile" ]; then
# 使用 pidfile 第一行的值 pidlist=`/bin/head -n1 "$pidfile"`
else

# 使用 pidof 命令 pidlist=`pidof
"${program}"` fi

# 检查所有列出的 PID 是否正在运行 for pid in ${pidlist}; do
kill -0 ${pid} 2> /dev/null

如果 [ "${?}" -eq "0" ]; 那么
lpids="${lpids}${pid} " 否则

    退出状态="1"
fi
完成

如果 [ -z "${lpids}" -a ! -f "$pidfile" ]; 那么
    返回 3
否则
echo "${lpids}" 返回 "${exitstatus}"
fi }

#####
##### 状态进程() # # 用法: 状态进程 [-p 进程 ID 文件] 路径名 # # # 功能: 此函数将特定守护进程的状态打印到标准输出 # # # # 输入: -p 进程 ID 文件, 使用指定的进程 ID 文件而非 pidof 命令 # # 路径名, 指定程序的路径 # # # # 返回值: # # 0 - 状态已打印 # # 1 - 输入错误。未指定要检查的守护进程。
##### statusproc() { local
pidfile local pidlist

if [ "${#}" = "0" ]; then echo "用法: statusproc [-p pidfile] {程序
名}" exit 1 fi

# 处理参数
while true; do

```

```

case "${1}" in
  -p)
    pidfile="${2}"
    移动 2
    ;;
  *)
    如果 [ -n "${2}" ]; then
      echo "参数过多" return 1 else
        中断
      fi
    ;;
  esac
完成

如果 [ -n "${pidfile}" ]; 那么 pidlist=`pidofproc -p
"${pidfile}" $@` 否则 pidlist=`pidofproc $@` 结束

# 去除尾部空格 pidlist=`echo "${pidlist}" | sed -r 's/ +$//' `

base="${1##*/}"

if [ -n "${pidlist}" ]; then /bin/echo -e "${INFO}${base} 正在运行, 进程" \
"ID 为 ${pidlist}.${NORMAL}" else if [ -n "${base}" -a -e \
"/run/${base}.pid" ]; then
  /bin/echo -e "${WARNING}${1} 未运行但" "/run/${base}.pid 文件存
在.${NORMAL}" else if [ -n "${pidfile}" -a -e "${pidfile}" ]; then
  /bin/echo -e "${WARNING}${1} 未在运行" \ "但 ${pidfile} 存在。${NORMAL}" else
  /bin/echo -e "${INFO}${1} 未在运行。${NORMAL}" fi fi fi }

#####
##### 功能: 内部工具函数, 用于格式化时间戳 # # 以记录启动日志文件。设置 STAMP 变量。# # # # 返回值: 未使用 #
##### timespec() { STAMP=$(echo
`date +"%b %d %T %:z"` `hostname` ) " return 0 }
##### log_success_msg() #

```

```

# 用法: log_success_msg ["消息"]
#
# 功能: 向屏幕和启动日志文件打印成功状态消息
#
# 输入参数: $@ - 消息内容
#
# 返回值: 未使用
#####
log_success_msg() {

    if [ "x$HEADLESS" != "xyes" ]
    then
        /bin/echo -n -e "${BMPREFIX}${@}"
        /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"
    else
        logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`
        /bin/echo -e "${logmessage} OK"
    fi
    # 从日志文件中去除不可打印字符
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`

    timespec /bin/echo -e "${STAMP} ${logmessage} OK" >> ${BOOTLOG}

    返回 0
}

log_success_msg2()
{
    if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${BMPREFIX}${@}" /bin/echo -e
"${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}" else echo " OK" fi

    echo " OK" >> ${BOOTLOG}

    返回 0
}

#####
# log_failure_msg() ## 用
法: log_failure_msg ["消息"] ## ## ## 功能: 向屏幕和启动日志文件打印失败状态信息 ## ## ## 输入参数: $@ - 消息
内容 ## ## ## 返回值: 未使用 ##
#####
log_failure_msg() {

    if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${BMPREFIX}${@}" /bin/echo -e
"${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}" else logmessage=`echo "${@}" | sed
's/\\\033[^a-zA-Z]*.//g'` /bin/echo -e "${logmessage} 失败"
}

```

```

fi

# 从日志文件中去除不可打印字符

时间戳 logmessage=`echo "{$@}" | sed 's/\x033[^a-zA-Z]*./g'` /bin/echo -e
"${STAMP} ${logmessage} 失败" >> ${BOOTLOG}

}

返回 0

log_failure_msg2()
{
    if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${BMPREFIX}${@}" /bin/echo -e
    "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}" else echo "失败" fi

    echo "FAIL" >> ${BOOTLOG}

    返回 0
}

#####
##### log_warning_msg() # # 用
##### 法: log_warning_msg ["消息"] # # # 功能: 向屏幕和启动日志文件打印警告状态信息 # # # # 返回值: 未使用 #
##### log_warning_msg() {

if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${BMPREFIX}${@}" /bin/echo -e
"${CURS_ZERO}${WARNING_PREFIX}${SET_COL}${WARNING_SUFFIX}" else logmessage=`echo "{$@}" | sed
's/\x033[^a-zA-Z]*./g'` /bin/echo -e "${logmessage} 警告" fi

# 从日志文件中去除不可打印字符 logmessage=`echo "{$@}" | sed 's/\x033[^a-zA-
Z]*./g'` timespec /bin/echo -e "${STAMP} ${logmessage} WARN" >> ${BOOTLOG}

}

返回 0

log_skip_msg()
{
    if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${BMPREFIX}${@}" /bin/echo -e
    "${CURS_ZERO}${SKIP_PREFIX}${SET_COL}${SKIP_SUFFIX}" else logmessage=`echo "{$@}" | sed
's/\x033[^a-zA-Z]*./g'` /bin/echo "SKIP"
}

```

```

fi

# 从日志文件中去除不可打印字符 logmessage=`echo "${@}" | sed 's/\x033[^a-zA-Z]*.//g'` /bin/echo "SKIP" >> ${BOOTLOG}

返回 0
}

#####
##### log_info_msg() #####
##### 用法: log_info_msg 消息 ##### 功能: 向屏幕和启动日志文件打印信息消息。 ##### 不打印结尾的换行符。 ##### 返回值: 未使用 #
#####
log_info_msg() {
    if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${BMPREFIX}${{@}}" else
        logmessage=`echo "${@}" | sed 's/\x033[^a-zA-Z]*.//g'` /bin/echo -n -e
    "${@}" fi

# 从日志文件中去除不可打印字符 logmessage=`echo "${@}" | sed 's/\x033[^a-zA-Z]*.//g'` timespec /bin/echo -n -e "${STAMP} ${logmessage}" >>
${BOOTLOG}

返回 0
}

log_info_msg2()
{
    if [ "x$HEADLESS" != "xyes" ] then /bin/echo -n -e "${{@}}" else
        logmessage=`echo "${@}" | sed 's/\x033[^a-zA-Z]*.//g'` /bin/echo -n -e
    "${@}" fi

# 从日志文件中去除不可打印字符 logmessage=`echo "${@}" | sed 's/\x033[^a-zA-Z]*.//g'` /bin/echo -n -e "${logmessage}" >> ${BOOTLOG}

返回 0
}

#####
##### evaluate_retval() #####
##### 用途: 评估返回值并相应打印成功或失败信息 ##### 功能: 便捷函数用于终止信息提示 ##### 返回值: 未使用 #
#####
evaluate_retval() {

```

本地错误值="\$?!"

如果 [\${error_value} = 0]; 则 记录成功消息
否则 记录失败消息 2 结束 }

```
#####
##### 检查信号() # # 用法: 检查信号 [-{信号}] # # # 目的: 检查信号是否有效。该功能未在任何 LSB 草案中定义, # # 但需要检查信号以确定所选信号是否对其他函数 # # 是无效参数。 # # # 输入: 接受一个形如 -{信号} 的字符串值 # # # # 返回值: # # 0 - 成功 (信号有效) # # 1 - 信号无效 #
##### 检查信号() {
```

局部变量 valsig

为无效信号添加错误处理

```
valsig="-ALRM -HUP -INT -KILL -PIPE -POLL -PROF -TERM -USR1 -USR2"
valsig="$valsig -VTALRM -STKFLT -PWR -WINCH -CHLD -URG -TSTP -TTIN"
valsig="$valsig -TTOU -STOP -CONT -ABRT -FPE -ILL -QUIT -SEGV -TRAP"
valsig="$valsig -SYS -EMT -BUS -XCPU -XFSZ -0 -1 -2 -3 -4 -5 -6 -8 -9"
valsig="$valsig -11 -13 -14 -15 "
```

```
echo "$valsig" | grep -- "$1" > /dev/null
```

```
if [ "$?" -eq "0" ]; then
    返回 0
否则
    返回 1
fi
}
```

```
#####
##### check_sig_type() # # 用法: check_signal [-{信号} | {信号}] # # # 功能: 检查信号是程序终止信号还是控制信号 # # 该功能未在任何 LSB 草案中定义, 但需要检查信号以确定其用途是终止程序 # # 还是仅用于控制程序。 # # # # 输入: 接受单个字符串值, 格式为-{信号}或{信号} # # # # 返回值: # # 0 - 信号用于程序终止 # # 1 - 信号用于程序控制 # #
##### check_sig_type() {
```

local valsig

终止信号列表 (仅限于常用项)

```

valsig=" -ALRM -INT -KILL -TERM -PWR -STOP -ABRT -QUIT -2 -3 -6 -9 -14 -15 "
echo "${valsig}" | grep -- ${1} > /dev/null

if [ "${?}" -eq "0" ]; then
    返回 0
否则
    return 1
fi
}

#####
##### 功能: 若非无头系统则等待用户响应 #####
##### wait_for_user() { # 默认等待用户输入 [ ${HEADLESS=0} = "0" ] && read ENTER return 0 }

#####
##### 功能: 判断变量是否为 true | yes | 1 的实用函数 #####
##### is_true() { [ "$1" = "1" ] || [ "$1" = "yes" ] || [ "$1" = "true" ] || [ "$1" = "y" ] || [ "$1" = "t" ] }

# 结束 /lib/lsb/init-functions

```

D.3. /etc/rc.d/init.d/mountvirtfs

```

#!/bin/sh
#####
##### 功能描述 : 确保 proc、sysfs、run 和 dev 文件系统已挂载 #####
##### 作者 : Gerard Beekmans -
gerard@linuxfromscratch.org # DJ Lucas - dj@linuxfromscratch.org # 更新者 : Bruce Dubbs -
bdubbs@linuxfromscratch.org # Xi Ruoyao - xry111@xry111.site # 版本 : LFS 12.0 #
#####

### 初始化信息开始
# 提供: mountvirtfs # 必须启动: $first # 应该
启动: # 必须停止: # 应该停止: # 默认启动: S

```

```
# 默认停止项: # 简短描述: 启动时挂载所需的各种特殊文件系统 # 详细描述: 挂载/sys 和/proc 虚拟(内核)文
件系统。 # 挂载/run(tmpfs)和/dev(devtmpfs)。 # 仅当它们尚未挂载时执行此操作。 # 按照书中建议的内核配
置, dev # 应该由内核自动挂载。 # X-LFS-提供方: LFS ### 结束初始化信息
```

```
. /lib/lsb/init-functions

case "${1}" in
    start) # 在记录任何消息前确保/run 可用 if ! mountpoint /run >/dev/null; then

        mount /run || failed=1 fi
```

创建目录并设置权
限: /run/lock 权限模式 1777

记录信息消息 "正在挂载虚拟文件系统: \${INFO}/run"

如果 /proc 未挂载, 则记录信息消息 " \${INFO}/proc" 并以指定参数挂
载, 失败时标记错误

如果 /sys 未挂载, 则记录信息消息 " \${INFO}/sys" 并以指定参数挂载,
失败时标记错误

```
if ! mountpoint /dev >/dev/null; then log_info_msg2 "
${INFO}/dev" mount -o mode=0755, nosuid /dev || failed=1 fi
```

```
mkdir -p /dev/shm log_info_msg2 " ${INFO}/dev/shm" mount -
o nosuid, nodev /dev/shm || failed=1
```

```
mkdir -p /sys/fs/cgroup log_info_msg2 " ${INFO}/sys/fs/cgroup" mount -o
nosuid, noexec, nodev /sys/fs/cgroup || failed=1
```

```
(exit ${failed})
evaluate_retval if [ "${failed}" = 1 ];
then exit 1 fi
```

log_info_msg "在/dev 目录创建指向/proc 的符号链接: \${INFO}/dev/stdin" ln -sf /proc/self/fd/0
/dev/stdin || failed=1

```
log_info_msg2 " ${INFO}/dev/stdout" ln -sf /proc/self/fd/1
/dev/stdout || failed=1
log_info_msg2 " ${INFO}/dev/stderr" ln -sf /proc/self/fd/2
/dev/stderr || failed=1
```

```
log_info_msg2 " ${INFO}/dev/fd"
```

```

ln -sf /proc/self/fd /dev/fd || failed=1
if [ -e /proc/kcore ]; then log_info_msg2 "${INFO}/dev/core"
ln -sf /proc/kcore /dev/core || failed=1 fi

(exit ${failed})
evaluate_retval
退出 ${failed}
;;

*) echo "用法: ${0} {start}" 退出 1 ;;
esac

```

挂载虚拟文件系统结束

D.4. /etc/rc.d/init.d/modules

```

#!/bin/sh
#####
# 模块加载开始 #
# 描述: 模块自动加载脚本 #
# 作者: Zack Winkles #
# DJ Lucas - dj@linuxfromscratch.org #
# 更新: Bruce Dubbs - bdubbs@linuxfromscratch.org #
# 版本: LFS 7.0 #
#####

### 初始化信息开始 ###
# 提供功能: 模块加载 #
# 必需启动项: 挂载虚拟文件系统 #
# 建议启动项: #
# 必需停止项: #
# 建议停止项: #
# 默认启动级别: S #
# 默认停止级别: #
# 简短描述: 加载所需内核模块。 #
# 详细描述: 加载/etc/sysconfig/modules 中列出的模块。 #
# LFS 提供支持: LFS #
### 初始化信息结束 ###

# 确保内核支持模块功能
[ -e /proc/modules ] || exit 0

. /lib/lsb/init-functions

case "${1}" in
启动) # 若不存在模块文件或没有有效条目则退出 [ -r /etc/sysconfig/modules ] ||
退出 0 grep -E -qv '^(\$|#)' /etc/sysconfig/modules || 退出 0

```

记录信息消息 “正在加载模块: ”

```

# 仅当用户确实提供了需要加载的模块时，才尝试加载模块

while read module args; do

    # 忽略注释行和空行
    case "$module" in ""|#"*) continue ;;
    esac


    # 尝试加载模块，并传递提供的所有参数
    modprobe ${module} ${args} >/dev/null

    # 若模块加载成功则打印模块名，否则记录失败模块
    if [ $? -eq 0 ]; then
        log_info_msg2 "${module}"
    else
        failedmod="${failedmod} ${module}"
    fi
    done < /etc/sysconfig/modules

    # 在正确行打印模块加载成功信息
    log_success_msg2

    # 打印加载失败的模块列表
    if [ -n "${failedmod}" ]; then
        log_failure_msg "以下模块加载失败:${failedmod}"
        exit 1
    fi ;;

*) echo "用法: ${0} {start}" 退出 1 ;;
esac

exit 0

# 结束模块

```

D.5. /etc/rc.d/init.d/udev

```

#!/bin/sh
#####
##### 开始 udev #####
##### 描述：
Udev 冷插拔脚本 ## 作者：Zack Winkles, Alexander E. Patrakov ## DJ Lucas -
dj@linuxfromscratch.org ## 更新：Bruce Dubbs - bdubbs@linuxfromscratch.org ## Xi Ruoyao -
xry111@xry111.site ## 版本：LFS 12.0 #
#####

### 初始化信息开始 ###
# 提供: udev $time

```

```
# 必需启动: localnet # 建议启动: modules # 必需停止: # 建议停止: # 默认启动: S # 默认停止: # 简短描述:  
在/dev 目录下创建设备节点 # 详细描述: 在/dev 挂载临时文件系统并启动 udevd 守护进程 # 设备节点由 udev  
规则创建 # X-LFS-提供者: LFS ### 结束初始化信息
```

```
. /lib/lsb/init-functions

case "${1}" in
  start) 记录信息消息 "正在为/dev 目录创建设备节点..." if ! grep -q  
'[:space:]' /proc/mounts; then
```

记录失败消息 2 消息="失败:\n\n 无法在没有 SysFS 文件系统的情况下\n\n"
消息="\${消息}创建设备\n\n" 消息="\${消息}按回车键后, 系统将\n\n" 消息
="\${消息}关机并切断电源\n\n" 记录信息消息 "\$消息"

记录信息消息 "按回车键继续..." 等待用户输入
/etc/rc.d/init.d/halt start fi

```
# 启动 udev 守护进程以持续监视并处理# uevent 事件  
SYSTEMD_LOG_TARGET=kmsg /sbin/udevd --daemon
```

```
# 现在遍历/sys 目录以"冷插拔"已发现的设备  
/bin/udevadm trigger --action=add --type=subsystems  
/bin/udevadm trigger --action=add --type=devices  
/bin/udevadm trigger --action=change --type=devices
```

```
# 等待 udevd 处理我们触发的 uevent 事件  
if ! is_true "$_OMIT_UDEV_SETTLE"; then  
/bin/udevadm settle  
fi
```

```
# 如果系统存在基于 LVM 的分区, 确保它们被激活以便使用  
if [ -x /sbin/vgchange ]; then  
/sbin/vgchange -a y >/dev/null;
```

```
log_success_msg2  
;;
```

```
*) echo "用法 ${0} {start}" exit 1 ;;  
esac
```

退出 0

结束 udev

D.6. /etc/rc.d/init.d/swap

```
#!/bin/sh
#####
##### 交换空间控制脚本 #####
##### 描述：交换分区控制脚本 #####
##### 作者：杰拉德·比克曼斯 - gerard@linuxfromscratch.org #####
##### DJ·卢卡斯 - dj@linuxfromscratch.org #####
##### 更新：布鲁斯·杜布斯 - bdubbs@linuxfromscratch.org #####
##### 版本：LFS 7.0 #####
#####

### 初始化信息开始 ###
# 提供: 交换空间
# 必须启动于: udev # 建议启动于: 模块 # 必须停止于: 本地网络 # 建议停止于: $local_fs # 默认启动级别: S
# 默认停止级别: 0 6 # 简短描述: 激活与停用交换分区。 # 详细描述: 激活和停用在/etc/fstab 中定义的交换分区。
# X-LFS-提供者: LFS ### 结束初始化信息

. /lib/lsb/init-functions

case "${1}" in
    启动) log_info_msg "正在激活所有交换文件/分区..." swapon -a evaluate_retval ;;

    停止) log_info_msg "正在停用所有交换文件/分区..." swapoff -a evaluate_retval ;;

    重启)
        ${0} 停止
        sleep 1
        ${0} 启动
        ;;

    status) log_success_msg "正在获取交换分区状态。" swapon -s
        ;;

*) echo "用法: ${0} {start|stop|restart|status}" exit 1 ;;
esac
```

退出 0

交换分区设置结束

D.7. /etc/rc.d/init.d/setclock

```
#!/bin/sh
#####
##### 开始设置时钟 #####
##### 功能
##### 描述：设置 Linux 系统时钟 #####
##### 作者：杰拉德·比克曼斯 - gerard@linuxfromscratch.org #####
##### DJ·卢卡斯
##### - dj@linuxfromscratch.org #####
##### 更新者：布鲁斯·杜布斯 - bdubbs@linuxfromscratch.org #####
##### 版本：LFS
##### 7.0 #####
#####

### 初始化信息开始 ###
# 提供：
# 必需启动：
# 应启动： 模块 # 必需停止： # 应停止： $syslog # 默认启动： S # 默认停止： # 简短描述： 从硬件时钟存
# 储和恢复时间 # 详细描述： 启动时，系统时间从 hwclock 获取。 # 硬件时钟也可以在关机时设置。 # X-LFS-提
# 供者：LFS ### 结束 INIT 信息

. /lib/lsb/init-functions

[ -r /etc/sysconfig/clock ] && . /etc/sysconfig/clock

case "${UTC}" in
    yes|true|1) CLOCKPARAMS="${CLOCKPARAMS} --utc" ;;
    no|false|0) CLOCKPARAMS="${CLOCKPARAMS} --localtime" ;;

esac

case ${1} in
    启动) hwclock --hctosys ${CLOCKPARAMS} >/dev/null ;;
    停止) log_info_msg "正在设置硬件时钟..." hwclock --systohc
    ${CLOCKPARAMS} >/dev/null evaluate_retval ;;

*)

```

```
echo "用法: ${0} {start|stop}" exit 1 ;;
```

```
esac
```

```
退出 0
```

D.8. /etc/rc.d/init.d/checkfs

```
#!/bin/sh
#####
# 开始文件系统检查 ## 描述
# 文件系统检查 ## 作者 : 杰拉德·比克曼斯 - gerard@linuxfromscratch.org # A. 吕贝克 -
# luebke@users.sourceforge.net # DJ·卢卡斯 - dj@linuxfromscratch.org # 更新 : 布鲁斯·杜布斯 -
# bdubbs@linuxfromscratch.org # 版本 : LFS 7.0 ## 基于 LFS-3.1 及更早版本的文件系统检查脚本 ## 摘自 fsck 手册 # 0 - 无错误 # 1 - 已修复文件系统错误 # 2 - 应重新启动系统 # 4 - 未修复的文件系统错误 # 8 - 操作错误 # 16 - 用法或语法错误 # 32 - 用户请求取消 fsck # 128 - 共享库错误 #
#####

#####
```

```
### 初始化信息开始 ###
# 提供: checkfs # 必需启动: udev swap # 应该启动: # 必需停止: # 应该停止: # 默认启动: S
# 默认停止: # 简短描述: 挂载前检查本地文件系统 # 描述: 在挂载前检查本地文件系统 # X-
# LFS-提供者: LFS ### 结束初始化信息
```

```
. /lib/lsb/init-functions
```

```
case "${1}" in
  start) if [ -f /fastboot ]; then
    msg="/fastboot 文件存在, 将按请求跳过" msg="${msg} 文件系统检
    查。\\n" log_info_msg "${msg}" exit 0 fi
```

```
log_info_msg "正在以只读模式挂载根文件系统..."
```

```
mount -n -o remount,ro / >/dev/null

if [ ${?} != 0 ]; then log_failure_msg2 msg="\n\n 无法检查根文件系统，"
msg="$msg因为它无法以只读模式挂载。 \n\n" msg="$msg按回车键后，该系统将"
msg="$msg关机并切断电源。 \n\n" log_failure_msg "$msg"
```

```
log_info_msg "按回车键继续..." wait_for_user
/etc/rc.d/init.d/halt start else log_success_msg2 fi
```

```
if [ -f /forcefsck ]; then msg="发现/forcefsck 文件，正在按
要求" msg="$msg强制执行文件系统检查。" log_success_msg
"$msg" options="-f" else options="" fi
```

```
log_info_msg "正在检查文件系统..." # 注意：原-a 选项曾为-p；但若启用"$VERBOSE_FSCK"时，此
选项在 fsck.minix 等场景会失败
if is_true "$VERBOSE_FSCK"; then
fsck ${options} -a -A -C -T
else
fsck ${options} -a -A -C -T >/dev/null
fi
```

错误值=\${?}

```
if [ "${error_value}" = 0 ]; then
log_success_msg2
fi

if [ "${error_value}" = 1 ]; then
msg="\n 警告：\n\n 检测到文件系统错误"
msg="$msg并已完成自动修复。 \n"
msg="$msg 建议您二次确认"
msg="$msg所有问题是否已妥善解决。"
log_warning_msg "$msg"
fi

if [ "${error_value}" = 2 -o "${error_value}" = 3 ]; then
msg="\n 警告：\n\n 检测到文件系统错误"
msg="$msg并已完成修复，但由于"
msg="$msg错误性质特殊，需要重启系统。 \n\n"
msg="$msg按回车键后，"
msg="$msg系统将立即重启\n\n"
log_failure_msg "$msg"
fi

log_info_msg "按 Enter 键继续..." wait_for_user reboot -
f fi
```

```
如果 [ "${error_value}" -gt 3 -a "${error_value}" -lt 16 ]; then msg="\n 严重错误:
\n\n 检测到文件系统错误 " msg="$msg" 且无法自动修复。 \n 该系统 " msg="$msg" 无法继续
启动, 将 " msg="$msg" 保持关机状态, 直到系统管理员 " msg="$msg" 手动修复这些错误。
\n\n" msg="$msg" 按下回车键后, 系统将立即 " msg="$msg" 关机并切断电源。 \n\n"
log_failure_msg "$msg"
```

```
log_info_msg "按回车键继续..." wait_for_user
/etc/rc.d/init.d/halt start fi
```

```
如果 [ "${error_value}" -ge 16 ]; then msg="严重错误: \n\n 运行
fsck 时发生意外故障 " msg="$msg"。退出错误 " msg="$msg" 代码:
$error_value"。 \n" log_info_msg $msg exit ${error_value} fi
```

```
退出 0
;;
*) echo "用法: ${0} {start}" 退出 1 ;;
esac
```

```
# 结束文件系统检查
```

D.9. /etc/rc.d/init.d/mountfs

```
#!/bin/sh
#####
# 挂载文件系统脚本开始 #
# 描述: 文件系统挂载脚本 # # 作者: Gerard Beekmans - gerard@linuxfromscratch.org # DJ Lucas -
# dj@linuxfromscratch.org # 更新: Bruce Dubbs - bdubbs@linuxfromscratch.org # # 版本: LFS 7.0 #
#####

### 初始化信息开始 ###
# 提供功能: $local_fs # 必须启动项: udev checkfs # 建议启动项: modules # 必须停止项: localnet # 建议停止项:
# 默认启动级别: S # 默认停止级别: 0 6 # 功能简述: 挂载/卸载在/etc/fstab 中定义的本地文件系统 # 详细描述: 以
# 读写模式重新挂载根文件系统, 并挂载/etc/fstab 中定义的所有剩余本地文件系统
```

```

# 开始。重新挂载根文件系统为只读并卸载
# 停止时卸载剩余的文件系统。
# X-LFS-Provided-By: LFS ### END
INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start) log_info_msg "以读写模式重新挂载根文件系统..." mount --options remount,rw /
    >/dev/null evaluate_retval

    # 移除与 fsck 相关的文件系统标记
    rm -f /fastboot /forcefsck

    # 确保/dev/pts 目录存在
    mkdir -p /dev/pts

    # 这将挂载所有选项列表中不包含_netdev 的文件系统
    # _netdev 表示网络文件系统

    log_info_msg "正在挂载剩余文件系统..."
    failed=0
    mount --all --test-opts no_netdev >/dev/null || failed=1
    evaluate_retval
    exit $failed ;;

停止) # 不要卸载虚拟文件系统如/run
日志信息 "正在卸载所有其他当前已挂载的文件系统..."
# 确保移除所有循环设备
losetup -D
umount --all --detach-loop --read-only \
--types notmpfs,nosysfs,nodevtmpfs,noproc,nodevpts >/dev/null
评估返回值

# 确保/以只读方式重新挂载（解决 umount 错误）
mount --options remount,ro /

# 如适用，使所有 LVM 卷组不可用
# 如果交换分区或/位于 LVM 分区上，此操作将失败
#if [ -x /sbin/vgchange ]; then /sbin/vgchange -an > /dev/null; fi
if [ -r /etc/mdadm.conf ]; then
    日志信息 "正在将阵列标记为干净状态..."
    mdadm --wait-clean --scan
    评估返回值
    fi ;;

*) echo "用法: ${0} {start|stop}" exit 1 ;;
esac

# 挂载文件系统结束

```

D.10. /etc/rc.d/init.d/udev_retry

```
#!/bin/sh
```

```

/bin/sh #####
# 开始 udev_retry 脚本
#
# 功能描述 : Udev 冷插拔处理脚本 (重试机制)
#
# 作者 : Alexander E. Patrakov
# DJ Lucas - dj@linuxfromscratch.org
# 更新者 : Bruce Dubbs - bdubbs@linuxfromscratch.org
# Bryan Kadzban
#
# 版本 : LFS 7.0 #
#####

### 初始化信息开始 ###
# 服务标识: udev_retry
# 依赖启动项: udev
# 建议启动项: $local_fs cleanfs
# 依赖停止项:
# 建议停止项:
# 默认启动级别: S
# 默认停止级别:
# 功能简述: 重放失败的 uevent 并创建设备节点
# 详细描述: 重放因硬件初始化缓慢而跳过的所有失败 uevent,
# 并创建所需的设备节点
# X-LFS-提供方: LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    log_info_msg "正在重试失败的 uevent (如有) ..."
    rundir=/run/udev
    # 来自 Debian 的说明: "复制在根目录挂载为读写模式前
    # 已生成的规则":
    对于 ${rundir}/tmp-rules--* 中的每个文件; 执行以下操作:
    目标文件=${文件##*tmp-rules--}
    [ "$目标文件" = '*' ] && 中断
    将 $文件 内容追加到 /etc/udev/rules.d/$目标文件
    删除 $文件
    ↳ ↳

    # 重新触发可能失败的 uevent 事件
    # 希望这次能成功执行
    /bin/sed -e 's/#.*$/ /' /etc/sysconfig/udev_retry | /bin/grep -v '^$' | \
    while read 行 ; do
      for 子系统 in $行 ; do
        /bin/udevadm trigger --subsystem-match=$子系统 --action=add
      完成
    完成

    # 现在等待 udevd 处理我们触发的 uevent 事件
    if ! is_true "$ OMIT_UDEV_RETRY_SETTLE"; then
      /bin/udevadm settle
    fi

    evaluate_retval
;;

```

```
*) echo "用法 ${0} {start}" exit 1 ;;
esac
```

退出 0

```
# 结束 udev_retry
```

D.11. /etc/rc.d/init.d/cleanfs

```
#!/bin/sh
#####
# 开始清理文件系统 #
# 描述: 清理文件系统 #
# 作者: Gerard Beekmans - gerard@linuxfromscratch.org #
# DJ Lucas - dj@linuxfromscratch.org #
# 更新: Bruce Dubbs - bdubbs@linuxfromscratch.org #
# 版本: LFS 7.0 #
#####
```

```
### 初始化信息开始 ###
# 提供: cleanfs
# 必需启动: $local_fs
# 建议启动:
# 必需停止:
# 建议停止:
# 默认启动: S
# 默认停止:
# 简短描述: 在启动过程中早期清理临时目录。
# 详细描述: 清理临时目录 /run、/var/lock 以及可选的 /tmp。cleanfs 还会创建 /run/utmp 以及
/etc/sysconfig/createfiles 中定义的所有文件。
# X-LFS-提供者: LFS
### 结束 INIT 信息
```

```
. /lib/lsb/init-functions
```

```
# 启动时创建文件/目录的函数
create_files() {
# 将文件描述符 9 重定向到标准输入
exec 9>&0 < /etc/sysconfig/createfiles
```

```
while read name type perm usr grp dtype maj min junk
do
# 忽略注释和空行
case "${name}" in
  ""|#\*) continue ;;
esac

# 忽略已存在的文件
if [ ! -e "${name}" ]; then # 根据类型创建对应内容

  case "${type}" in
    dir)
      创建目录 "${name}"
    ;;
  esac
fi
done
```

```

;;
文件)
:> "${name}"
;;
设备)
case "${dtype}" in
    字符设备) mknod "${name}" c ${maj} ${min} ;;
    块设备) mknod "${name}" b ${maj} ${min} ;;
    管道设备)

mknod "${name}" p
;;
*) 记录警告信息 "\n 未知设备类型: ${dtype}" ;;
esac ;;
*) 记录警告信息 "\n 未知
类型: ${type}" continue ;;
esac

# 同时设置权限
chown ${usr}:${grp} "${name}"
chmod ${perm} "${name}"
fi
done

# 关闭文件描述符 9 (结束重定向)
exec 0>&9 9>&-
return 0
}

case "${1}" in
start)
log_info_msg "正在清理文件系统: "

if [ "${SKIPTMPCLEAN}" = "" ]; then
log_info_msg2 "/tmp"
cd /tmp && find . -xdev -mindepth 1 ! -name lost+found -delete || failed=1
fi

> /run/utmp

如果 grep -q '^utmp:' /etc/group ; then chmod 664
/run/utmp chgrp utmp /run/utmp fi

(exit ${failed})
evaluate_retval

if grep -E -qv '#|$' /etc/sysconfig/createfiles 2>/dev/null; then log_info_msg "正在创建文
件和目录..." create_files # 始终返回 0 evaluate_retval fi

```

```
退出 $failed
;;
*) echo "用法: ${0} {start}" 退出 1 ;;
esac
```

```
# 清理文件系统结束
```

D.12. /etc/rc.d/init.d/console

```
#!/bin/sh
#####
# 控制台初始化开始 #
#
功能描述 : 设置键盘映射和屏幕字体 # # 作者 : 杰拉德·比克曼斯 - gerard@linuxfromscratch.org # 亚
历山大·E·帕特拉科夫 # DJ·卢卡斯 - dj@linuxfromscratch.org # 更新者 : 布鲁斯·杜布斯 -
bdubbs@linuxfromscratch.org # # 版本 : LFS 7.0 #
#####
```

```
### 初始化信息开始 ###
# 服务标识: console # 依赖启动项: $local_fs # 建议启动项: udev_retry # 依赖停止项: # 建议停止项: #
默认启动级别: S # 默认停止级别: # 功能简述: 配置本地化控制台 # 详细描述: 根据/etc/sysconfig/console
配置, # 为用户设置字体和语言环境 # X-LFS-技术支持: LFS ### 初始化信息结束
```

```
. /lib/lsb/init-functions
```

```
# 英语母语用户可能根本没有/etc/sysconfig/console 文件 [ -r /etc/sysconfig/console ] && .
/etc/sysconfig/console
```

```
失败计数=0
```

```
case "${1}" in
启动) # 检查是否需要执行任何操作 if [ -z
"${KEYMAP}"
] && [ -z "${KEYMAP_CORRECTIONS}" ] &&
[ -z "${FONT}" ] && [ -z "${LEGACY_CHARSET}" ] && ! is_true "${UNICODE}"; then exit 0
fi
```

```
# 以下不应出现任何虚假的错误!
log_info_msg "正在配置 Linux 控制台..."
```

```
# 判断是否使用帧缓冲控制台
[ -d /sys/class/graphics/fbcon ] && use_fb=1 || use_fb=0
```

```

# 确定设置控制台为所需模式的命令
is_true "${UNICODE}" && MODE_COMMAND="echo -en '\033%G' && kbd_mode -u" ||
MODE_COMMAND="echo -en '\033@\033(K' && kbd_mode -a"

# 在帧缓冲控制台上, UTF-8 模式下需要为每个虚拟终端设置字体。在非 UTF-8 模式下执行
此操作也无妨。

! is_true "${use_fb}" || [ -z "${FONT}" ] ||
MODE_COMMAND="${MODE_COMMAND} && setfont ${FONT}"

# 将该命令应用于/etc/inittab 中提到的所有控制台。重要提示: 在 UTF-8 模式下,
这应该在 setfont 之前执行, 否则会出现内核错误, 导致字体 unicode 映射无法使
用。

对于 TTY 终端设备 (通过`grep '^#[^#].*respawn:/sbin/agetty' /etc/inittab | grep -o
'\btty[:digit:]*\b'` 获取的列表), 执行以下操作: openvt -f -w -c ${TTY#tty} -- \
/bin/sh -c "${MODE_COMMAND}" || failed=1 完成循环

# 设置字体 (如果之前未设置) 和键盘映射 [ "${use_fb}" == "1" ] || [ -z "${FONT}" ] || setfont
$FONT || failed=1

[ -z "${KEYMAP}" ] || loadkeys ${KEYMAP} >/dev/null
2>&1 || failed=1

[ -z "${KEYMAP_CORRECTIONS}" ] || loadkeys ${KEYMAP_CORRECTIONS}
>/dev/null 2>&1 || failed=1

# 将键位映射从$LEGACY_CHARSET 转换为 UTF-8 编码
[ -z "$LEGACY_CHARSET" ] || dumpkeys -c "$LEGACY_CHARSET" | loadkeys -u >/dev/null 2>&1
|| failed=1

# 若上述任一命令执行失败, 顶部的 trap 会将$failed 设为 1
( exit $failed )
evaluate_retval

退出 $failed
;;

*) echo "用法: ${0} {start}" 退出 1 ;;
esac

# 控制台配置结束

```

D.13. /etc/rc.d/init.d/localnet

```

#!/bin/sh
#####

```

```

# 启动本地网络
# 描述: 回环设备
# 作者: Gerard Beekmans - gerard@linuxfromscratch.org
# DJ Lucas - dj@linuxfromscratch.org
# 更新: Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# 版本: LFS 7.0

#####
#####

### 初始化信息开始 ###
# 提供: localnet
# 必须启动: mountvirtfs
# 建议启动: modules
# 必须停止:
# 建议停止:
# 默认启动: S
# 默认停止: 0 6
# 简短描述: 启动本地网络。
# 详细描述: 设置机器主机名并启动
# 回环接口。
# X-LFS-提供者: LFS
### 结束初始化信息

. /lib/lsb/init-functions
[ -r /etc/sysconfig/network ] && . /etc/sysconfig/network
[ -r /etc/hostname ] && HOSTNAME=`cat /etc/hostname`


case "${1}" in
    启动) log_info_msg "正在启用环回接口..." ip addr add 127.0.0.1/8 label lo
    dev lo ip link set lo up evaluate_retval

        log_info_msg "正在设置主机名为${HOSTNAME}..." hostname ${HOSTNAME}
        evaluate_retval ;;

    停止) log_info_msg "正在关闭环回接口..." ip link set lo down evaluate_retval
    ;;

    重启)
        ${0} 停止
        sleep 1
        ${0} 启动
        ;;

    状态) echo "当前主机名为: $(hostname)" ip link
    show lo ;;

*) echo "用法: ${0} {start|stop|restart|status}" exit 1 ;;

```

```
esac
```

```
退出 0
```

```
# 结束本地网络配置
```

D.14. /etc/rc.d/init.d/sysctl

```
#!/bin/sh
#####
# 开始 sysctl 配置 #
# 功能描述：该文件通过 /etc/sysctl.conf 设置内核运行时参数 #
#
# 作者：内森·库尔森 (nathan@linuxfromscratch.org) #
# 马修·伯吉斯 (matthew@linuxfromscratch.org) #
# DJ·卢卡斯 - dj@linuxfromscratch.org #
# 更新者：布鲁斯·杜布斯 - bdubbs@linuxfromscratch.org #
#
# 版本：LFS 7.0 #
#####

### 初始化信息开始 ###
# 提供功能: sysctl
# Required-Start: mountvirtfs # Should-Start: console # Required-Stop: # Should-Stop: # Default-
Start: S # Default-Stop: # Short-Description: 修改 proc 文件系统 # Description: 根据/etc/sysctl.conf
配置文件修改 proc 文件系统。参见'man sysctl(8)'手册页。 # X-LFS-Provided-By: LFS ### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
    start) if [ -f "/etc/sysctl.conf" ]; then
        log_info_msg "正在设置内核运行时参数..." sysctl -q -p evaluate_retval fi
        ;;
    status)
        系统调用 -a
        ;;
*) echo "用法: ${0} {启动|状态}" 退出 1 ;;
择

退出 0

# 系统调用结束
```

D.15. /etc/rc.d/init.d/sysklogd

```
#!/bin/sh
#####
##### 开始 sysklogd #####
##### 描述
: Sysklogd 加载器 ## 作者 : Gerard Beekmans - gerard@linuxfromscratch.org ## DJ Lucas -
dj@linuxfromscratch.org ## 更新 : Bruce Dubbs - bdubbs@linuxfromscratch.org ## 更新 : Bruce Dubbs -
bdubbs@linuxfromscratch.org LFS12.1 ## 移除内核日志守护进程。该功能已 # 合并至 syslogd 中。 ## 版本 : LFS 7.0 #
#####

### 初始化信息开始 ###
# 提供: $syslog # 必需启动: $first localnet # 建议启动: # 必需停
止: $local_fs # 建议停止: sendsignals # 默认启动: 2 3 4 5 # 默认停
止: 0 1 6 # 简短描述: 启动系统日志守护进程。 # 描述: 启动系统日志
守护进程。 # /etc/fstab. # X-LFS-提供者: LFS ### 结束 INIT 信息

. /lib/lsb/init-functions

case "${1}" in
    启动) 日志信息提示 "正在启动系统日志守护进程..." 参数
    =${SYKLOGD_PARMS}' -m 0' 启动守护进程 /sbin/syslogd $参数 评估
    返回值 ;;
    停止) log_info_msg "正在停止系统日志守护进程..." killproc
    /sbin/syslogd evaluate_retval ;;

    重新加载) log_info_msg "正在重新加载系统日志守护进程配置文件..." pid=`pidofproc
    syslogd` kill -HUP "${pid}" evaluate_retval ;;

    重启)
    ${0} 停止
    sleep 1
    ${0} 启动
    ;;
esac
```

```
状态) 状态进程 /sbin/syslogd ;;

*) echo "使用方法: ${0} {启动|停止|重载|重启|状态}" exit 1 ;;
esac
```

退出 0

结束 sysklogd

D.16. /etc/rc.d/init.d/network

```
#!/bin/sh
#####
# 开始网络配置 #
# 描述 : 网络控制脚本 #
# 作者 : Gerard Beekmans - gerard@linuxfromscratch.org #
# Nathan Coulson - nathan@linuxfromscratch.org #
# Kevin P. Fleming - kpfelemin@linuxfromscratch.org #
# DJ Lucas - dj@linuxfromscratch.org #
# 更新 : Bruce Dubbs - bdubbs@linuxfromscratch.org #
# 版本 : LFS 7.0 #
#####
```

```
### 初始化信息开始 ###
# 提供功能: $network
# 启动依赖: $local_fs localnet swap
# 建议启动: $syslog firewalld iptables nftables
# 停止依赖: $local_fs localnet swap
# 建议停止: $syslog firewalld iptables nftables
# 默认启动级别: 2 3 4 5
# 默认停止级别: 0 1 6
# 功能简述: 启动并配置网络接口
# 详细描述: 启动并配置网络接口
# X-LFS-提供方: LFS
### 结束初始化信息 ###
```

```
case "${1}" in
  start)
    # 如果默认路由已存在, 说明网络已配置完成
    if ip route | grep -q "default"; then return 0; fi
    # 启动所有网络接口
    for file in /etc/sysconfig/ifconfig.*
    do
      interface=${file##*/ifconfig.}

      # 如果$file 为*则跳过 (因为未找到任何内容)
      if [ "${interface}" = "*" ]; then continue; fi

      /sbin/ifup ${interface} done ;;
```

stop)

```

# 卸载所有网络挂载的文件系统 umount --all --force --types
nfs,cifs,nfs4

# 反转列表
net_files="" for file in /etc/sysconfig/ifconfig.* do net_files="${file} ${net_files}" done

# 停止所有网络接口 for file in ${net_files} do
interface=${file##*/ifconfig.}

# 如果$file 为*则跳过 (因为未找到任何内容)
if [ "${interface}" = "*" ]; then continue; fi

# 检查接口是否存在
if [ ! -e /sys/class/net/$interface ]; then continue; fi

# 接口是否处于 UP 状态?
ip link show $interface 2>/dev/null | grep -q "state UP"
if [ $? -ne 0 ]; then continue; fi

/sbin/ifdown ${interface} 完成 ;;

重启)
${0} 停止
sleep 1
${0} 启动
;;

*) echo "用法: ${0} {start|stop|restart}" 退出 1 ;;
esac

退出 0

# 网络配置结束

```

D.17. /etc/rc.d/init.d/sendsignals

```

#!/bin/sh
#####
# 开始发送信号脚本
#
# 描述: 发送信号脚本
#
# 作者: 杰拉德 · 比克曼斯 - gerard@linuxfromscratch.org
# DJ · 卢卡斯 - dj@linuxfromscratch.org
# 更新: 布鲁斯 · 杜布斯 - bdubbs@linuxfromscratch.org
#
# 版本: LFS 7.0
#####

### 初始化信息开始 ###
```

```
# 提供: 发送信号 # 必需启动: # 应启动: # 必需停止: $local_fs swap localnet # 应停止: # 默认启动: # 默认停止: 0 6 # 简短描述: 尝试终止剩余进程。 # 详细描述: 尝试终止剩余进程。 # X-LFS-提供者: LFS ### 结束初始化信息
```

```
. /lib/lsb/init-functions
```

```
case "${1}" in
stop)
omit=$(pidof mdmon) [ -n "$omit" ] && omit="-o
$omit"
```

```
log_info_msg "正在向所有进程发送 TERM 信号..." killall5 -15 $omit
error_value=${?}
```

```
sleep ${KILLDELAY}
```

```
如果 [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then log_success_msg
else log_failure_msg fi
```

```
log_info_msg "正在向所有进程发送 KILL 信号..." killall5 -9 $omit
error_value=${?}
```

```
sleep ${KILLDELAY}
```

```
如果 [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then log_success_msg
else log_failure_msg fi ;
```

```
*) echo "用法: ${0} {stop}" 退出 1 ;;
```

```
esac
```

```
退出 0
```

```
# 发送信号结束
```

D.18. /etc/rc.d/init.d/reboot

```
#!/bin/sh
#####
##### # 开始重启 ## ## 说明: 重启
脚本
```

```

#
# 作者 : 杰拉德·比克曼斯 - gerard@linuxfromscratch.org # DJ·卢卡斯 -
dj@linuxfromscratch.org # 更新 : 布鲁斯·杜布斯 - bdubbs@linuxfromscratch.org # : 皮埃尔·
拉巴斯提 - pierre@linuxfromscratch.org # # 版本 : LFS 7.0 # # 备注 : 2022 年 3 月 24 日更
新: 将"stop"改为"start" # 在 Required-start 中添加$last 功能 #

#####
#####

### 初始化信息开始 ###
# 提供: 重启
# 必需启动: $last # 建议启动: # 必需停止: # 建议停止: # 默
认启动: 6 # 默认停止: # 简短描述: 重启系统 # 详细描述: 重
启系统 # X-LFS-提供者: LFS ### 结束初始化信息

. /lib/lsb/init-functions

case "${1}" in
    启动) 日志信息消息 "正在重启系统..." 重启 -d -f -i
    ;;
    *) echo "用法: ${0} {start}" exit 1 ;;

esac

# 重启结束

```

D.19. /etc/rc.d/init.d/halt

```

#!/bin/sh
#####
##### 停止脚本开始 ## ## 描述
: 系统停止脚本 # # 作者 : Gerard Beekmans - gerard@linuxfromscratch.org # DJ Lucas -
dj@linuxfromscratch.org # 更新 : Bruce Dubbs - bdubbs@linuxfromscratch.org # : Pierre Labastie -
pierre@linuxfromscratch.org # # 版本 : LFS 7.0 # # 备注 : 2022 年 3 月 24 日更新: 将"stop"改
为"start"。 # 在 Required-start 中添加$last 功能 #
#####

```

```
### 初始化信息开始 ###
# 提供: 关机
# 必需启动: $last # 建议启动: # 必需停止: # 建议停止: #
默认启动: 0 # 默认停止: # 简短描述: 停止系统运行。 # 详细描述: 停止系统运行。 # X-LFS-提供者: LFS ### 结束初始化信息
```

```
case "${1}" in
启动)
halt -d -f -i -p
;;
*) echo "用法: {start}" exit 1 ;;
```

```
# 结束 halt
```

D.20. /etc/rc.d/init.d/模板文件

```
#!/bin/sh
#####
# 脚本名称开始 # # 描述
说明: # # 作者信息: # # 版本号: LFS x.x # # 注意事项: #
#####
```

```
### 初始化信息开始 ###
# 提供: template # 必需启动: # 应启动: # 必
需停止: # 应停止: # 默认启动: # 默认停止: #
简短描述: # 详细描述: # X-LFS-提供者: ###
结束初始化信息
```

```
. /lib/lsb/init-functions

case "${1}" in
start) 日志信息消息 "正在启动..." # 如果可以使用
start_daemon
```

```
启动守护进程 完整路径 # 如果无法使用 start_daemon # (启动守护进程的命令不够简单) if !
pidofproc 守护进程名称_如 ps 命令所示 >/dev/null; then
```

```
启动服务的命令 fi evaluate_retval ;;
```

```
stop) log_info_msg "正在停止..." # 如果可以使用 killproc killproc 完整路径 # 如果无法
使用 killproc # (不应通过终止方式停止守护进程) if pidofproc 守护进程名称_如 ps 命令所
示 >/dev/null; then
```

```
停止服务的命令 fi evaluate_retval ;;
```

重启)

```
 ${0} 停止
sleep 1
${0} 启动
;;
```

```
*) echo "用法: ${0} {start|stop|restart}" exit 1 ;;
esac
```

退出 0

脚本结束

D.21. /etc/sysconfig/modules

```
#####
/etc/sysconfig/modules # # 描述 : 模块自动加载配置 # # 作者 : # # 版本 : 00.00 # # 说明 : 本文件
语法如下: # <模块名> [<参数 1> <参数 2> ...] # # 每个模块应独占一行, 需要传递给模块的参数应跟在模
块名后。行分隔符可以是 # 空格或制表符。
#####
```

结束 /etc/sysconfig/modules

D.22. /etc/sysconfig/createfiles

```
#####
```

```

# 开始 /etc/sysconfig/createfiles 配置文件 #
# 描述：创建文件脚本的配置文件 #
# 作者： #
# 版本：00.00 #
# 说明：本文件的语法如下： #
# 如果类型为“file”或“dir” #
# <文件名> <类型> <权限> <用户> <组> #
# 如果类型为“dev” #
# <文件名> <类型> <权限> <用户> <组> <设备类型> #
# <主设备号> <次设备号> #
# #
# <文件名> 是要创建的文件的名称 #
# <类型> 可以是 file、dir 或 dev #
# file 创建一个新文件 #

##### 创建###的新目录#####
# 结束 /etc/sysconfig/createfiles

```

D.23. /etc/sysconfig/udev-retry

```

#####
# 开始 /etc/sysconfig/udev_retry 文件
#
# 描述：udev_retry 脚本配置
#
# 作者：
#
# 版本：00.00
#
# 备注：每个在 mountfs 运行后可能需要重新触发的子系统都应列在此文件中。
# 可能需要列出的子系统包括 rtc（由于/var/lib/hwclock/adjtime 文件）
# 和 sound（由于/var/lib/alsa/asound.state 文件和/usr/sbin/alsactl 命令）。
# 各条目之间用空格分隔。

#####
rtc
# 结束 /etc/sysconfig/udev_retry

```

D.24. /sbin/ifup

```

#!/bin/sh
#####
# 开始 /sbin/ifup
#
# 描述：网络接口启用

```

```
# 作者 : Nathan Coulson - nathan@linuxfromscratch.org # Kevin P. Fleming -
kpfleming@linuxfromscratch.org # 更新 : Bruce Dubbs - bdubbs@linuxfromscratch.org # DJ Lucas -
dj@linuxfromscratch.org # # 版本 : LFS 7.7 # # 备注 : IFCONFIG 变量被传递给/lib/services 目录中的
SERVICE 脚本, # 用于指示服务应引用哪个文件来获取接口规范。 #
```

```
#####
#####
```

```
启动()
{ 记录信息消息 "正在启动${1}接口..."
```

```
if ip link show $1 > /dev/null 2>&1; then
link_status=`ip link show $1`
```

```
如果 [ -n "${link_status}" ]; then if ! echo "${link_status}" |
grep -q UP; then
    ip link set $1 up
fi
fi
```

```
否则 log_failure_msg "接口 ${IFACE} 不存在。" exit 1 fi
```

```
evaluateRetVal
}
```

```
RELEASE="7.7"
```

用法说明: “用法: \$0 [-hV] [--help] [--version] 网络接口” 版本信息=“LFS
ifup, 版本 \${RELEASE}”

```
while [ $# -gt 0 ]; do case
"$1" in
    --help | -h) 帮助=y; break ;;
    --version | -V) echo "${VERSTR}"; exit 0 ;;
    -*)
        echo "ifup: ${1}: 无效选项" >&2 echo "${USAGE}" >&
        2 exit 2 ;;
    *)
        break ;;
esac
```

完成

if [-n "\$help"]; then echo "\${VERSTR}" echo "\${USAGE}" echo cat << HERE_EOF ifup 用于启动
网络接口。接口参数（如 eth0 或 eth0:2）必须与接口配置文件（如
/etc/sysconfig/ifconfig.eth0:2）的后缀部分匹配。

HERE_EOF

```

退出 0
fi

文件=/etc/sysconfig/ifconfig.${1}

# 跳过备份文件 [ "${file}" = "${file}"" ] || exit 0

. /lib/lsb/init-functions

if [ ! -r "${file}" ]; then log_failure_msg "无法启动 ${1} 接口! ${file} 文件缺失或无法访问。" exit 1 fi

.

$file

如果 [ "$IFACE" = "" ]; 那么
log_failure_msg "无法启动 ${1} 接口! ${file} 未定义接口 [IFACE]。"
退出 1
结束

# 如果由启动过程调用且 ONBOOT 未设置为 yes, 则不处理此服务
如果 [ "${IN_BOOT}" = "1" -a "${ONBOOT}" != "yes" ]; 那么
退出 0
结束

# 启动接口
如果 [ "$VIRTINT" != "yes" ]; 那么
启动 ${IFACE}
结束

对于 S 在 ${SERVICE} 中; 执行
如果 [ ! -x "/lib/services/${S}" ]; 那么
MSG="\n 无法处理 ${file}。可能 ${MSG} 服务 '${S}' 不存在 "
MSG="${MSG} 或无法执行。" log_failure_msg "$MSG" exit 1 fi done

#if [ "${SERVICE}" = "wpa" ]; then log_success_msg; fi

# 为服务创建/配置接口 for S in ${SERVICE}; do IFCONFIG=${file}
/lib/services/${S} ${IFACE} up done

# 启用虚拟接口链路 if [ "$VIRTINT" == "yes"
]; then up ${IFACE} fi

# 为所有接口组件启动额外接口: 遍历$INTERFACE_COMPONENTS 中的每
个 I 并执行 up $I

# 按需设置 MTU。检查 MTU 是否为有效值。if test -n "${MTU}"; then if [[ ${MTU}
= ^[0-9]+$ ]] && [[ ${MTU} -ge 68 ]]; then

# 为${IFACE} 和$INTERFACE_COMPONENTS 中的每个接口 I 设置
MTU 值: ip link set dev $I mtu ${MTU}

```

完成
否则 log_info_msg2 "无效的 MTU 值 \$MTU" fi fi

```
# 如果请求设置默认网关路由 if [ -n "${GATEWAY}" ]; then if
ip route | grep -q default; then

log_warning_msg "网关已设置; 跳过。" else log_info_msg "正在为 ${IFACE} 接口添加默认网关 ${GATEWAY}..." ip
route add default via ${GATEWAY} dev ${IFACE} evaluate_retval fi fi

# 结束 /sbin/ifup
```

D.25. /sbin/ifdown

```
#!/bin/bash
#####
# 开始 /sbin/ifdown # # 功
能描述 : 网络接口关闭 # # 作者 : Nathan Coulson - nathan@linuxfromscratch.org # Kevin P. Fleming -
kpfelemin@linuxfromscratch.org # 更新 : Bruce Dubbs - bdubbs@linuxfromscratch.org # # 版本 : LFS 7.0
# # 注意事项 : IFCONFIG 变量会传递给 /lib/services 目录下的脚本, # 用于指定服务应读取哪个文件来获取
接口配置信息 #

#####
RELEASE="7.0"

USAGE="用法: $0 [ -hV ] [--help] [--version] 网络接口" VERSTR="LFS ifdown, 版
本 ${RELEASE}"

当 [ $# -gt 0 ]; do case "$1"
in
    --help | -h) help="y"; break ;;
    --version | -V) echo "${VERSTR}"; exit 0 ;;
    -*)
        echo "ifup: ${1}: 无效选项" >&2 echo "${USAGE}" >&
        2 exit 2 ;;

    *)
        终止 ;;
esac
完成

if [ -n "$help" ]; then echo
"${VERSTR}" echo "${USAGE}" echo
```

cat << HERE_EOF echo ifdown 用于关闭网络接口。接口参数（如 eth0 或 eth0:2）必须与接口配置文件（如 /etc/sysconfig/ifconfig.eth0:2）的后缀部分匹配。

```

HERE_EOF
    退出 0
fi

文件=/etc/sysconfig/ifconfig.${1}

# 跳过备份文件 [ "${文件}" = "${文件}~" ] || 退出 0

. /lib/lsb/init-functions

如果 [ ! -r "${file}" ]; 那么
log_warning_msg "${file} 文件缺失或无法访问。"
exit 1
fi

. ${file}

如果 [ "$IFACE" = "" ]; 那么
log_failure_msg "${file} 文件未定义网络接口 [IFACE]。"
exit 1
fi

# 我们只需要第一个服务来关闭接口
S=`echo ${SERVICE} | cut -f1 -d" "`

如果 ip link show ${IFACE} > /dev/null 2>&1; 那么 if [ -n "${S}" -a -x
"/lib/services/${S}" ]; 则
IFCONFIG=${file} /lib/services/${S} ${IFACE} down 否则 MSG="无法处理 ${file}。可能
是 " MSG="${MSG} SERVICE 变量未设置 " MSG="${MSG} 或指定的服务无法执行。"
log_failure_msg "$MSG" exit 1 fi 否则 log_warning_msg "接口 ${1} 不存在。" fi

# 如果设备中存在其他接口, 则保持该接口启用状态 link_status=`ip link show ${IFACE} 2>/dev/null` 

if [ -n "${link_status}" ]; then if [ "$(echo ${link_status} | grep UP)" !=
"" ]; then
如果 [ "$(ip addr show ${IFACE} | grep 'inet ')" == "" ]; 那么 log_info_msg "正在关闭
${IFACE} 网络接口..." ip link set ${IFACE} down evaluate_retval fi fi fi

# 结束 /sbin/ifdown

```

D.26. /lib/services/ipv4-static

```
#!/bin/sh
#####
# /lib/services/ipv4-static # # 描述：IPv4 静态启动脚本 # # 作者：Nathan Coulson -
# nathan@linuxfromscratch.org # Kevin P. Fleming - kpfleming@linuxfromscratch.org # 更新：Bruce
# Dubbs - bdubbs@linuxfromscratch.org # # 版本：LFS 7.0 #
#####

. /lib/lsb/init-functions .
${IFCONFIG}

if [ -z "${IP}" ]; then log_failure_msg "\n${IFCONFIG} 中缺少 IP 变量，无法继续。" exit 1 fi
if [ -z "${PREFIX}" -a -z "${PEER}" ]; then log_warning_msg "\nPREFIX 变量在${IFCONFIG} 中缺失，默认设为
24。" PREFIX=24 args="${args} ${IP}/${PREFIX}"

elif [ -n "${PREFIX}" -a -n "${PEER}" ]; then log_failure_msg "\n${IFCONFIG} 中同时指定了 PREFIX 和 PEER 参数，无法继
续执行。" exit 1

elif [ -n "${PREFIX}" ]; then args="${args}
${IP}/${PREFIX}"
else
    if [ -n "${PEER}" ]; then args="${args} ${IP}
peer ${PEER}" fi
fi

如果 [ -n "${LABEL}" ]; 那么 args="${args}
标签 ${LABEL}" 结束

如果 [ -n "${BROADCAST}" ]; 那么 args="${args} 广播
${BROADCAST}" 结束

情况 "${2}" 于
启动) 如果 [ "$(ip 地址 显示 ${1} 2>/dev/null | 查找 ${IP})" = "" ]; 那么
    log_info_msg "正在为 ${1} 接口添加 IPv4 地址 ${IP}..." ip addr add ${args} dev ${1} evaluate_retval else
    log_warning_msg "无法为 ${1} 添加 IPv4 地址 ${IP}，该地址已存在。" fi ;;

down) if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP})" != "" ]; then
    log_info_msg "正在从 ${1} 接口移除 IPv4 地址 ${IP}..."
```

```

ip addr del ${args} dev ${1}
evaluate_retval fi

if [ -n "${GATEWAY}" ]; then # 仅当不存在剩余 IPv4 地址时才移除网关 if [ "$(ip addr show ${1}
2>/dev/null | grep 'inet')" != "" ]; then

log_info_msg "正在移除默认网关..." ip route del default
evaluate_retval fi fi ;;

*) echo "用法: ${0} [接口名] {up|down}" exit 1 ;;
esac

# 结束 /lib/services/ipv4-static

```

D.27. /lib/services/ipv4-static-route

```

#!/bin/sh
#####
## 开始
/lib/services/ipv4-static-route ## 描述 : IPv4 静态路由脚本 ## 作者 : Kevin P. Fleming -
kpfleming@linuxfromscratch.org ## DJ Lucas - dj@linuxfromscratch.org ## 更新 : Bruce Dubbs -
bdubbs@linuxfromscratch.org ## 版本 : LFS 7.0 ##
#####

. /lib/lsb/init-functions .
${IFCONFIG}

case "${TYPE}" in
("") | "网络")
    need_ip=1
    need_gateway=1
;;
("默认")
    需要网关=1 参数="${参数} 默认"
    描述="默认" ;;
("主机")
    需要 IP=1
;;
("不可达")
    需要 IP=1 参数="${参数} 不可达" 描述
    ="不可达" ;;
esac

```

```
(*) log_failure_msg "未知路由类型 (${TYPE}) 出现在 ${IFCONFIG} 中，无法继续执行。" exit 1 ;;
```

```
; ; if [ -n "${GATEWAY}" ]; then MSG="静态路由配置中 ${IFCONFIG} 文件不允许设置 GATEWAY 变量。\\n" log_failure_msg "$MSG 请仅使用 STATIC_GATEWAY 参数，无法继续执行" exit 1 fi
```

```
if [ -n "${need_ip}" ]; then if [ -z "${IP}" ]; then log_failure_msg "${IFCONFIG} 文件中缺少 IP 变量配置，无法继续执行。" exit 1 fi
```

如果 [-z "\${PREFIX}"]; 那么 log_failure_msg "\${IFCONFIG} 中缺少 PREFIX 变量，无法继续。" exit 1 fi

```
args="${args} ${IP}/${PREFIX}"
desc="${desc}${IP}/${PREFIX}" fi
```

```
if [ -n "${need_gateway}" ]; then if [ -z "${STATIC_GATEWAY}" ]; then log_failure_msg "${IFCONFIG} 中缺少 STATIC_GATEWAY 变量，无法继续。" exit 1 fi args="${args} via ${STATIC_GATEWAY}" fi
```

```
if [ -n "${SOURCE}" ]; then
    args="${args} src ${SOURCE}"
fi
```

情况 "\${2}" 于
up) log_info_msg "正在将'\${desc}'路由添加到\${1}接口..." ip route add \${args} dev \${1}
evaluate_retval ;;

down) log_info_msg "正在从\${1}接口移除'\${desc}'路由..." ip route del \${args} dev \${1}
evaluate_retval ;;

```
*) echo "用法: ${0} [接口] {启用|禁用}" exit 1 ;; esac
```

```
# 结束 /lib/services/ipv4-static-route
```

附录 E. Udev 配置规则

本附录所列规则仅为方便查阅。实际安装通常按照第 8.76 节"Systemd-257.3 中的 Udev"的说明进行。

E.1. 55-lfs.rules

```
# /etc/udev/rules.d/55-lfs.rules: LFS 系统专用规则定义

# 核心内核设备

# 当/dev/rtc 可用时立即设置系统时钟。SUBSYSTEM=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock
start" KERNEL=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"
```

附录 F. LFS 许可证

本书采用知识共享署名-非商业性使用-相同方式共享 2.0 许可协议进行授权。

书中涉及的计算机指令可依据 MIT 许可证提取使用。

F.1. 知识共享许可协议

知识共享法律文本

署名-非商业性使用-相同方式共享 2.0

重要提示

创意共享组织（CREATIVE COMMONS CORPORATION）并非律师事务所，不提供法律服务。本许可协议的发布不构成律师-客户委托关系。创意共享组织以"现状"基础提供本信息，对所提供的信息不作任何担保，并对因使用该信息导致的损害不承担任何责任。

许可协议

本作品（定义如下）依据《创意共享公共许可协议》（简称"CCPL"或"本协议"）条款提供。本作品受著作权法和/或其他相关法律保护。任何未经本许可协议或著作权法授权的作品使用行为均被禁止。

通过行使本作品授予的任何权利，即表示您接受并同意受本许可条款约束。许可方授予您此处包含的权利，以换取您接受这些条款和条件。

1. 定义

a. "集体作品"指将本作品未经修改的完整内容与其他若干独立创作的贡献作品（这些作品本身构成独立作品）汇编而成的整体作品，例如期刊、选集或百科全书。在本许可条款下，构成集体作品的作品不被视为衍生作品（定义见下文）。

b. "衍生作品"指基于本作品或本作品与其他既有作品创作的作品，包括翻译、乐曲改编、戏剧化改编、小说化改编、电影改编、录音、艺术复制、节略、缩编等任何对本作品进行重构、转换或改编的形式；但构成集体作品的作品不被视为本许可条款下的衍生作品。为避免歧义，若本作品为音乐作品或录音制品，将其与动态画面进行时间同步处理（"同步配乐"）的行为在本许可条款下视为创作衍生作品。

c. "许可方"指依据本许可条款提供作品的个人或实体。

d. "原作者"指创作作品的个人或实体。

e. "作品"指依据本许可条款提供的可受版权保护的原创著作。f. "您"指依据本许可行使权利且未曾就作品违反过本许可条款的个人或实体，或虽曾违反但已获得许可方明确授权继续依据本许可行使权利的个人或实体。

- g. "许可要素"指许可方在本许可标题中选定的下列高级许可属性：署名、非商业性使用、相同方式共享。
2. 合理使用权。本许可条款无意削减、限制或约束任何基于版权法或其他适用法律中合理使用、首次销售或其他对版权所有者专有权利限制条款所产生的权利。
 3. 许可授予。在遵守本许可条款及条件的前提下，许可方特此授予您一项全球性、免版税、非排他性、永久性（在适用版权存续期间内）的许可，以行使下述作品权利：
 - a. 复制作品、将作品纳入一个或多个汇编作品，以及复制被纳入汇编作品中的作品；
 - b. 创作并复制衍生作品；
 - c. 通过分发复印件或录音制品、公开展示、公开表演以及通过数字音频传输方式公开表演作品（包括被纳入汇编作品中的作品）；
 - d. 通过分发复印件或录音制品、公开展示、公开表演以及通过数字音频传输方式公开表演衍生作品；

上述权利可在现有及未来开发的所有媒介和格式中行使，包括为适应其他媒介和格式所需的技术性修改权利。许可方未明确授予的所有权利均予保留，包括但不限于第 4(e)条和第 4(f)条所列权利。
 4. 限制条款。上述第 3 节授予的许可明确受到以下限制性条款的约束：
 - a. 您仅能依照本许可条款对作品进行分发、公开展示、公开表演或公开数字表演，且必须在您分发、公开展示、公开表演或公开数字表演的每份作品副本或录音制品中附带本许可协议的副本或其统一资源标识符。您不得对作品附加任何修改或限制本许可条款、或限制被许可人行使本许可所授权利的条款。您不得对作品进行分许可。您必须保留作品中所有涉及本许可及免责声明的完整声明。您不得以任何技术措施对作品进行分发、公开展示、公开表演或公开数字表演，若该技术措施以与本许可协议条款相冲突的方式控制对作品的访问或使用。上述规定同样适用于被收录至汇编作品中的情况，但除作品本身外，并不要求该汇编作品整体受本许可条款约束。若您创建汇编作品，在收到任何许可方通知后，您应在可行范围内按要求从汇编作品中删除对该许可方或原作者的引用。若您创作演绎作品，在收到任何许可方通知后，您应在可行范围内按要求从演绎作品中删除对该许可方或原作者的引用。
 - b. 您仅能依据本许可协议条款、具有相同许可要素的后续版本许可协议，或包含与本许可协议相同许可要素的创作共用 iCommons 许可协议（例如署名-非商业性使用-相同方式共享 2.0 日本版）来分发、公开展示、公开表演或公开数字表演演绎作品。在您分发、公开展示、公开表演或公开数字表演的每份演绎作品副本或录音制品中，必须包含本许可协议或前文所述其他许可协议的副本或其统一资源标识符。您不得对演绎作品附加任何改变或限制本许可条款或受让人行使本许可所授权利的条件，且必须完整保留所有涉及本许可及免责声明的声明。您不得以任何技术措施分发、公开展示、公开表演或公开数字表演演绎作品，这些技术措施以控制作品访问或使用的方式

与本许可协议的条款不一致。上述规定适用于作为集体作品组成部分的衍生作品，但这并不要求除衍生作品本身之外的集体作品也受本许可条款的约束。

或以任何技术措施公开数字表演衍生作品，这些技术措施以控制作品访问或使用的方式 c. 您不得以主要意图或直接面向商业利益或私人金钱报酬的任何方式行使第 3 节授予您的任何权利。通过数字文件共享或其他方式将作品与其他受版权保护的作品进行交换，只要不涉及与受版权保护作品交换相关的任何金钱报酬，则不应被视为主要意图或直接面向商业利益或私人金钱报酬。

d. 若您分发、公开展示、公开表演或公开数字表演本作品或其衍生作品或集体作品，必须完整保留作品的所有版权声明，并根据您所采用的媒介或方式合理注明原作者署名——若提供原作者姓名（或笔名）则予以标注；若提供作品标题则予以标注；在合理可行的范围内标注许可方指定的与作品相关联的统一资源标识符（如有），除非该标识符未指向作品的版权声明或许可信息；对于衍生作品，须标注说明作品在衍生作品中的使用情况（例如“原作品的法语翻译，原作者：XXX”，或“剧本改编自原作品，原作者：XXX”）。此类署名可以任何合理方式实现；但对于衍生作品或集体作品，该署名至少应出现在其他类似作者署名出现的位置，并以至少同等显著的方式呈现。

e. 为避免疑义，若作品为音乐作品：

i. 一揽子许可下的表演版税。许可方保留独家收取作品公开表演或公开数字表演（如网络广播）版税的权利，无论是个别收取还是通过表演权协会（如 ASCAP、BMI、SESAC）收取，前提是该表演主要旨在或面向商业利益或私人金钱报酬。

ii. 机械复制权与法定版税。许可方保留独家收取版税的权利（无论通过个人、音乐版权代理机构或指定代理人[如 Harry Fox Agency]收取），针对您基于本作品制作的任何留声机录音（即“翻唱版本”）并进行发行的行为，但须遵守美国版权法第 17 编第 115 条（或其他司法管辖区的等效条款）规定的强制许可条款——前提是您发行此类翻唱版本的主要目的是为了获取商业利益或私人金钱报酬。6. 网络传播权与法定版税。为避免歧义，当本作品为录音制品时，许可方保留独家收取版税的权利（无论通过个人、表演权协会[如 SoundExchange]收取），针对本作品的公开数字表演（如网络广播）行为，但须遵守美国版权法第 17 编第 114 条（或其他司法管辖区的等效条款）规定的强制许可条款——前提是您的公开数字表演行为主要目的是为了获取商业利益或私人金钱报酬。

f. 网络传播权与法定版税。为避免疑义，若作品为录音制品，许可方保留独家收取作品公开数字表演（如网络广播）版税的权利（无论单独收取或通过表演权协会如 SoundExchange 收取），该权利受限于美国版权法第 17 编第 114 条（或其他司法管辖区的等效条款）创设的强制许可，前提是您的公开数字表演主要旨在或面向商业利益或私人金钱补偿。

5. 声明、保证与免责条款

免责声明 除非各方另有书面协议，许可方按“现状”提供作品，且不对作品作任何明示或默示的声明或保证，包括但不限于对所有权、适销性、特定用途适用性、不侵权、无潜在或其他缺陷、准确性、是否存在错误的保证，无论这些错误是否可被发现。部分司法管辖区不允许排除默示保证，因此此类排除条款可能对您不适用。

6. 责任限制 除非适用法律另有要求，在任何情况下，许可方均不对您因本许可或使用作品而产生的任何特殊、附带、后果性、惩罚性或惩戒性损害承担法律责任，即使许可方已被告知发生此类损害的可能性。

7. 终止

a. 本许可及依此授予的权利将在您违反本许可条款时自动终止。然而，根据本许可从您处获得衍生作品或集体作品的个人或实体，只要其持续完全遵守相关许可条款，其许可将不会被终止。本许可终止后，第 1、2、5、6、7 和 8 条条款仍然有效。

b. 在遵守上述条款和条件的前提下，此处授予的许可是永久性的（适用于作品版权有效期内）。尽管有上述规定，许可方保留以不同许可条款发布作品或随时停止分发作品的权利；但此类选择不得撤回本许可（或根据本许可条款已授予或必须授予的任何其他许可），且本许可将持续完全有效，除非按上述规定终止。

8. 其他条款

a. 每当您发行或以数字方式公开表演本作品或集体作品时，许可方即按本许可授予您的相同条款向接收者提供本作品的许可。

b. 每当您发行或以数字方式公开表演演绎作品时，许可方即按本许可授予您的相同条款向接收者提供原作品的许可。

c. 若本许可任何条款根据适用法律无效或不可执行，该条款不影响本许可其余条款的有效性或可执行性；未经本协议各方进一步行动，该条款应在最小必要范围内进行修改以使其有效且可执行。

d. 除非以书面形式并由被要求放弃或同意违约的一方签署，否则本许可的任何条款或规定均不得被视为放弃，任何违约亦不得被视为获得同意。

e. 本许可构成双方就此处授权作品达成的完整协议。对于本协议未明确规定的作品相关事项，不存在任何谅解、协议或声明。许可方不受您任何通讯中可能出现的附加条款约束。未经许可方与您双方书面同意，不得修改本许可协议。

重要提示

知识共享组织（Creative Commons）并非本许可协议的缔约方，对本作品不作任何形式的担保。无论基于何种法律理论，知识共享组织均不对您或任何一方因本许可协议导致的任何损害承担责任，包括但不限于任何直接、间接、偶然或结果性损害。尽管有前述两项条款规定，若知识共享组织已明确声明自身为本许可协议的授权方，则其应享有授权方的全部权利并承担相应义务。

除向公众表明本作品采用 CCPL 许可这一有限目的外，任何一方未经知识共享组织事先书面同意，不得使用“知识共享”商标或任何相关商标标识。所有获准使用行为均须遵守知识共享组织现行商标使用准则，该准则可能发布于其官方网站或根据要求不定期提供。

可通过 <http://creativecommons.org/> 联系知识共享组织。

F.2. MIT 许可证

版权所有 © 1999-2025 Gerard Beekmans

特此授权，允许任何获得本软件及相关文档文件（以下简称“软件”）副本的个人，在不受限制的情况下处理该软件，包括但不限于使用、复制、修改、合并、发布、分发、再授权及/或销售软件副本的权利，并允许被提供软件的个人行使上述权利，但须遵守以下条件：

上述版权声明和本许可声明须包含在软件的所有副本或主要部分中。

本软件按“原样”提供，不附带任何明示或暗示的担保，包括但不限于对适销性、特定用途适用性和非侵权性的担保。在任何情况下，作者或版权持有人均不对因软件或使用或其他交易行为引起的任何索赔、损害或其他责任负责，无论是合同诉讼、侵权诉讼还是其他诉讼。

目录

软件包列表

Acl: 135
Attr: 134
Autoconf: 172
Automake: 173
Bash: 158
 工具: 62
Bash: 158
 工具链: 62
Bc: 119
Binutils: 127
 工具链, 第一遍: 47
 工具链, 第二遍: 75
二进制工具集: 127
 工具链, 第一遍: 47
 工具链, 第二遍: 75
Binutils: 127
 工具链, 第一遍: 47
 工具链, 第二遍: 75
Bison: 156
 工具: 85
Bison: 156
 工具: 85
启动脚本: 239
 用法: 249
启动脚本: 239
 用法: 249
Bzip2: 109
检查: 192
Coreutils: 187
 工具: 63
Coreutils: 187
 工具集: 63
DejaGNU: 125
Diffutils: 193
 工具: 64
Diffutils: 193
 工具: 64
E2fsprogs: 230
Expat: 163
Expect: 123
File: 115
 工具: 65
文件: 115

 工具: 65
 查找工具: 195
 工具: 66
 查找工具集: 195
 工具集: 66
Flex 工具: 120
Flit 核心库: 181
Gawk: 194
 工具: 67
Gawk: 194
 工具: 67
GCC: 143 工具,
libstdc++第一阶段: 56 工
具, 第一阶段: 49 工具, 第
二阶段: 76
GCC: 143 工具,
libstdc++第一阶段: 56 工
具, 第一阶段: 49 工具, 第
二阶段: 76
GCC: 143 工具,
libstdc++第一阶段: 56 工
具, 第一阶段: 49 工具, 第
二阶段: 76
GCC: 143 工具,
libstdc++第一阶段: 56 工
具, 第一阶段: 49 工具, 第
二阶段: 76
GDBM: 161
Gettext: 154 工具: 84
Gettext: 154 工具: 84
Glibc: 100 工具: 53
Glibc: 100 工具: 53
GMP: 130
Gperf: 162
Grep: 157 工具: 68
Grep: 157 工具: 68
Groff: 196
GRUB: 199
Gzip: 201 工具: 69
Gzip: 201

工具集: 69
 IANA 协议集: 99
 网络工具集: 164
 国际化工具: 171
 IPRoute2: 202
 Jinja2: 216
 Kbd: 204
 Kmod: 186
 Less: 166
 Libcap: 136
 Libelf: 176
 libffi: 177
 Libpipeline: 206
 Libtool: 160
 Libxcrypt: 137
 Linux: 264
 工具, API 头文件: 52
 Linux: 264
 工具, API 头文件: 52
 Lz4: 113
 M4: 118
 工具: 59
 M4: 118
 工具: 59
 Make: 207
 工具: 70
 Make: 207
 工具: 70
 Man-DB: 220
 Man-pages: 98
 MarkupSafe: 215
 Meson: 185
 MPC: 133
 MPFR: 132
 Ncurses: 149
 工具: 60
 Ncurses: 149
 工具: 60
 Ninja: 184
 OpenSSL: 174
 Patch: 208
 工具: 71
 Patch: 208
 工具: 71
 Perl: 167
 工具: 86
 Perl: 167

工具: 86
 Pkgconf: 126
 Procps-ng: 223
 Psmisc: 153
 Python: 178
 临时: 87
 Python: 178
 临时文件: 87
 rc.site: 255
 Readline: 116
 Sed: 152
 工具: 72
 Sed: 152
 工具: 72
 Setuptools: 183
 影子密码: 139
 配置: 140
 影子密码: 139
 配置: 140
 Sysklogd: 233
 配置中: 233
 Sysklogd: 233
 配置中: 233
 SysVinit: 234
 配置中: 250
 SysVinit: 234
 配置中: 250
 Tar: 209
 工具: 73
 Tar: 209
 工具: 73
 Tcl: 121
 Texinfo: 210
 临时: 88
 Texinfo: 210
 临时: 88
 Udev: 217
 配置: 219
 用法: 241
 Udev: 217
 配置: 219
 用法: 241
 Udev: 217
 配置: 219
 用法: 241
 Util-linux 工具集: 225
 工具: 89

Util-linux: 225

工具集: 89

Vim: 212

wheel: 182

XML::解析器: 170

Xz 压缩工具: 111

工具集: 74

Xz 压缩工具: 111

工具: 74

Zlib 压缩库: 108

zstd 压缩工具: 114

程序

[: 187, 188

2to3: 178

accessdb: 220, 221

aclocal: 173, 173

aclocal-1.17: 173, 173

addftinfo: 196, 196

addpart: 225, 226

addr2line: 127, 128

afmtodit: 196, 196

agetty: 225, 226

apropos: 220, 222 ar:

127, 128 as: 127, 128

属性: 134, 134

autoconf: 172, 172

autoheader: 172, 172

autom4te: 172, 172

automake: 173, 173

automake-1.17: 173, 173

autopoint: 154, 154

autoreconf: 172, 172

autoscan: 172, 172

autoupdate: 172, 172

awk: 194, 194 b2sum:

187, 188 badblocks: 230,

231 base64: 187, 188, 187,

188 base64: 187, 188, 187,

188 basename: 187, 188

basenc: 187, 188

bash: 158, 159

bashbug: 158, 159

bc: 119, 119

bison: 156, 156

blkdiscard: 225, 226

blkid: 225, 226

blkzone: 225, 226

blockdev: 225, 226

bomtool: 126, 126

bootlogd: 234, 234

bridge: 202, 202

bunzip2: 109, 110

bzcat: 109, 110

bzcmp: 109, 110

bzdiff: 109, 110

bzegrep: 109, 110

bzfgrep: 109, 110

bzgrep: 109, 110

bzip2: 109, 110

bzip2recover: 109,

110 bzless: 109, 110

bzmore: 109, 110

c++: 143, 147

c++filt: 127, 128 cal:

225, 226 capsh: 136,

136 captainfo: 149,

150 cat: 187, 188

catman: 220, 222

cc: 143, 147 cfdisk:

225, 226 chacl: 135,

135 chage: 139, 141

chattr: 230, 231

chcon: 187, 188

chcpu: 225, 226

checkmk: 192, 192

chem: 196, 196

chfn: 139, 141

chgpasswd: 139, 141

chgrp: 187, 188

chmem: 225, 226

chmod: 187, 188

choom: 225, 226

chown: 187, 189

chpasswd: 139, 141

chroot: 187, 189

chrt: 225, 226

chsh: 139, 141

chvt: 204, 205

cksum: 187, 189

156 清除: 149, 150
 比较: 193, 193
 列: 225, 226
 列转换: 225, 226
 colrm: 225, 226
 column: 225, 226
 comm: 187, 189
 compile_et: 230,
 231 corelist: 167,
 168 cp: 187, 189
 cpan: 167, 168
 cpp: 143, 147
 csplit: 187, 189
 ctrlaltdel: 225, 226
 ctstat: 202, 202
 cut: 187, 189
 c_rehash: 174, 175
 date: 187, 189 dc:
 119, 119 dd: 187,
 189 deallocvt: 204,
 205 debugfs: 230,
 231 dejagnu: 125,
 125 delpart: 225,
 227 depmod: 186,
 186 df: 187, 189

 diff: 193, 193
 diff3: 193, 193
 dir: 187, 189
 dircolors: 187, 189
 dirname: 187, 189
 dmesg: 225, 227
 dnsdomainname: 164,
 165 du: 187, 189
 dumpe2fs: 230, 231
 dumpkeys: 204, 205
 e2freefrag: 230, 231
 e2fsck: 230, 231
 e2image: 230, 231
 e2label: 230, 231
 e2mmpstatus: 230, 231
 e2scrub: 230, 231
 e2scrub_all: 230, 231
 e2undo: 230, 231
 e4crypt: 230, 231
 e4defrag: 230, 231
 echo: 187, 189

 egrep: 157, 157
 eject: 225, 227
 elfedit: 127, 128
 enc2xs: 167, 168
 encguess: 167, 168
 env: 187, 189
 envsubst: 154, 154
 eqn: 196, 196
 eqn2graph: 196, 196
 ex: 212, 214
 expand: 187, 189
 预期值: 123, 124
 有效期: 139, 141
 表达式: 187, 189
 因数: 187, 189
 faillog: 139, 141
 fallocate: 225, 227
 false: 187, 189
 fdisk: 225, 227
 fgconsole: 204, 205
 fgrep: 157, 157
 file: 115, 115
 filefrag: 230, 232
 fincore: 225, 227
 find: 195, 195
 findfs: 225, 227
 findmnt: 225, 227
 flex: 120, 120
 flex++: 120, 120
 flock: 225, 227
 fmt: 187, 189
 fold: 187, 189
 free: 223, 223
 fsck: 225, 227
 fsck.cramfs: 225, 227
 fsck.ext2: 230, 232
 fsck.ext3: 230, 232
 fsck.ext4: 230, 232
 fsck.minix: 225, 227
 fsfreeze: 225, 227
 fstab-decode: 234,
 234 fstrim: 225, 227
 ftp: 164, 165
 fuser: 153, 153
 g++: 143, 147
 gawk: 194, 194
 gawk-5.3.1: 194,
 194

194 gcc: 143, 147
 gc-ar: 143, 147
 gc-nm: 143, 147
 gc-ranlib: 143, 147
 gcov: 143, 147
 gcov-dump: 143, 147
 gcov-tool: 143, 147
 gdbmtool: 161, 161
 gdbm_dump: 161, 161
 gdbm_load: 161, 161
 gdiffmk: 196, 196
 gencat: 100, 106
 genl: 202, 202
 getcap: 136, 136
 getconf: 100, 106
 getent: 100, 106
 getfacl: 135, 135
 getattr: 134, 134
 getkeycodes: 204,
 205 getopt: 225, 227
 getpcaps: 136, 136
 getsubids: 139, 141
 gettext: 154, 154
 gettext.sh: 154, 154
 gettextize: 154, 154
 glilypond: 196, 196
 gpasswd: 139, 141
 gperf: 162, 162
 gperl: 196, 196
 gpinyin: 196, 196
 gprof: 127, 128
 gprofng: 127, 128
 grap2graph: 196,
 197 grep: 157, 157
 grn: 196, 197
 grodvi: 196, 197
 groff: 196, 197
 groffer: 196, 197
 grog: 196, 197
 grolbp: 196, 197
 grolj4: 196, 197
 gropdf: 196, 197
 grops: 196, 197
 grotty: 196, 197
 groupadd: 139, 141
 groupdel: 139, 141
 groupmems: 139,
 141

groupmod: 139, 141
 groups: 187, 189
 grpck: 139, 141
 grpconv: 139, 141
 grpunconv: 139, 141 grub-bios-
 setup: 199, 200 grub-editenv:
 199, 200 grub-file: 199, 200
 grub-fstest: 199, 200 grub-
 glue-efi: 199, 200 grub-
 install: 199, 200 grub-
 kbdcomp: 199, 200 grub-
 macbless: 199, 200 grub-
 menulst2cfg: 199, 200 grub-
 mkconfig: 199, 200 grub-
 mkimage: 199, 200 grub-
 mklayout: 199, 200 grub-
 mknetdir: 199, 200 grub-
 mkpasswd-pbkdf2: 199, 200
 grub-mkrelpath: 199, 200
 grub-mkrescue: 199, 200 grub-
 mkstandalone: 199, 200 grub-
 ofpathname: 199, 200 grub-
 probe: 199, 200 grub-reboot:
 199, 200 grub-render-label:
 199, 200 grub-script-check:
 199, 200 grub-set-default: 199,
 200 grub-setup: 199, 200 grub-
 syslinux2cfg: 199, 200
 gunzip: 201, 201 gzexe: 201,
 201 gzip: 201, 201 h2ph: 167,
 168 h2xs: 167, 168 halt: 234,
 234 hardlink: 225, 227 head:
 187, 189 hexdump: 225, 227
 hostid: 187, 189 hostname:
 164, 165 hpftodit: 196, 197
 hwclock: 225, 227 i386: 225,
 227 iconv: 100, 106
 iconvconfig: 100, 106 id: 187,
 189

189 idle3: 178
 ifconfig: 164, 165
 ifnames: 172, 172
 ifstat: 202, 202
 idxbib: 196, 197
 info: 210, 210
 infocmp: 149, 150
 infotocap: 149, 150
 init: 234, 234
 insmod: 186, 186
 install: 187, 189 install-
 info: 210, 211
 instmodsh: 167, 168
 intltool-extract: 171,
 171 intltool-merge: 171,
 171 intltool-prepare:
 171, 171 intltool-
 update: 171, 171
 intltoolize: 171, 171
 ionice: 225, 227 ip: 202,
 202 ipcmk: 225, 227
 ipcrm: 225, 227 ipcs:
 225, 227 irqtop: 225, 227
 isosize: 225, 227 join:
 187, 189 json_pp: 167,
 168 kbdinfo: 204, 205
 kbdrate: 204, 205
 kbd_mode: 204, 205
 kill: 225, 227 killall: 153,
 153 killall5: 234, 234
 kmod: 186, 186 last:
 225, 227 lastb: 225, 227
 ld: 127, 128 ld.bfd: 127,
 128 ldattach: 225, 227
 ldconfig: 100, 106 ldd:
 100, 106 lddlibc4: 100,
 106 less: 166, 166
 lessecho: 166, 166
 lesskey: 166, 166 lex:
 120, 120 lexgrog: 220,
 222
 lfskernel-6.13.4: 264,
 269 libasan: 143, 147
 libatomic: 143, 147
 libccl: 143, 147
 libnetcfg: 167, 168
 libtool: 160, 160
 libtoolize: 160, 160 link:
 187, 189 linux32: 225,
 227 linux64: 225, 227
 lkbib: 196, 197
 ln: 187, 189
 lnstat: 202, 203
 loadkeys: 204, 205
 loadunimap: 204,
 205 locale: 100, 106
 localedef: 100, 106
 locate: 195, 195
 logger: 225, 227
 login: 139, 142
 logname: 187, 189
 logoutd: 139, 142
 logsave: 230, 232
 look: 225, 227
 lookbib: 196, 197
 losetup: 225, 227
 ls: 187, 189
 lsattr: 230, 232
 lsblk: 225, 228
 lscpu: 225, 228
 lsfd: 225, 228
 lsipc: 225, 228
 lsirq: 225, 228
 lslocks: 225, 228
 lslogins: 225, 228
 lsmem: 225, 228
 lsmod: 186, 186
 lsns: 225, 228
 lto-dump: 143, 147
 lz4: 113, 113
 lz4c: 113, 113
 lz4cat: 113, 113
 lzcat: 111, 111
 lzcmp: 111, 111
 lzdiff: 111, 111
 lzegrep: 111, 111
 lzfgrep: 111, 111

111 lzgrep: 111, 111
 lzless: 111, 111
 lzma: 111, 111
 lzmadec: 111, 111
 lzmainfo: 111, 111
 lzmore: 111, 112 m4:
 118, 118 make: 207,
 207 makedb: 100,
 106 makeinfo: 210,
 211 man: 220, 222
 man-recode: 220,
 222 mandb: 220, 222
 manpath: 220, 222
 mapscrn: 204, 205
 mcookie: 225, 228
 md5sum: 187, 189
 mesg: 225, 228

 meson: 185, 185
 mkdir: 187, 190
 mke2fs: 230, 232
 mkfifo: 187, 190
 mkfs: 225, 228
 mkfs.bfs: 225, 228
 mkfs.cramfs: 225, 228
 mkfs.ext2: 230, 232
 mkfs.ext3: 230, 232
 mkfs.ext4: 230, 232
 mkfs.minix: 225, 228
 mklost+found: 230,
 232 mknod: 187, 190
 mkswap: 225, 228
 mktemp: 187, 190
 mk_cmds: 230, 232
 mmroff: 196, 197
 modinfo: 186, 186
 modprobe: 186, 186
 more: 225, 228
 mount: 225, 228
 mountpoint: 225,
 228 msgattrib: 154,
 154 msgcat: 154,
 154 msgcmp: 154,
 154 msgcomm: 154,
 155 msgconv: 154,
 155 msgen: 154, 155
 msgexec: 154, 155

 msgfilter: 154, 155
 msgfmt: 154, 155
 msggrep: 154, 155
 msginit: 154, 155
 msgmerge: 154, 155
 msgunfmt: 154, 155
 msguniq: 154, 155
 mtrace: 100, 106
 mv: 187, 190
 namei: 225, 228
 ncursesw6-config: 149,
 150
 neqn: 196, 197
 newgidmap: 139, 142
 newgrp: 139, 142
 newuidmap: 139, 142
 newusers: 139, 142
 ngettext: 154, 155
 nice: 187, 190
 ninja: 184, 184
 nl: 187, 190
 nm: 127, 128
 nohup: 187, 190
 nologin: 139, 142
 nproc: 187, 190
 nroff: 196, 197
 nsenter: 225, 228
 nstat: 202, 203
 numfmt: 187, 190
 objcopy: 127, 128
 objdump: 127, 128
 od: 187, 190
 openssl: 174, 175
 openvt: 204, 205
 partx: 225, 228
 passwd: 139, 142
 paste: 187, 190
 patch: 208, 208
 pathchk: 187, 190
 pcprofiledump: 100, 106
 pdfmom: 196, 197
 pdfroff: 196, 197
 pdftexi2dvi: 210, 211
 peekfd: 153, 153
 perl: 167, 168
 perl5.40.1: 167, 168
 perlbug: 167, 168
 perldoc: 167, 168

168 perlvp: 167,
 168 perlthanks:
 167, 168 pfbtops:
 196, 197 pgrep:
 223, 223 pic: 196,
 197 pic2graph: 196,
 197 piconv: 167,
 168 pidof: 223, 223
 ping: 164, 165
 ping6: 164, 165
 pinky: 187, 190
 pip3: 178
 pivot_root: 225, 228
 pkgconf: 126, 126
 pkill: 223, 223 pl2pm:
 167, 168 pldd: 100,
 106 pmap: 223, 223
 pod2html: 167, 168
 pod2man: 167, 168
 pod2texi: 210, 211
 pod2text: 167, 168
 pod2usage: 167, 168
 podchecker: 167, 168
 podselect: 167, 168
 post-grohtml: 196,
 197 poweroff: 234,
 234 pr: 187, 190

 pre-grohtml: 196,
 197 preconv: 196, 197
 printenv: 187, 190
 printf: 187, 190
 prlimit: 225, 228
 prove: 167, 168
 prtstat: 153, 153 ps:
 223, 224 psfaddtable:
 204, 205 psfgettable:
 204, 205
 psfstriptable: 204,
 205 psfxtable: 204,
 205 pslog: 153, 153
 pstree: 153, 153
 pstree.x11: 153, 153
 ptar: 167, 168
 ptardiff: 167, 168
 ptargrep: 167, 169
 ptx: 187, 190

 pwck: 139, 142
 pwconv: 139, 142
 pwd: 187, 190
 pwndx: 223, 224
 pwunconv: 139, 142
 pydoc3: 178
 python3: 178
 ranlib: 127, 128
 readelf: 127, 128
 readlink: 187, 190
 readprofile: 225, 228
 realpath: 187, 190
 reboot: 234, 234
 recode-sr-latin: 154,
 155
 refer: 196, 197
 rename: 225, 228
 renice: 225, 228
 reset: 149, 150
 resize2fs: 230, 232
 resizepart: 225, 228
 rev: 225, 228
 rfkill: 225, 228
 rm: 187, 190
 rmdir: 187, 190
 rmmod: 186, 186
 roff2dvi: 196, 198
 roff2html: 196, 198
 roff2pdf: 196, 198
 roff2ps: 196, 198
 roff2text: 196, 198
 roff2x: 196, 198
 routel: 202, 203
 rtacct: 202, 203
 rtcwake: 225, 228
 rtmon: 202, 203
 rtpr: 202, 203
 rtstat: 202, 203
 runcon: 187, 190
 runlevel: 234, 234
 runtest: 125, 125
 rview: 212, 214
 rvim: 212, 214
 script: 225, 228
 scriptlive: 225, 228
 scriptreplay: 225, 228
 sdiff: 193, 193
 sed: 152, 152

152 seq: 187, 190
 setarch: 225, 228
 setcap: 136, 136
 setfacl: 135, 135
 setattr: 134, 134
 setfont: 204, 205
 setkeycodes: 204,
 205 setleds: 204, 205
 setmetamode: 204,
 205 setsid: 225, 228
 setterm: 225, 229
 setvtrgb: 204, 205
 sfdisk: 225, 229 sg:
 139, 142 sh: 158, 159

 shalsum: 187, 190
 sha224sum: 187, 190
 sha256sum: 187, 190
 sha384sum: 187, 190
 sha512sum: 187, 190
 shasum: 167, 169
 showconsolefont: 204,
 205 showkey: 204, 205
 shred: 187, 190 shuf:
 187, 190 shutdown: 234,
 234 size: 127, 128
 slabtop: 223, 224 sleep:
 187, 190 sln: 100, 106
 soelim: 196, 198 sort:
 187, 190 sotruss: 100, 106
 splain: 167, 169 split:
 187, 190 sprof: 100, 106
 ss: 202, 203 stat: 187,
 191 stdbuf: 187, 191
 strings: 127, 129 strip:
 127, 129 stty: 187, 191
 su: 139, 142 sulogin:
 225, 229 sum: 187, 191
 swaplable: 225, 229
 swapoff: 225, 229

 swapon: 225, 229
 switch_root: 225,
 229 sync: 187, 191
 sysctl: 223, 224
 syslogd: 233, 233
 tabs: 149, 150 tac:
 187, 191 tail: 187,
 191 talk: 164, 165
 tar: 209, 209 taskset:
 225, 229 tbl: 196, 198
 tc: 202, 203 tclsh:
 121, 122 tclsh8.6:
 121, 122 tee: 187,
 191 telinit: 234, 234
 telnet: 164, 165 test:
 187, 191 texi2dvi:
 210, 211 texi2pdf:
 210, 211 texi2any:
 210, 211 texindex:
 210, 211 tfmtodit:
 196, 198 tftp: 164,
 165

 tic: 149, 151
 timeout: 187, 191
 tload: 223, 224
 toe: 149, 151
 top: 223, 224
 touch: 187, 191
 tput: 149, 151
 tr: 187, 191
 traceroute: 164, 165
 troff: 196, 198
 true: 187, 191
 truncate: 187, 191
 tset: 149, 151
 tsort: 187, 191
 tty: 187, 191
 tune2fs: 230, 232
 tzselect: 100, 106
 uclampset: 225, 229
 udev-hwdb: 217, 219
 udevadm: 217, 219
 udevd: 217, 219
 ul: 225, 229

229 umount: 225, 229
 uname: 187, 191
 uname26: 225, 229
 uncompress: 201, 201
 unexpand: 187, 191
 unicode_start: 204,
 205 unicode_stop:
 204, 205 uniq: 187,
 191 unlink: 187, 191
 unlz4: 113, 113
 unlzma: 111, 112
 unshare: 225, 229
 unxz: 111, 112
 updatedb: 195, 195
 uptime: 223, 224
 useradd: 139, 142
 userdel: 139, 142
 usermod: 139, 142
 users: 187, 191
 utmpdump: 225, 229
 uuid: 225, 229
 uuidgen: 225, 229
 uuidparse: 225, 229
 vdir: 187, 191 vi: 212,
 214 view: 212, 214
 vigr: 139, 142 vim:
 212, 214 vimdiff: 212,
 214 vimtutor: 212,
 214 vipw: 139, 142
 vmstat: 223, 224 w:
 223, 224 wall: 225,
 229 watch: 223, 224
 wc: 187, 191 wdctl:
 225, 229 whatis: 220,
 222 wheel: 182
 whereis: 225, 229
 who: 187, 191
 whoami: 187, 191
 wipefs: 225, 229
 x86_64: 225, 229
 xargs: 195, 195
 xgettext: 154, 155
 xmlwf: 163, 163

xsubpp: 167, 169
 xtrace: 100, 106
 xxd: 212, 214
 xz: 111, 112
 xzcat: 111, 112
 xzcmp: 111, 112
 xzdec: 111, 112
 xzdiff: 111, 112
 xzgrep: 111, 112
 xzfgrep: 111, 112
 xzgrep: 111, 112
 xzless: 111, 112
 xzmore: 111, 112
 yacc: 156, 156
 yes: 187, 191
 zcat: 201, 201
 zcmp: 201, 201
 zdiff: 201, 201
 zdump: 100, 106
 zegrep: 201, 201
 zfgrep: 201, 201
 zforce: 201, 201
 zgrep: 201, 201
 zic: 100, 106
 zipdetails: 167,
 169 zless: 201, 201
 zmore: 201, 201
 znew: 201, 201
 zramctl: 225, 229
 zstd: 114, 114
 zstdgrep: 114, 114
 zstdless: 114, 114

库文件

Expat: 170, 170 ld-
 2.41.so: 100, 106 libacl:
 135, 135 libanl: 100, 107
 libasprintf: 154, 155
 libattr: 134, 134 libbfd:
 127, 129 libblkid: 225,
 229 libBrokenLocale:
 100, 107 libbz2: 109, 110

libc: 100, 107
 libcap: 136, 136
 libcheck: 192, 192

192 libcom_err: 230, 232
 libcrypt: 137, 138
 libcrypto.so: 174, 175
 libctf: 127, 129
 libctf-nobfd: 127, 129
 libc_malloc_debug: 100, 107
 libdl: 100, 107
 libe2p: 230, 232
 libelf: 176, 176
 libexpat: 163, 163
 libexpect-5.45.4: 123, 124
 libext2fs: 230, 232
 libfdisk: 225, 229
 libffi: 177
 libfl: 120, 120
 libformw: 149, 151
 libg: 100, 107
 libgcc: 143, 147
 libgcov: 143, 147
 libgdbm: 161, 161
 libgdbm_compat: 161, 161
 libgettextlib: 154, 155
 libgettextpo: 154, 155
 libgettextsrc: 154, 155
 libgmp: 130, 131
 libgmpxx: 130, 131
 libomp: 143, 147
 libprofng: 127, 129
 libhistory: 116, 117
 libhwasan: 143, 147
 libitm: 143, 147
 libkmod: 186
 liblsan: 143, 147
 libltdl: 160, 160
 liblto_plugin: 143, 147
 liblz4: 113, 113
 liblzma: 111, 112
 libm: 100, 107
 libmagic: 115, 115
 libman: 220, 222
 libmandb: 220, 222
 libmcheck: 100, 107
 libmemusage: 100, 107
 libmenuw: 149, 151
 libmount: 225, 229
 libmpc: 133, 133
 libmpfr: 132, 132
 libmvec: 100, 107
 libncurses++w: 149, 151
 libncursesw: 149, 151
 libnsl: 100, 107 libnss_*:
 100, 107 libopcodes: 127,
 129 libpanelw: 149, 151
 libpcprofile: 100, 107
 libpipeline: 206
 libpkgconf: 126, 126
 libproc-2: 223, 224 libpsx:
 136, 136 libpthread: 100,
 107 libquadmath: 143, 147
 libreadline: 116, 117
 libresolv: 100, 107 librt:
 100, 107 libsframe: 127,
 129 libsmartcols: 225, 229
 libss: 230, 232 libssl.so:
 174, 175 libssp: 143, 148
 libstdbuf: 187, 191
 libstdc++: 143, 148
 libstdc++exp: 143, 148
 libstdc++fs: 143, 148
 libsubid: 139, 142
 libsupc++: 143, 148
 libtcl8.6.so: 121, 122
 libtclstub8.6.a: 121, 122
 libtextstyle: 154, 155
 libthread_db: 100, 107
 libtsan: 143, 148 libubsan:
 143, 148 libudev: 217, 219
 libutil: 100, 107 libuuid:
 225, 229 liby: 156, 156 libz:
 108, 108 libzstd: 114, 114
 preloadable_libintl: 154,
 155

脚本

checkfs: 239, 239
 cleanfs: 239, 239
 控制台: 239, 239
 配置中: 252

控制台: 239, 239
 配置: 252
 启动时文件创建配置:
 255
 功能: 239, 239
 关机: 239, 239
 主机名配置: 248
 ifdown: 239, 239
 ifup: 239, 239
 ipv4 静态地址: 239,
 239
 本地网络: 239, 239
`/etc/hosts`: 248
 本地网络: 239, 239
`/etc/hosts`: 248
 模块: 239, 239
 挂载文件系统: 239,
 239
 挂载虚拟文件系统:
 239, 239
 网络: 239, 239
`/etc/hosts`: 248
 配置: 247
 网络: 239, 239
`/etc/hosts`: 248
 配置: 247
 网络: 239, 239
`/etc/hosts`: 248
 配置: 247
`rc`: 239, 239
 重启: 239, 239
 发送信号: 239, 239
 设置时钟: 239, 239
 配置: 251
 设置时钟: 239, 239
 配置: 251
 交换分区: 239, 240
 系统控制: 239, 240
 系统日志: 239, 240
 配置: 255
 系统日志: 239, 240
 配置: 255
 模板: 239, 240
`udev`: 239, 240
`udev` 重试: 239, 240
`udev`: 127, 128
其他
`/boot/config-6.13.4`: 264, 269
`/boot/System.map-6.13.4`: 264,
 269 `/dev/*`: 78 `/etc/fstab`: 262
`/etc/group`: 81 `/etc/hosts`: 248
`/etc/inittab`: 250 `/etc/inputrc`:
 259 `/etc/ld.so.conf`: 105 `/etc/lfs-`
`release`: 273 `/etc/localtime`: 104
`/etc/lsb-release`: 273
`/etc/mke2fs.conf`: 231
`/etc/modprobe.d/usb.conf`: 269
`/etc/nsswitch.conf`: 104 `/etc/os-`
`release`: 273 `/etc/passwd`: 81
`/etc/profile`: 257 `/etc/protocols`:
 99 `/etc/resolv.conf`: 248
`/etc/services`: 99
`/etc/syslog.conf`: 233 `/etc/udev`:
 217, 219 `/etc/udev/hwdb.bin`: 219
`/etc/vimrc`: 213 `/run/utmp`: 81
`/usr/include/asm-generic/*.h`:
 52, 52 `/usr/include/asm/*.h`: 52,
 52 `/usr/include/drm/*.h`: 52, 52
`/usr/include/linux/*.h`: 52,
 52 `/usr/include/misc/*.h`:
 52, 52
`/usr/include/mtd/*.h`: 52,
 52 `/usr/include/rdma/*.h`:
 52, 52
`/usr/include/scsi/*.h`: 52, 52
`/usr/include/sound/*.h`: 52,
 52 `/usr/include/video/*.h`:
 52, 52
`/usr/include/xen/*.h`: 52, 52
`/var/log/btmp`: 81
`/var/log/lastlog`: 81
`/var/log/wtmp`: 81